Microsoft EDW Architecture, Guidance and Deployment Best Practices

# Chapter 4 - Database Administration

**By Microsoft Corporation**

**Acknowledgements:**

**Contributing writers from Solid Quality Mentors**: Ron Talmage

**Technical reviewers from Microsoft**: Ross LoForte, Benjamin Wright-Jones, Brian Mitchell, Adrian Wilson

**Technical reviewers from Solid Quality Mentors**: Gianluca Hotz, Larry Barnes

**Contributing editors from Solid Quality Mentors**: Kathy Blomstrom

**Published:**

**Applies to:** SQL Server 2008 R2

[Chapter 4 - Database Administration 1](#_Toc274454041)

[Introduction 4](#_Toc274454042)

[Context 4](#_Toc274454043)

[Read and Write Patterns in a Relational Data Warehouse 5](#_Toc274454044)

[Contrast with Typical OLTP Read and Write Usage 6](#_Toc274454045)

[Addressing Large Read Volumes 7](#_Toc274454046)

[Data Warehouse DBA Essentials 8](#_Toc274454047)

[Working on a DBA Team 8](#_Toc274454048)

[Knowledge Transfer 10](#_Toc274454049)

[Operational Practices 10](#_Toc274454050)

[Working with a Data Warehouse Database Server 11](#_Toc274454051)

[Documenting the Database Server Hardware 11](#_Toc274454052)

[Server Storage 15](#_Toc274454053)

[Pre-Installation Guidelines 19](#_Toc274454054)

[Installation Guidelines 21](#_Toc274454055)

[Post-Installation Steps 26](#_Toc274454056)

[Ensuring Best Practices for Configuration 28](#_Toc274454057)

[Managing Databases 28](#_Toc274454058)

[Choosing a Database Recovery Model 28](#_Toc274454059)

[Database Configurations 30](#_Toc274454060)

[Staging Databases 32](#_Toc274454061)

[Production Databases 32](#_Toc274454062)

[Partitioning Large Fact Tables 35](#_Toc274454063)

[Work Databases 41](#_Toc274454064)

[Database Server Security 43](#_Toc274454065)

[Security Strategies 43](#_Toc274454066)

[Querying Other Servers 45](#_Toc274454067)

[Maintaining and Monitoring Security 47](#_Toc274454068)

[Managing Database Change 48](#_Toc274454069)

[Change Management Tools 49](#_Toc274454070)

[Version Deployment 49](#_Toc274454071)

[Business Continuity 51](#_Toc274454072)

[Planned Outages 51](#_Toc274454073)

[Unplanned Loss of Service 52](#_Toc274454074)

[SLAs, RPO, and RTO 53](#_Toc274454075)

[Reliability, Redundancy, and Recovery 54](#_Toc274454076)

[High Availability and Disaster Recovery 54](#_Toc274454077)

[Recovery Model–Independent Approaches to High Availability and Disaster Recovery 55](#_Toc274454078)

[Recovery-Model-Dependent Approaches to High Availability and Disaster Recovery 57](#_Toc274454079)

[Backup Planning 58](#_Toc274454080)

[Maintenance 59](#_Toc274454081)

[Backup 61](#_Toc274454082)

[Index Maintenance 61](#_Toc274454083)

[Integrity Checks (DBCC) 61](#_Toc274454084)

[Monitoring Tools 62](#_Toc274454085)

[Conclusion and Resources 63](#_Toc274454086)

[Resources 63](#_Toc274454087)

## Introduction

A relational database system such as SQL Server 2008 serves as the usual—and nearly universal—basis for relational data warehouses. SQL Server’s query engine is mature and well understood, it supports storage for very large data sets, and the standard data warehouse database designs involve structured data that works well with SQL Server.

Relational data warehouses often have a different structure and behavior than OLTP database systems, using denormalization techniques and unique load and query patterns. As a result, many of the usual practices for administering relational data warehouses require a distinct slant and approach that reflects these diverse aims.

In this chapter, you’ll learn about the major challenges that a database administrator of a relational data warehouse must address, with an emphasis on the unique requirements of data warehousing.

The following topics in this chapter build on the previous three chapters and lead into Chapter 5 on querying, monitoring, and performance tuning:

* We will discuss implementing the design patterns and practices from Chapter 2, “Data Architecture,” as they relate to database administration and operations management.
* Because a large part of the activity of a relational data warehouse is loading and sometimes transforming data, we will look at how to manage the database system so that ETL best practices regarding database usage can be implemented successfully.
* Finally, we will look at how, by ensuring that the database system is stable and has an optimized infrastructure, DBA best practices make it possible to fine-tune the performance of the relational system using the techniques covered next in Chapter 5, “Querying, Monitoring, and Performance Tuning.”

## Context

How does a relational data warehouse look to a DBA? How is it different from an OLTP system? One way to approach this issue is to look at the read and write usage patterns that often or normally occur in a data warehouse.

As noted in Chapter 1, competing requests for data from the data warehouse lead to a typical cycle of activity called data warehouse access patterns. You may recall Figure 4-1, repeated from Chapter 2, which shows the major components in a representative, somewhat typical daily cycle.



**Figure 4-1:** Data Warehouse access patterns

Let’s take a closer look at the type of read and write volumes that generally occur during this daily cycle and compare it with those of an OLTP system.

### Read and Write Patterns in a Relational Data Warehouse

Figure 4-2 shows a simplified view of a relational data warehouse usage pattern in terms of relative amounts of read versus write data volumes.

**Figure 4-2:** Typical read and write patterns in a relational data warehouse

Note the following:

* The read data volume is much larger than the write data volume during user access periods.
	+ The read and write data volumes do not rise and fall proportionally.
* During the period of normal user access, read data volumes far outstrip write volumes.
	+ During user access, there may be some write activity as users save data to work databases.
	+ However user access remains primarily read activity, not in terms of the number of queries, but in terms of the volume of data.
* The Load and Maintenance period has a much higher write volume, primarily due to load activity.
	+ This is simplified in the diagram to make it easier to read. Load and maintenance tasks might or might not occur together, depending on the tasks and databases in question.
		- When many databases are involved, some could be backed up or undergo index maintenance while others are being loaded, and vice versa.
	+ Write activity may fluctuate widely from one task to the next.
	+ Load activity typically involves reads of data to determine ranges of missing rows and may also update some data. Load activity may also update rollback tables and ETL logging tables.
	+ Backups are typically heavy on reads.
	+ Index maintenance involves both reads and writes.
	+ The Load/Maintenance window may be treated differently from one installation to the next.
		- Some installations may insist that no load occur until after all backups are completed.
		- Normally, indexing maintenance would occur after loads to help optimize for upcoming queries during user access.
* The diagram illustrates a typical weekday; weekends might differ dramatically if user access is absent and more backup and index maintenance is performed.

### Contrast with Typical OLTP Read and Write Usage

What’s illuminating is to compare that data volume with a (somewhat) typical OLTP system. It is difficult to generalize about OLTP systems because there are so many different designs and applications. But within reason, Figure 4-3 shows what the read and write volumes might look like in a moderately normal OLTP system.

**Figure 4-3:** Typical usage patterns for an OLTP system

Note the following:

* The read and write data volumes will tend to rise and fall together proportionally.
	+ The applications accessing an OLTP database might still read more data than they write.
	+ The ratio of read-to-write volume remains the same in each function of the application.
	+ Often only the frequency of operations increases or decreases.
* Also, OLTP systems often offload reporting or catalog reading activities to other servers by replicating or otherwise distributing data to read-only servers.
	+ This helps keep OLTP read and write volumes roughly proportional.
* If the OLTP system does not require constant uptime (24x7), it might have an available maintenance window, as suggested in Figure 4-3.
	+ In that case, read and write volumes would accordingly differ based on the backup and index maintenance tasks performed.
* However, many OLTP systems do not have maintenance windows at all, forcing maintenance to be done as an ongoing process.
	+ Index maintenance may be performed using the ONLINE options in SQL Server 2008.
	+ Backups may proceed during user access periods.
	+ In this case, the read and write data volumes might comparatively flatten out, but will still preserve a roughly proportional ratio of read-to-write volume.

### Addressing Large Read Volumes

In summary, as a DBA you need to be prepared to deal with a database system that will often have a disproportionately high ratio of read volume to write volume. Depending on the system, this focus will permeate all of your DBA tasks. Indeed, the larger the data warehouse, the more the high read volume affects everything.

This chapter addresses the needs of a DBA in charge of a relational data warehouse by covering the following areas:

* Database administration essentials
* Hardware configuration – Ensuring hardware is ready and benchmarking I/O
* Software configuration – Windows Server and SQL Server installation and configurations
* Managing storage – Working with drive volumes and placing multiple data files and filegroups
* Change management – Deploying new and updated database objects
* Security – User organization and permissions assignment
* Database maintenance – Backup, load, and index-related tasks

## Data Warehouse DBA Essentials

There are many factors that contribute to successful database administration, so many that it’s probably impossible to list them all. Nevertheless, there are certain best practices and guidelines that might seem obvious when you read them, but can be challenging to put into practice. These DBA essentials result in a more professional approach to database administration, and most important, they contribute to a successful data warehouse.

### Working on a DBA Team

Every data warehouse DBA in an enterprise of any significant size works on a team. The type of team may differ from one situation to another, but the work is simply too much for one person. However, DBAs are notoriously strict and scrupulous about their own actions and ideally take full responsibility; with that much self-focus, how can they work in a team?

**Working with Fellow DBAs**

A well-functioning DBA team has members who:

* Are good communicators. Good team members make sure the team knows about all planned changes to servers, configurations, standards, and data. They ensure that all planned actions are clear to those on the team that need to know about them.
* Request feedback from each other. When tasks require extensive research or testing, time may be saved by airing the issue and benefiting from other team members’ experience and knowledge.
* Inform each other about server or database emergencies and how long the emergency lasts.
* Keep a good calendar. As Woody Allen says, “Eighty percent of success is showing up.” So it’s important to make the important meetings and conference calls that keep other members of the DBA team informed.
* Manage time well and prioritize tasks. Good team members have a handle on their own tasks and what their tasks are in priority order. Setting priorities that mesh well with the DBA team can help members say no to requests that might otherwise be distracting and a lower priority.

**Virtual Teams and Committees**

Virtual teams of people are almost standard in large organizations today. In nationwide and global enterprises, company personnel and data are often geographically distributed, leading to the importance of working on teams and committees that are not physically located in one place.

To work well on a virtual team:

* Attend the regular conference calls and virtual meetings (LiveMeetings, etc.).
* Travel occasionally to attend group meetings.
	+ Often, virtual teams get together periodically to confer in one place. This helps with team-building and resolving issues that might not otherwise be addressed.
	+ Equally important are meetings at the close of a project or a major deployment, where colleagues debrief each other.
* Balance the methods of communication. Sometimes a phone call is better than an email or a chat dialog. When multiple people on a particular email thread don’t need to be involved, you might take some action offline, phone a colleague and work together to complete a task, and just report the result back to the main email thread, thereby reducing everyone’s Inbox.
* Log off remote desktop sessions on production servers when work is finished. Remote desktop is one of the virtual DBA’s most important tools, but leaving remote desktop sessions running on production database servers may be impolite to other DBAs if the maximum number of sessions is reached—not to mention the memory it might steal from the server. In addition, you could leave processes running on such a session that are unknown to other DBAs on your team.

**Working with Other Colleagues**

Here are other colleagues that DBAs need to have good working relationships with:

* **Superiors.** DBAs need to inform superiors and managers of database administration activities that have an impact on the business or on projects that the DBA team is involved in. This communication should follow the normal chain of command unless the situation demands otherwise.
* **Developers**. The goals and interests of DBAs and developers are not always the same. DBAs naturally want to protect data and are reluctant to change normal operations without careful preparation. Developers want to finish their projects as quickly as possible and on time, and do not want difficulties with underlying database objects to impede their progress. It’s important for DBAs to communicate the reasons for their actions and work together with developers to provide timely but safe results.
* **Domain administrators.** SQL Server DBAs are dependent on domain (or network) administrators for a number of things, including provisioning SQL Server service accounts, configuring servers and accounts for Kerberos, and creating groups for database users. Good DBAs know who they can contact for problems relating to those issues. In many cases, network administrators may not know Kerberos well or be able to diagnose it, so DBAs must be able to step in and ensure that Kerberos is set up correctly.
* **Storage Area Network (SAN) administrators**. The SAN is the most important external component of a data warehouse. It’s important to work with SAN engineers and understand their perspective when setting up and managing the SAN. It is also important that the SAN administrators understand the DBAs’ standpoint, both for I/O performance requirements and SQL Server transactional I/O patterns. When the SAN does not perform as expected, you must be prepared to challenge the SAN engineer.

**Vendor Technical Support**

DBAs often have to interact with software vendors, including:

* **Microsoft.** A good DBA should know how to research issues regarding SQL Server efficiently and recognize problems that require the help of Microsoft support.
* **Application vendors.** When databases are not originated in-house, as is often the case with vertical market applications, the DBA team must know what they can and cannot change in terms of database objects and configurations. It’s important to build up technical knowledge of how to work around issues that might be more directly addressed if the application originated in-house.
* **Tools Vendors.** Third-party tools may occasionally present problems that the DBA can help resolve by contacting technical support for purchased tools.

### Knowledge Transfer

As part of a team, no single DBA should have unique knowledge of the data warehouse system. Some useful ways of sharing and transferring knowledge include:

* **Useful documentation.** It’s important to achieve a balance in documentation, not spending too much time on it but including all the essential information in a concise way that others can easily read and use. You also need to keep the documentation updated. Keeping good documentation should not take a lot of time. The key is to make the documentation brief but usable.
* **Knowledge Base.** It can help to develop or contribute to an internal knowledge base in which you can record lessons learned that have special import for your environment.
* **Server and database events.** When a server or database outage or severe event occurs, document the essentials of the event for future reference. Some organizations have internal knowledge bases, somewhat like Microsoft’s Knowledge Base. But even when the technical issue is not particularly difficult or unusual, just knowing when and how often certain types of server, SAN, or database events occur can be helpful for future planning.
* **DBA Run Book.** Traditionally, a DBA Run Book is a record of where all the software is located that might be required to recreate a database server, along with license keys and installation instructions.
* **Logging changes.** DBAs should log all major changes to productions servers, databases, and database objects. When done carefully and regularly, the log record can help troubleshoot problems that occur in the future as well as show when DBA activities are not related to a particular event.

### Operational Practices

Probably the most important imperative for the data warehouse DBA is to protect production data. A number of practices, guidelines, and technologies flowing from this imperative are discussed in this chapter, but here’s a brief summary of what a DBA should focus on:

* **Make changes to production data with care.** The DBA has system administrator access to production databases, so all changes to production data must be made with extreme care and with a plan for rolling back changes if necessary.
* **Maintain a test environment.** The DBA team needs to have a test environment where team members can test change scripts before applying them to production systems.
* **Ensure that backups are current and periodically restored.** Data warehouse databases are key to the success of the enterprise, so it’s vital that the DBA team makes sure that the data is appropriately backed up. The backups must be restored on a regular basis to verify their integrity.

All of these operational practices will be covered in more depth in the next section.

## Working with a Data Warehouse Database Server

When you are first charged with the responsibility for managing a database server, it’s important to extract essential information about the database server and the SQL Server instance. You can get some basics from simple tools such as Task Manager on the server and the Server Properties dialog box in SQL Server Management Studio (SSMS). But how can you get more detailed information in a way that’s usable for inserting into documentation? In addition, when you have to build a database server by installing SQL Server, what steps are critical for a relational data warehouse? Let’s look at the answers to these questions.

### Documenting the Database Server Hardware

It’s important when first starting work on a database server to gather and document fundamental information about the server’s hardware configuration, and its Windows Server version.

**System Information**

One excellent source of extensive and usable documentation is the System Information dialog box in Windows. You can invoke it from Control Panel or just run MSInfo32.exe. The System Summary dialog box contains a wealth of information, as shown in Figure 4-4. This screen shot is taken from a Windows Server 2008 R2 server.



**Figure 4-4:** Example of the System Summary dialog box displayed by running System Information (msinfo32.exe)

Note that the System Summary includes:

* The operating system, edition, build, and version
* A list of CPUs, logical and cores, and CPU speed
* Total physical memory
* Page file space and location
* The ability to export the summary data to save it as documentation

For information about performance tuning and Windows Server, see the white paper [Performance Tuning Guidelines for Windows Server 2008](http://www.microsoft.com/whdc/system/sysperf/Perf_tun_srv.mspx).

In addition, you can document basic SQL Server relational engine information by saving the results of sp\_configure. First, make sure that “show advanced options” is set to 1, and then run the following command:

EXEC sp\_configure

You can copy the output as a grid and paste to Microsoft Excel or Word, or you can output to text and then paste the results into Word to see the textual results. You can also collect configuration information by exception by using the SQL Server 2008 R2 Utility Control Point, described later in this chapter.

**Rack Position**

If you have access to the data center or the documentation for the data center, document the rack positions of the database servers. You might consider producing a graph as well. This exercise can help you determine what servers are available, which are in use and for what purpose, and who is in charge of the servers at the data center.

If possible, make a visit to the data center. Even if you never go back, it can be helpful to know the physical layout of the servers in their rack. It can also be helpful to talk with the people who maintain the servers on a daily basis.

Such server rack documentation can come in handy if you need to have a physical action performed that cannot be done remotely and must be done in off hours. For example, if the server needs to be restarted and the remote software will not work, there may be no choice but to find the server in the data center and physically operated the power switch. If you know where the server is located and who is in charge, it can save you a lot of time getting the required action performed.

**CPU**

The System Summary shows you the number and speed of CPUs on the database server. But how can you know ahead of time whether your database server has enough CPU power? It is difficult to predict CPU needs unless you have some kind of baseline. For example, you might have a server with 16 CPUs across 4 cores, and you know that server is under-powered. As a result, you can make sure a new replacement server has more and faster CPUs.

If you have a data warehouse in place and you can reasonably estimate further growth, you may be able to produce an estimate of the required server resources such as CPU, memory, and disk space. For one example of making such an estimate, see the “FTDW CPU Core Calculator” in the Microsoft white paper [Fast Track Data Warehouse 2.0 Architecture](http://msdn.microsoft.com/en-us/library/dd459178.aspx).

**Note:** The System Information dialog box, and Windows Server dialog boxes in general, do not detect whether Hyperthreading is enabled on the sever. You can use other utilities such as Intel’s CPUCount.exe to help determine Hyperthreading status. For more information, see the Microsoft Knowledge Base (KB) article [SQL Server support in a hyper-threaded environment](http://support.microsoft.com/kb/322385).

**Memory**

SQL Server uses the database server for memory cache, and relational data warehouses typically have queries that involve large amounts of data. When SQL Server can cache the requested data, queries can respond faster. Therefore, it’s important to make sure the system has sufficient memory.

Estimating memory needs can be just as difficult as measuring CPU requirements. Again, if you have a baseline server that can give you an approximation of the activity you can anticipate, you can use that as a basis for making a thumbnail estimate of the memory your server may need. In any case, it is always wise to have more memory than absolutely required as well as having the capability of adding more memory in the future if required.

Here are some memory guidelines to follow:

* Ensure sufficient memory so that SQL Server will have sufficient memory for the data cache. On a data warehouse server, the bulk of memory will be used for the data cache.
* Use 64-bit Windows Server and SQL Server for better large-memory management. For database servers, 32-bit memory is simply too cumbersome.
* Make sure the editions of Windows and SQL Server will support the amount of RAM on the server that will be allocated to it. There are upper limits to the amount of memory the various editions of Windows will expose to SQL Server. For details, see [Editions](http://www.microsoft.com/sqlserver/2008/en/us/editions.aspx) on the SQL Server 2008 web site.

**Note**: To determine whether a SQL Server instance has enough memory to successfully cache data required for queries, you can observe a system over a period of time using the System Monitor (Perfmon) SQL Server counter called Buffer Manager:Page Life Expectancy. This counter returns the average number of seconds a page of data stays in the SQL Server data cache before being replaced by a newer page.

A good rule of thumb is that an average value less than 500 seconds is an indication that the server is under memory pressure. A healthy server should average numbers in the thousands. An extremely high average number, in the hundreds of thousands, could indicate an under-utilized server. While the average might dip below 500 seconds during heavy utilization, it should not stay below 500 for an extended period of time.

You may also want to cross-check with the values of Buffer Manager:Free pages in memory and compare BufferManager:Target pages, Buffer Manager:Total pages, and Memory: Available Mbytes .

**Network Cards**

To return data to client applications, SQL Server must send that data across the network. So it’s important for the database server to have the proper network interface cards (NICs) in it. Generally speaking, follow these guidelines:

* Multiple redundant NICs are useful if one fails.
* Multiple NICs normally should be identical to each other.
* Ensure that the NICs have sufficient speed to match the network switches.
* If the database server is clustered, multiple NICs are required and the public network cards may be teamed for redundancy.

**Virtualization**

Virtualized servers are becoming more common because virtualization software has matured and users have gained confidence in its reliability. However, it usually makes sense to virtualize a server when you plan to use it for multiple virtual machines. No matter how mature and efficient virtualization becomes, for a single server, it will insert an extra layer between the hardware and the operating system, which is a penalty for large relational database servers. For more information on virtualization, see the [SQLCAT whitepaper: Running SQL Server 2008 in a Hyper-V Environment - Best Practices and Performance Recommendations](http://sqlcat.com/whitepapers/archive/2008/10/03/running-sql-server-2008-in-a-hyper-v-environment-best-practices-and-performance-recommendations.aspx)

Generally speaking, for enterprise-level data warehouses, virtualization should be avoided for the following reasons:

* A relational data warehouse is likely to need all the hardware resources available on a physical server, so there is no advantage to adding a virtualization layer.
* Virtual machines typically limit the number of processors, whereas an enterprise data warehouse will likely require all of a server’s physical CPU resources.
* For a relational data warehouse, many of the benefits of virtualization are lost because the server will have only one primary function, as a database server.

If you must use virtualization on an enterprise-level data warehouse, make sure you thoroughly research the issue to understand any potential pitfalls. Enterprise data warehouse systems tend to “push the envelope” of all the technologies, and you don’t want to risk configurations that virtualized systems can barely support.

### Server Storage

Probably the most important consideration a DBA has for a relational data warehouse is the placement of database data. While you might place some database data on the local server for some reason, normally the large database files must be placed on an external storage device, typically a SAN. Figure 4-5 shows a typical relational data warehouse storage configuration.



**Figure 4-5:** A typical storage configuration for a relational data warehouse

**Uses of Local Storage**

The local drives on an enterprise-level database server should always be configured for some level of RAID. Some considerations to keep in mind:

* Given the large drives available with servers today, two disks formatted for RAID 1 usually provide a local drive with all the necessary space.
* What’s critical is that RAID be used, so that a single disk failure will not halt Windows or SQL Server.
* The physical RAID 1 enables the use of basic disks in Windows for forming local drives, which is generally superior to dynamic disks. This guarantees I/O isolation across drives.
* Most servers can be configured with at least four local disks, enabling two RAID 1 drives.

Local storage on a relational data warehouse database server tends to be used primarily for system files, such as:

* The Windows and SQL Server binaries. SQL Server’s binary files are normally placed during Setup.
* The Windows Page File, which normally should be on its own physical drive.
* The SQL Server system databases (master, model, and msdb). These system databases are usually placed locally because they are required by the SQL Server during normal startup.
* Tempdb usually installs locally and is then moved to the SAN. Tempdb configuration is discussed in the “Post Setup Configurations” section later in this chapter.

Consider that I/O to local drives is usually faster than I/O to an external drive system such as a SAN. If tempdb performance is critical, and local disks are sufficiently large and numerous, you might be able to place tempdb locally, as shown in Figure 4-6.



**Figure 4-6:** Placing the tempdb database on local drives

SQL Server recreates the tempdb database every time it restarts. It also requires the storage location of tempdb to be available. Therefore, losing tempdb on a SAN could result in a SQL Server being unable to start. You then have to start SQL Server manually and place tempdb locally to get the SQL Server instance back online. Placing tempdb locally can help avoid this issue.

In addition, you might be able to leverage new production-quality Solid State Drives (SSDs) on the local server to provide even faster I/O for tempdb.

**Note**: Microsoft does not support a local tempdb on a Windows failover cluster. For a failover cluster, tempdb should be placed on shared storage.

**External SAN Storage**

SANs are the storage system of choice for enterprise-level relational data warehouses. SANs provide an unparalleled level of redundancy and management for large numbers of drives, and they can enable easy growth of storage volumes—something impossible on other types of storage devices.

However, SANs are not typically faster than local drives or external drive arrays. SANs are not necessarily a performance solution, but rather more of a scalable disk management solution. Generally speaking, for all types of disk systems, the more disk spindles available, the better the I/O performance.

SANs are complex, and SAN administration is an entire specialty of its own, apart from network and database administration. If possible, get to know the team that will be administering your SAN, and if you are able to participate in SAN planning, get to know the SAN engineer who will be designing the SAN solution.

Try to impress upon the SAN storage designers the importance of separating the comparatively random access of database data from the sequential read and write access of the SQL Server transaction log. In many cases, SAN engineers can ensure this separation by physically arranging separate disk groups for database data and log data, so that the distinct I/O patterns are served by physically distinct disk groups. However, this is not always possible because some SAN architectures do not allow custom disk groups.

The Microsoft SQL Server Customer Advisory Team (SQLCAT) has determined that one of the most important items to check with a storage subsystem, including a SAN, is disk partition alignment, which should be set to 1024KB (using diskpart.exe). This is the default for new partitions with Windows Server 2008. Partition alignment offset should normally be set to 64KB. See the white paper [Disk Partition Alignment Best Practices for SQL Server](http://msdn.microsoft.com/en-us/library/dd758814.aspx) for details.

From a relational data warehouse DBA’s standpoint, you should ensure that:

* Data and log reads and writes are physically separated into distinct disk groups on the SAN
* Each SAN LUN (drive) spans a large number of disks (spindles)
* The HBAs and SAN support Multipath I/O; the database servers should use multiple HBAs to the SAN switches so that SQL Server can scale across multiple I/O paths to the SAN
* The SAN switches do not become a bottleneck
* You have made reasonable size and growth estimates
* SAN virtualization (with thin provisioning) is used for ease of LUN growth
* The SAN cache is optimized for read-mostly data

Also, ensure that the SAN LUNs that are exposed as Windows drives provide a sufficient number of Windows volumes. You can use drive letters, but with Windows Server 2008, mount points—an alternative to drive letters—are becoming more common. For more information see the KB article [SQL Server support for mounted volumes](http://support.microsoft.com/kb/819546).

**Note:** Windows Server 2008 can format drives that can exceed 2TB. Prior versions of Windows Server, such as Windows Server 2003, could only mount drives with a maximum size of 2TB. However, if Windows Server 2008 has a drive originally formatted with a legacy Windows Server version, converting it so that it can grow beyond 2TB will cause data to be lost. You have to move the data to another drive first before reformatting it.

SAN failures do occur, so be prepared. Know how to contact storage administrators and vendors if required. In addition, be prepared for outages during SAN and HBA firmware updates.

**Solid State Drives (SSDs)**

SSDs are faster than mechanical drives and are not generally subject to some of the issues that mechanical drives have, such as file fragmentation. While currently more expensive than mechanical disk drives, they can be very useful in a relational data warehouse. Generally speaking, they are best used when targeted to provide fast disk I/O on I/O pressure points, such as tempdb.

Some guidelines for using SSDs include:

* Use SSDs for fast storage, intermediate in speed between RAM and mechanical drives.
* Focus on using SSDs for data that must be retrieved quickly.

You should still use mechanical drives for backup, work databases, and other storage that does not demand the performance level of SSDs.

**Assign Backup Drives**

SQL Server 2008 has a robust built-in backup system, and the Enterprise Edition supports backup compression, and SQL Server 2008 R2 Standard Edition also supports backup compression. However, there are many third-party backup utilities that serve other enterprise-level needs, and often enterprises may standardize on such a backup tool. In that case, the relational data warehouse DBA must work with the third-party backup tool.

Typically, backing up SQL Server databases to disk first, with compression, is the fastest method of backup. This approach helps reduce the amount of the maintenance window that backups require. Whether using disk-to-tape or disk-to-disk-to-tape backup, ensure that you have sufficient drive space on the backup drive for all the backups.

**Benchmarking Storage**

Before putting the database server hardware into production, it is wise to get a benchmark of the storage subsystem’s I/O performance. This benchmark is helpful as a method of comparison among storage subsystems and as a way to provide feedback to the SAN engineers if a test results in numbers that indicate poor performance.

Benchmarking is best done before putting a SAN into production and with I/O patterns that simulate SQL Server relational data warehouse patterns. Benchmarking is difficult because it is often impossible to reproduce production behavior even when queries are known. Production behavior can be unpredictable, although it may be known within certain ranges. In addition, production systems change over time: data grows, new indexes are added, and configurations may change.

The best approach is to make sure the new system meets minimal performance requirements by running a benchmarking tool such as SQLIO or IOmeter. SQLIO is provided as a free download from Microsoft. It is most useful for initial tests where you:

* Run a sample set with parameters that reflect data warehouse I/O patterns
* Get a baseline from a comparable system if possible
* Run the tests when the system is idle
* Run Perfmon logs alongside to observe I/O performance

For information about running the SQLIO utility, see the white paper [Pre-deployment I/O Best Practices](http://technet.microsoft.com/en-us/library/cc966412.aspx).

**SQL Server I/O Resources**

A good starting point for learning about SQL Server storage is the Microsoft SQLCAT article [Storage Top 10 Best Practices](http://technet.microsoft.com/en-us/library/cc966534.aspx). For comprehensive information about SQL Server I/O, see the following white papers:

* [SQL Server I/O Basics](http://technet.microsoft.com/en-us/library/cc966500.aspx)
* [SQL Server I/O Basics, Chapter 2](http://technet.microsoft.com/en-us/library/cc917726.aspx)
* [Analyzing I/O Characteristics and Sizing Storage Systems for SQL Server Database Applications](http://msdn.microsoft.com/en-us/library/ee410782.aspx)

### Pre-Installation Guidelines

When building a relational data warehouse database server, you need to plan around the SQL Server installation process. This and following sections cover the pre-installation, Setup, and post-installation steps that are most common when building a database server.

Sometimes network administrators will install SQL Server to the DBA’s specifications, but often the DBA is the one who does the SQL Server install. The first important step is to make sure the Windows Server OS and the physical server are ready for a SQL Server installation. Another critical pre-install step is to list the required SQL Server components for each server:

* Itemize which components will be installed on the data warehouse or supporting server.
* Decide how much memory to initially allocate to SQL Server or SQL Server Analysis Services (SSAS), respectively.

**Note**: SQL Server and SSAS can run on the same server, but configuring both for memory usage can be challenging. Combining the two in one instance is not generally recommended if both components require large amounts of memory. As a result, in an enterprise setting, the SQL Server relational data warehouse and SSAS instance should normally be installed on separate servers. For information about configuring server memory to host both the SQL Server relational engine and SSAS, see the white paper [Microsoft SQL Server 2008 Analysis Services Consolidation Best Practices](http://sqlcat.com/technicalnotes/archive/2010/02/08/microsoft-sql-server-2008-analysis-services-consolidation-best-practices.aspx).

Here are some guidelines for preparing the database server for SQL Server installation after the Windows Server OS is installed:

* Make sure the database server is named correctly. This usually means following company-wide naming standards.
	+ If you have the option of choosing a server name and already have a set of standard naming patterns, choose one that is descriptive and fits the pattern.
	+ If you do not yet have a server naming standard, choose one that will be flexible, meaningful, and easily understood.
	+ A common naming pattern involves a project or server function (such as DW), followed by region or data center location, and then a number. It can be useful to make the number the last octet of the server’s IP address to assist in remembering both.
* Make sure the proper Windows Server OS is installed.
* To install SQL Server, your account should have temporary Local Administrator rights on the server.
* Make sure the current MS Installer is on the server.
* Install Microsoft .NET Framework 3.51 SP1. This saves a little time during the installation process.
* Most likely, you will install SQL Server via remote desktop. Make sure the server is enabled for remote desktop and that the DBA group has remote desktop access.
* If Kerberos authentication is required, check with the network administrators that the server has been enabled for Kerberos authentication.

Some steps related to SQL Server service accounts include:

* Request service accounts for the required SQL Server components from network administrators.
* Choose the SQL Server domain for the accounts carefully. Normally, it would be the same domain as the servers, but sometimes there may be a special domain for the data warehouse project.
* Request distinct accounts for distinct SQL Server components.
* Add the Windows SQL Server service account to the Perform Volume Maintenance Tasks security policy to enable fast file initialization.
	+ With fast file initialization, the potential exists to disclose deleted data, so ensure that the conditions under which this might take place are not a concern in your environment. See the SQL Server Books Online (BOL) article [Database File Initialization](http://msdn.microsoft.com/en-us/library/ms175935.aspx) for details.
* Add the Windows SQL Server relational engine service account to the *Lock pages in memory* policy. Although not required for 64-bit SQL Server, it is nonetheless recommended. See the KB article [How to reduce paging of buffer pool memory in the 64-bit version of SQL Server](http://support.microsoft.com/kb/918483).
* If Kerberos is required and the server is enabled for Kerberos authentication, SQL Server 2008 will automatically register Server Principal Names (SPNs) for the SQL Server instance as long as the service account has the appropriate permissions in Active Directory.
	+ For details about using Kerberos in SQL Server, see:
		- [How to troubleshoot the "Cannot generate SSPI context" error message](http://support.microsoft.com/kb/811889)
		- [How to use Kerberos authentication in SQL Server](http://support.microsoft.com/kb/319723)
	+ For more information about SQL Server and SPNs, see the SQL Server BOL article [Registering a Service Principal Name](http://msdn.microsoft.com/en-us/library/ms191153.aspx).

Some additional Windows Server configurations to be done or checked by the DBA:

* Disable the firewall temporarily for the install process.
* Enable the server for the Application role.
* Turn enhanced security off in Internet Explorer for administrators.
* Turn User Account Control (UAC) off to remove frustrating confirmation screens.
* Check for pending Windows updates, and apply them with a server restart if necessary.

### Installation Guidelines

You install SQL Server components using SQL Server Setup.exe. Although Setup.exe is a very user-friendly tool, there are just a few places that are subtle enough to merit some tips and guidelines.

**Use a Configuration File**

Even though Setup.exe is a set of wizard-like dialog boxes, it records your choices in a configuration file that is automatically created at the end of the installation process. This file is named Configuration.ini by default although you can rename it; just make sure you keep the .ini suffix.

Using a configuration file to install SQL Server consists of two steps: generating the file by using the Setup.exe dialogs, and then choosing the configuration file later to start the installation. The first step, generating the file, is done at the *Ready to Install* dialog box, just before you click Install to commit to the installation. The location and name of the file is shown at the bottom of the dialog box, as shown in Figure 4-7. You can cancel the installation at this point and the named file will still be saved to disk with all the options you chose.



**Figure 4-7:** Automatically generating a configuration file in the final Setup dialog box

Once you’ve created the configuration file, you can return to the SQL Serve Installation Center dialog box and select the Advanced pane, which gives you the option for installing based on a configuration file, as Figure 4-8 shows.



**Figure 4-8:** Installing interactively using a configuration file

There are several advantages to using a SQL Server Setup configuration file:

* The configuration file documents the choices you made for the installation.
* Configuration files are easy to clone to repeat the same installation on another server.
* The configuration file helps eliminate mistakes that might be made while trying to repeat the Setup.exe GUI choices from one server to another.

You can run Setup using a configuration file interactively or as an unattended install. When you use it interactively, Setup will automatically populate each dialog box with the configuration file information. But you still have to step from one dialog box to the next.

As an unattended install, Setup runs from the command line with numerous parameters, as documented in the BOL article [How to: Install SQL Server 2008 R2 from the Command Prompt](http://msdn.microsoft.com/en-us/library/ms144259.aspx).

Some guidelines when building a configuration file include:

* Don’t put the actual service accounts, with their passwords, into the configuration file.
	+ Instead, use the NT Authority/System account for the default service accounts.
	+ Later, add the service accounts using the SQL Server Configuration Manager.
	+ This can be convenient when only the network administrators know the passwords.
* Choose to install initially with Windows-only authentication in the configuration file.
	+ You don’t have to record the SQL Server sa password in the configuration file.
	+ You can change to mixed authentication later if desired and enter the sa password interactively.
	+ SQL Server Setup will automatically assign a random strong password even if you are using Windows-only authentication.
	+ You may want to supply your own strong sa password and then later disable the sa account.
* Always choose the Books Online component.
	+ You may have to remote desktop to the server and need BOL in an emergency situation.
* Also install SSMS (the Management Tools component) and Business Intelligence Development Studio (BIDS).
	+ The exception to this guidance is if either might violate company or departmental standards.
	+ Again, when using remote desktop to the server, having the client tools available can be very useful.

You can also edit the configuration file in a text editor or Notepad, and use the new version on a different install. Figure 4-9 shows a sample configuration file in Notepad.



**Figure 4-9:** Editing a configuration file in Notepad

For example, you can edit the FEATURES line to include or exclude components. For more information about the configuration parameter names and values, see the BOL article [How to: Install SQL Server 2008 R2 from the Command Prompt](http://msdn.microsoft.com/en-us/library/ms144259.aspx).

**File Placement: Data Directories**

During the installation process, SQL Server Setup.exe gives you the option of setting default file locations for data directories.

**Note:** Before you arrive at Data Directories, you can change two other file locations. The first file location you can change is that for the Shared Component Directory, the file location for all shared components. You can specify this change at the bottom of the Feature Selection page. The second changeable file location is the Instance Root directory, which you can change on the Instance Configuration dialog box. However, unless you have a good reason for changing the locations for these files, they should be left at the default values.

You can change default file locations for various data directories on the Data Directories tab of the Database Engine Configuration screen, shown in Figure 4-10.



**Figure 4-10:** Changing data directories in Setup

It’s important to take a minute to consider this dialog box because making the right choices here could save a lot of extra work later. Here is a brief summary of the various directories and options for them:

* **Data root directory.** This is the directory for the system tables (master, msdb, and model).
* **System database directory.** This is the directory for the Resource database named mssqlsystemresource, and it is not changeable. Therefore, SQL Server will install some of its required files on the system drive (usually C:) no matter what.
* **User database and log directories.** These are the default directories for database data and log files, respectively. If you plan to use multiple files or filegroups, the User database directory will be the location of the PRIMARY filegroup. Always provide separate default locations for database data and the database log.
* **Temp DB database and log directories.** Usually tempdb is assigned its own file locations on different drives from the database data and log files. You will normally add additional files to that location after installation. Even though tempdb does not use the transaction log quite as intensely as other databases, because tempdb is never recovered, it is still wise to place tempdb log and data on separate drives for performance reasons.
* **Backup directory.** This should be a distinct location—a drive dedicated for backups. The disks may be slower disks than those used for data warehouse database data, but you need to ensure that there will be sufficient disk space for the backups based on your backup schedule.

**Imaging**

If you can coordinate with system administrators, you might be able to use the SysPrep utility to prepare an image of SQL Server 2008 R2 along with the Windows Server image. This image would be useful when you have to install the same or roughly the same SQL Server configuration on multiple database servers.

SQL Server 2008 R2 supports the use of SysPrep for the database engine in a way that installs the binaries but does not require configuration of a server name. For more information, see the BOL article [How to: Install SQL Server 2008 R2 Using SysPrep](http://msdn.microsoft.com/en-us/library/ee210664.aspx).

### Post-Installation Steps

After SQL Server is installed on a relational data warehouse server, there are usually a number of actions that remain to be performed. In this section, we look at some of these key post-installation tasks.

**Apply Service Accounts**

One of the first things to do after installing SQL Server is to apply the assigned domain service accounts to the SQL Server services on the database server. This should be done using the SQL Server Configuration Manager, which sets all appropriate file and ACL permissions during the operation.

You should also then enable (and possibly disable) the services appropriately and set their start modes. If only the default instance of SQL Server is installed on the server, you can disable the SQL Browser service.

**Server Configuration**

For server configuration, there are a number of defaults you can configure immediately, as follows: SQL Server maximum memory

* Processor affinity (if any)
* Authentication type
	+ If choosing mixed authentication, either use the randomly assigned strong sa password from Setup or supply your own strong password.
	+ Rename and then disable the sa account, no matter what the authentication mode, to ensure it cannot be hacked.
* Max degree of parallelism (MAXDOP)
	+ Normally, this property is set in increments of 4. However, note that this value can be overridden by users if they supply a MAXDOP query hint. Only the degree of parallelism option in Resource Governor can override the query hints; see the “Resource Governor” section later in this chapter for details.
* Backup compression default

**Note:** If you are going to use database chaining, you might be tempted to enable it at the server level. However, it is a better practice to leave database chaining disabled at the server level and enable it only for the databases that require it. See “Database Configurations” in the next section for further discussion

**The Model Database**

You may also want to configure the model database. Because model is used as a template for all other databases, if all or most new databases need a certain option set other than to the server default, you can save time by putting those settings into the model database. Examples include:

* The default recovery model for new databases;
* The default collation for new databases (if different from that on the server)

**Resource Governor**

You can also set up Resource Governor during your post-installation tasks. You can build and run Transact-SQL (T-SQL) scripts that will create the Resource Governor objects and enable it. Here are some tips for using Resource Governor:

* Resource Governor limits CPU and memory access by percentages to selected groups of logins. But it will not prevent a query from using all the CPUs on the server. Therefore, always configure Max Degree of Parallelism when using Resource Governor.
* You can make Resource Governor data-driven by modifying the classification function to read a table. Such configuration tables are usually created in the master database. For larger tables, using global variables can be used to optimize the classification function.
* You can prevent users from using the MAXDOP query hint to override the configuration setting for max degree of parallelism by setting the Degree of Parallelism option for Workload groups.
* You can use Resource Governor to reduce “runaway” queries by limiting the CPU time.

For more information about Resource Governor, see [Using the Resource Governor](http://msdn.microsoft.com/en-us/library/ee151608%28SQL.100%29.aspx).

**Multiple Tempdb Files**

Normally after installing SQL Server, you will want to move the tempdb database to the SAN or a different location locally and add multiple files. (For instructions on moving tempdb, see [Moving System Databases](http://msdn.microsoft.com/en-us/library/ms345408.aspx).)

Microsoft provides a general guideline that the number of tempdb data files should be based on the total number of logical processors on the server, up to a maximum of 8. Because this is a general guideline, it is possible that some configurations will not benefit from multiple tempdb files. If possible, you should test to make sure that your system’s behavior benefits from multiple tempdb files. In any case, every tempdb file should be configured to have the same initial size and growth rate so that the proportional fill rates will be identical across files. See [Optimizing tempdb Performance](http://msdn.microsoft.com/en-us/library/ms175527.aspx) and [Storage Top 10 Best Practices](http://technet.microsoft.com/en-us/library/cc966534.aspx) for further discussion.

**Sample Databases**

Microsoft supplies the AdventureWorks sample databases on CodePlex. Normally DBAs do not install the sample databases on production servers. Instead, you may want to add them to development servers because often SQL Server Books Online will give examples of queries and provide queries in tutorials that make use of those databases. You can get the SQL Server 2008 R2 AdventureWorks samples at [AdventureWorks 2008R2 RTM](http://msftdbprodsamples.codeplex.com/releases/view/45907). If you have users that query the data warehouse directly, it will be useful to have somewhere they can apply examples from Books Online.

After installing the AdventureWorks sample databases, you can change the owner of the databases to a non-user account such as sa (see “Database Ownership” in the next section for more information).

### Ensuring Best Practices for Configuration

To help ensure that you have a server that is ready for user databases and data, you can also download or enable the following tools, which help with SQL Server configuration and management.

**Feature Pack/Best Practices Analyzer**

The Microsoft SQL Server Feature Pack contains a number of helpful utilities and drivers, such as the Best Practices Analyzer and Upgrade Advisor. The Best Practices Analyzer provides you a way to inspect a server on the spot and determine whether it meets the general SQL Server best practices that Microsoft has accumulated over the years. You can download the feature pack at [Microsoft SQL Server 2008 R2 Feature Pack](http://www.microsoft.com/downloads/details.aspx?FamilyID=ceb4346f-657f-4d28-83f5-aae0c5c83d52&displaylang=en).

**Policy Management/UCP**

In addition, you can enable the Policy Management utility in SQL Server and enlist the newly built server as a member of a Utility Control Point (UCP) server configuration. You can also enlist it as part of a Management Data Warehouse. For more information about the UCP, see Chapter 5, “Querying, Monitoring, and Performance Tuning.”

**Central Management System**

The Central Management System is a Codeplex project offered as a free download by Microsoft to extend the multi-server management capabilities built into SQL Server 2008 R2. For details, see [CodePlex Project Functional Specification - Central Management System](http://sqlcms.codeplex.com/).

## Managing Databases

Enterprise data warehouse databases have characteristics that often differ considerably from OLTP databases. In this section, you will learn about many of the more prominent differences you are likely to see.

### Choosing a Database Recovery Model

Normally, data warehouse databases do not need to restore to any point in time, just to a particular day, as the following characteristics demonstrate:

* Data warehouse databases are typically loaded on a periodic basis, often daily or weekdays, and not continuously. Therefore, it is only necessary to restore a database to a certain day, not to a certain time. In such a scenario, weekly full backups followed by daily differential backups are enough to restore to a given day, so transaction log backups are not required.
* In addition, data warehouse loads work the transaction log of a database very hard, but it is unlikely that you would ever want to restore to a particular point in a load. Rather, it would probably be more convenient to restore to a particular day and then restart the most recent loads in order to bring the data warehouse up-to-date.
* Data warehouse loads often use staging databases to assist with transformations. These staging databases typically would never need to be restored to a particular point in time. Rather, the load process would simply start over.

The Simple recovery model enables bulk loading for the ETL load processes, while also managing transaction log truncation automatically. As a result, the Simple recovery model is often sufficient for data warehouse databases.

It’s worth pausing for a moment to compare the SQL Server recovery models’ support for other features, such as those for high availability. Table 4-1 provides a simple comparison.

|  |  |  |  |
| --- | --- | --- | --- |
| Supports | Simple | Bulk-logged | Full |
| Transaction log backups | No | Yes | Yes |
| Point-in-time restore | No | No  | Yes |
| Database mirroring | No | No | Yes |
| Bulk loads | Yes | Yes | No |
| Log shipping | No | Yes | Yes |
| Data compression | Yes | Yes | Yes |
| Table partitioning | Yes | Yes | Yes |
| SQL Server transactional replication | Yes | Yes | Yes |

**Table 4-1:** SQL Server recovery models for different product features

Note the following about the recovery models:

* In the Simple recovery model, transaction log backups are not possible.
	+ Large transactions can make the transaction log grow temporarily, so you must manage its size.
	+ Hence, the transaction log must be sized appropriately and allowed to grow.
	+ In the Simple recovery model, the transaction log will grow just as it will grow in the Full or Bulk-logged models, but in the Simple recovery model, the log will be truncated automatically without the need for a transaction log backup.
* SQL Server database mirroring requires the Full recovery model, and log shipping requires the Full or Bulk-logged recovery model. (The transaction log under the Full recovery model will generally grow at a somewhat faster rate than a similar log in the Bulk-logged recovery model.)
* Fortunately, data compression, table partitioning, and SQL Server transactional replication are possible in all three recovery models. Be aware though that publishing tables using transactional replication will cause fully logged transactions regardless of the recovery model.
	+ However, transactional replication does require that replicated tables have primary key constraints, which is not always a practice in data warehouses.
	+ Also, undelivered replicated transactions will prevent the log from being truncated in all three recovery models.

### Database Configurations

There are a few configurations that should be made at the database level before any user tables are created, and here are several of the most prominent.

We can get the following database configuration items out of the way quickly:

* Always set Auto Close and Auto Shrink off, especially on large databases.
* Run the database at the SQL Server 2008 compatibility level of 100.
* Set the Backup Compression default On. For large databases, the gain is so great compared with the cost; this option is an easy choice.

Let’s look at several other configuration items that may affect you as a data warehouse DBA.

**Collation**

Database collation can be different from the server collation. If it is necessary for you to make such a change, be sure to research the myriad of issues surrounding collations, beginning with SQLCAT’s recent white papers dealing with SQL Server 2008 collations:

* [The Impact of Changing Collations and of Changing Data Types from Non-Unicode to Unicode](http://blogs.msdn.com/b/sqlcat/archive/2007/12/06/new-best-practices-whitepaper-published-the-impact-of-changing-collations-and-of-changing-data-types-from-non-unicode-to-unicode.aspx)
* [Working with the New SQL Server 2008 Collations and Earlier Versions of the Native Data Provider](http://sqlcat.com/whitepapers/comments/1018.aspx)

However, SQL Servers generally is easier to manage if the database and server collations are the same.

**Database Ownership**

A data warehouse database server is likely to have many databases placed on it for a variety of purposes. You should address the issue of database ownership early and settle on a DBA team standard if a sensible one is not already in place. Here’s why:

* Every database has an owner. The owner is some SQL Server login initially. You can specify the database owner when you create a database, either in SSMS or in a T-SQL script. If you don’t specify an actual database owner, the account you use when you are creating the database will become the owner.
* Database ownership is different from either the dbo user or the db\_owner role inside a database. Database ownership is a property of the database; the login may create the database and never enter the database to look at any data, yet still be the owner.
* Database ownership, in fact, refers to the SQL Server login that owns the database, as far as the master database is concerned. Because that login is the owner, it inherits db\_owner rights in the database, but database ownership by itself is not a permissions property.
* It’s easy to overlook database ownership and let the current user become the owner. We’ve all seen SQL Servers where databases are owned by domain logins, and it looks rather strange.
* Worse, if those logins become invalid in the network, the ownership will not change.
* Also, mismatched database owners can inhibit database chaining, which will be discussed later in this section.

Therefore, it’s a good practice to change the database ownership of all databases after install using the system stored procedure sp\_changedbowner. You have several options here:

1. Make all databases owned by sa
2. Make all user databases owned by some domain account, which could be:
	* A user account
	* A special domain account that will never be removed

SQL Server already makes the system databases owned by sa, and that cannot be changed. It doesn’t matter whether your SQL Server instance uses Windows-only authentication, the sa login always exists and always is the database owner of master, tempdb, and so on. Thus, the sa login can be the owner of other user databases even if the sa account is disabled.

Therefore, it is a best practice to make the sa login the owner of all databases. This strategy matches with the SQL Server practice, and database ownership then does not depends on entries in Active Directory.

**Database Chaining**

For databases that are required to participate in database chaining, you must enable it at the database level. Database chaining is useful when you present logical views of data warehouse data to users that have their own work databases as their database context. For an introduction, see the SQL Server Books Online article,

* [Ownership Chains](http://msdn.microsoft.com/en-us/library/ms188676.aspx)

Also see “Production Databases” later in this section for more information on the use of database chaining in a data warehouse.

**Partition Functions and Schemes**

If you plan to use a standard set of partition functions and schemes across the database, create them before creating any databases. You can then refer to the partition scheme in the CREATE TABLE statements of your T-SQL scripts. Here are a couple of things to keep in mind about partition functions and schemes:

1. It’s easy to believe at first that each partitioned table should have its own distinct partition scheme, storing it on its own distinct filegroups, but that can make managing fact databases with multiple partitioned tables very difficult to manage.
2. Instead, if you can share the partitioning logic across many tables, you do not have to manage nearly as many files for a given database.

On the other hand, if every fact table must have its own partition scheme and function, it may not be possible to create those partition functions and schemes ahead of time. (Other table partitioning best practices are discussed later in this section.)

### Staging Databases

There are many types of staging areas that may be on a data warehouse server, depending on the type of transformations required and the strategy chosen to transform the data. You can categorize staging based on the type of loading:

* From other SQL Server data sources
* From external data sources

Data from external data sources may require more transformations because the data types may not match or the data may not be normalized. In addition, the incoming data might be free-form and need to be cleaned up before being allowed into the data warehouse. Data from SQL Server databases or other relational databases will tend to be more reliable, at least for consistent data types.

For staging data, the volume may vary depending on the load. But usually the following holds true:

* The Simple recovery model is sufficient for staging databases
* Staging tables can be smaller and more volatile than production data tables, so table partitioning is not required.
* Normally, you would not partition a staging table because the loading may be occurring based on the partitioning key of the main table anyway, so it would overload one partition.
* Keep in mind that data compression will also be useful for the larger staging tables.

### Production Databases

The production databases are where the data warehouse data—typically dimension and fact tables—is stored. At the enterprise level, you will most likely have many different data warehouse servers, each having multiple production databases. Let’s look at some simple enterprise data warehouse scenarios that illustrate that possibility.

**Multiple Data Warehouse Servers**

Sometimes an enterprise data warehouse will be housed in a single database server, usually because the organization or user base is small enough that one server can handle the entire user query load. When that is not possible, the data warehouse may be distributed across many servers. Figure 4-11 illustrates two very simple versions of this distribution.



**Figure 4-11:** Enterprise data warehouses often involve several data stores

The two approaches shown here are idealized, and things usually become much more complex in practice. Essentially, your data warehouse may involve the use of several production database servers. Usually, the simplest way to implement the data transfer between these multiple servers is to keep the database schemas identical: the same tables, partitions, views, and indexes.

Thus, an enterprise data warehouse DBA team will often have to deal with multiple database servers with databases that may be nearly identical in structure.

**Multiple Databases on the Data Warehouse Server**

Most likely, your enterprise data warehouse will also involve many production databases on each database server. The implementation of production databases depends principally on decisions made by the data warehouse architect. (This discussion assumes you are familiar with the “Physical Design” section in Chapter 2, “Data Architecture.”)

Here are two such architectural decisions the data warehouse architect has to make:

* Allow users direct access to the data warehouse database or place users separately in their own database.
* Organize data warehouse data into one database or into several databases.

Combining these two decisions leads to four possible options, as Figure 4-12 illustrates.



**Figure 4-12:** Four optional ways to implement production data warehouse databases

As we work through these options, you can see a natural evolution in the complexity and flexibility of the enterprise data warehouse:

* In the simplest model, Option 1, all users have direct access to the database. This turns out to be very cumbersome because it implies that users will be able to save temporary data in the data warehouse. Further, the users will be selecting from the physical names of tables, and you may not want that.
* A better approach is Option 2, where users have a work database they access, which gives them the ability to select from the logical tables of the data warehouse, not the physical tables. Now users can store their intermediate data in a separate work database.
* However, different groups of users have different needs, and at the enterprise level, it is usually more convenient to have multiple sets of work databases. Your implementation may have one database used in common by some users, whereas other departments or groups have their own database. Providing a group of users their own database gives the department more privacy and security.
* These groups also give you a quick way to classify the connection for assigning the users to a particular Workload Group.
* Finally, enterprise data warehouses, for purposes of manageability, often will split up the data warehouse into several databases, based on their function. For example, there might be a dimension database, along with one or more related fact databases, based on how well the data can be distributed. If the data warehouse has just two or three very large fact tables that are related to each other (in the same design domain), then one fact database might be warranted. But if there are several large fact tables in somewhat unrelated domains, multiple fact tables may provide more advantages.

As a DBA, you’ll be called upon to create, implement, change, protect, and administer these multiple databases. The following discussion assumes you could be working with the most complex but also most flexible architecture, Option 4—which is probably the most common enterprise data warehouse scenario.

### Partitioning Large Fact Tables

The large fact tables normally make up the bulk of an enterprise data warehouse. At an enterprise level, there is no way to properly query or manage very large tables unless you partition them.

Table partitioning has many management and performance benefits. For example, users can place filters on their queries to restrict the query scope to a subset of partitions. (See the “Partitioned Tables” section in Chapter 5 for more discussion of tuning data warehouse queries that access partitioned tables.)

Let’s assume you and your DBA team are working with multiple production databases on multiple servers. Here are some preliminary guidelines for preparing dimension and fact tables before they are populated by the ETL processes.

**Partition Functions and Schemes**

As noted earlier, you should pre-create, to the extent possible, the partition functions and schemes that will be required in the production database. If you know that all fact tables will be partitioned on a date or datetime key, and that the date range is always within a specified 5-year or 10-year range, you can simplify and have just one set of partition functions—and possibly even one set of partition schemes.

Remember that you must create the partition function first, then its partition schemes, and that you must reference the partition scheme name in the CREATE TABLE statement while passing the partition key as a parameter to the partition scheme. Figure 4-13 shows the relationship of partition function, scheme, and table.



**Figure 4-13:** The dependencies between partition function, scheme, and table or index

In addition, use data compression on large fact tables. SQL Server 2008 supports table-level compression, which provides great benefit for larger tables. You can even compress individual partitions. If possible, use page compression, which includes row-level compression, for the maximum benefit. You specify compression settings when creating the table (or the clustered index). Compression can also be useful when applied to larger dimension tables. You can measure the potential benefit of compression using a stored procedure: see [sp\_estimate\_data\_compression\_savings (Transact-SQL)](http://msdn.microsoft.com/en-us/library/cc280574.aspx) in SQL Server 2008 R2 Books Online.

**Partition Windows**

Make sure you know what type of partition window your partitioned tables require. Some common types of windows are:

1. **Sliding window.** This is a window that moves forward, usually forward in time. For example, your database might keep only the past 7 years’ worth of data and move the window forward every year.
2. **Rotating window.** This is a fixed window where the partitions get truncated and overwritten one by one, but the window doesn’t move. For example, if your table keeps the last month’s data only, you could partition the table into 31 partitions and feed data in through stored procedures based on the day of the month for a particular date in the datas. At the beginning of each day, the partition from last month with today’s day gets cleared out and overwritten.
3. **Fixed window.** In a fixed window, all required partitions are created ahead of time. No old partitions are removed, only new partitions are created as they are needed, and the data moves forward in time. Note that you are limited to 1000 partitions.

Although all three of these types are in production, perhaps the most common in an enterprise data warehouse is the third, a fixed window. This is often because the enterprise has no rules on when to archive old data, and implementing a sliding window is simply too complex.

**Filegroups and Files**

Be sure to map the appropriate filegroups and files to each partition. Normally, you can map each table partition to one filegroup and one file within the filegroup.

When deciding on the filegroups, attempt to balance the filegroups across the available SQL Server database drives. (In this discussion, we’ll use the term *drive* as synonymous with volume or LUN.) This requires careful planning, especially regarding space allocation.

As an example, consider the situation where you have a partition by month, over 5 years. Let’s say you have four available drives on the SAN for storing the partitioned database data. Figure 4-14 shows one possible strategy for how to balance the data.



**Figure 4-14:** A first approach to balancing data across available drives

Figure 4-15 shows the same configuration in a more tabular fashion.



**Figure 4-15:** A tabular equivalent to the first approach to balancing data

The big advantage of this approach is that loading data on any partition effectively spreads all the I/O across all the available drives, thereby maximizing the use of available spindles.

However, notice the number of files: 60 partitions with four files each is 240 files total, with 60 files on each drive. Consider the number of files if the example were more realistic, say with 10 years’ worth of monthly data spread across 10 available drives. That would mean 60 partitions with 10 files each, giving us a total of 600 files, resulting in 60 files per drive.

One way to reduce the number of files is to choose only one file per filegroup and assume that even though loads may tax one particular drive at a time, the user queries will probably be spread across the data in a somewhat balanced fashion. Let’s modify our example to contain just one file per filegroup. This lets us place one partition on just one drive, as shown in Figure 4-16.



**Figure 4-16:** A tabular view of one file per filegroup across 60 partitions

Now note how the file count has changed: each year has 12 files, so 5 years gives us a total of 60 files. Dividing by four drives puts 15 files on each drive.

The assumption of using one file per filegroup is that the load will still be mostly balanced. In this example, we assume that the query load across the combinations of months will be relatively balanced because users are interested in scanning large parts of the historical data. If users are ignoring history and just querying for the most recent month, for example, then we would need to return to spreading all the data across all the drives, as shown in the first version of our example.

If you use one file per filegroup, be careful not to overload the “before” partition. To see how this can happen, suppose that we create 61 partitions for our 5-year span, where the first partition captures all the data that has a NULL in the partitioning key or a date before the first year of our 5-year window. The NULLs might be temporary and data prior to year one might be required to provide certain matching values for other fact tables. In this case, we might put Partition 1 on drive E, and then map the rest so that the months line up with the drives, as shown in Figure 4-17.



**Figure 4-17:** Tabular view of one file per filegroup with a “before” partition

The problem here is that more space has been allocated on drive E than on the other drives, assuming all files begin with the same space. Further, if the “before” partition P1 grows a lot, you may run out of drive space earlier on drive E. If we do this many times, you end up using more of drive E than of any other drive. Also, this strategy isn’t efficient because if you run out of drive space on drive E, you cannot make use of any space on any other drive unless that drive is expanded.

One solution is to spread partition P1 across all the drives to balance it out, as shown in Figure 4-18.



**Figure 4-18:** One file per filegroup, balancing the “before” partition across all available data drives

In this case, the initial “before” partition has four files, not just one file. Thus, anything loaded into that partition will be balanced across all the drives, and all four drives are now used equally and more efficiently.

The advantage to this approach is that the number of filegroups is a bit smaller. Large numbers of filegroups make it more complex to back up and especially restore a database by filegroup. As result, for relational databases with partitioned tables, it is more common to take a weekly full backup followed by daily differential backups.

**Partition Keys and Indexes**

Here are some tips for partition keys and indexes:

1. Partition just the large fact tables. This requires that your fact tables have a suitable partition key column, normally a date, and that you include the partition key column. Large generally means 100 million rows or more.
2. Partitioning is most useful for fact tables where data is distributed based on a date range.
3. Choosing the partition key is critical: It’s required as part of the clustered index, and to partition non-clustered indexes, the partition key must be either part of the non-clustered key or included in the index. SQL Server 2008 will do this automatically for non-unique non-clustered indexes.
4. Unique indexes and primary key indexes require [special guidelines for partitioning](http://msdn.microsoft.com/en-us/library/ms187526.aspx). They should be created on the default filegroup. You can remove or disable them when using the ALTER TABLE statement with SWITCH or when using the ALTER PARTITION FUNCTION statement with SPLIT or MERGE.

For a list of best practices when partitioning data warehouse fact tables, start with [Top 10 Best Practices for Building a Large Scale Relational Data Warehouse](http://sqlcat.com/top10lists/archive/2008/02/06/top-10-best-practices-for-building-a-large-scale-relational-data-warehouse.aspx). And for a thorough treatment of table partitioning in SQL Server 2008, see the Microsoft white paper [Partitioned Table and Index Strategies Using SQL Server 2008](http://technet.microsoft.com/en-us/library/dd578580%28SQL.100%29.aspx).

### Work Databases

You can use one or more work databases to provide a database context for users when they log into the data warehouse. You can then provide the users with a logical rather than physical view of the data warehouse fact and dimension tables.

Here are some notes to keep in mind about providing work databases:

1. An enterprise data warehouse will usually have several work databases, named for the types of users who have access to them. Often, you may have one database for each department, division, or project.
2. There may also be one common work database for users who don’t fit into a particular department or project, such as more casual users or reporting functions.
3. A work database gives users a place to store intermediate data, create their own objects such as tables and views, and store their own T-SQL code using functions and stored procedures.

There are two major approaches to giving users a logical view of the data warehouse data in their own work database: synonyms and views.

**Synonyms**

SQL Server synonyms are database objects that serve to replace a certain namespace such as a database.schema reference or a database.schema.tablename reference. A synonym called SalesHistory, for example, could replace a fact table name such as FactDB1.Sales.FactSalesHistory, and the user simply creates a query using the synonym rather than referencing the external database.

For example, instead of issuing the query

SELECT TOP 10 \* FROM FactDB1.Sales.FactSalesHistory

the user issues the much simpler query

SELECT TOP 10 \* FROM SalesHistory

Similarly, a synonym might stand in place of a schema. With a synonym of Sales standing in for FactDB1.Sales, the user simply issues the query

SELECT TOP 10 \* FROM Sales.FactSalesHistory

The primary advantages to synonyms are:

1. They are simple to create and maintain.
2. Because they just replace object names, you don’t need to worry about changing the synonym if the underlying table structure changes.
3. Synonyms do not require database chaining.

Disadvantages to synonyms include:

1. They are not objects for data source views in SSAS, so users cannot use them to build cubes. To build a cube in SSAS, the user would need access to the underlying tables.
2. Synonyms cannot replace complex joins; they only replace object names. As a workaround, you could create synonyms that access views created in the underlying fact and dimension databases.
3. Synonyms do not show the object metadata, so the user must rely on published documentation for column names and data types when synonyms stand in for tables.

**Views**

SQL Server views can also provide a logical view of the data warehouse to users. You can create the views in the work database. For example, you might define the following view:

CREATE VIEW Sales.FactSalesHistory

AS

SELECT TOP 10 \*

FROM FactDB1.Sales.FactSalesHistory

This has the same effect as the synonym, with some additional advantages:

1. Views can be source objects for data source views in SSAS, so users can build their cubes based on the views in the work database.
2. Views can show metadata, so the user can discover or verify column names and data types by inspecting the view using SSMS’s Object Explorer.
3. Views can embed complex joins to make querying easier for users.
4. Views can be encrypted if you want to hide physical information about the underlying tables.
5. Views are very flexible: You can use views to restrict data, expose only a subset of the columns to the user, or change the column names and data types that the user sees.

The disadvantage of using views is that you must revise or recreate the view if the underlying table changes.

Because the advantages of views outweigh those of synonyms, views are commonly found in enterprise data warehouses.

**Security and Database Chaining**

For either synonyms or views, you must grant users some kind of access to the underlying fact and dimension databases.

For views, you must enable database chaining, which involves the following steps:

1. Make sure all databases are owned by the same login. Normally, this should be the sa login (see “Database Ownership” earlier in this section for more information).
2. Enable database chaining on each database involved; this includes the work database and all underlying databases that will be referenced by the views.
3. Create the views logged in as the SQL Server system administrator.
4. Grant CONNECT and perhaps SHOWPLAN to the users in each underlying database. You do not have to grant the user direct access to any underlying tables.

Enabling database chaining can involve some risk, which you can avoid if you are careful. For details, see the “Potential Threats” section in the SQL Server 2008 R2 BOL article [Ownership Chains](http://technet.microsoft.com/en-us/library/ms188676.aspx).

If you place suitable restrictions on your users’ security, these risks can be mitigated, as we discuss in the next section on database server security.

## Database Server Security

Relational data warehouse users use data differently than OLTP users. They primarily query the data and never update data warehouse data, although they might update their copies of that data. The relational data warehouse DBA’s approach to SQL Server security needs to reflect that difference.

Starting with SQL Server 2005, all permissions in SQL Server are grantable. Essentially, that means the degree of granularity of permissions extends to every object in the database. This can result in considerable complexity, and it can be very laborious to maintain complex sets of permissions.

Therefore in a data warehouse, it is wise to adopt security strategies that can help simplify and standardize your approach to security and keep it manageable and under control.

### Security Strategies

Among available security strategies, a good place to start is with user authentication via logins. There are three steps you can take to help simplify security management in a relational data warehouse: use Windows authentication, use Windows groups as logins, and assign permissions to groups of objects, not objects directly. Let’s look at each of these strategies in turn.

**Use Windows Authentication**

SQL Server supports two methods of authentication: Windows logins and SQL Server-originated logins. Here are the main considerations for which method to use:

* **Windows authentication is more flexible.** It can be based on either an individual Windows Active Directory user account or a Windows Active Directory group. Microsoft generally recommends Windows authentication for SQL Server because SQL Server does not need to store a hash of the user’s password: It can query the Windows network directly for authentication of the user. In addition, SQL Server can store the Windows SID for the login and not have to create its own artificial SID. When a database is restored to another server, the user associated with that login will not be orphaned. (It is still possible to have orphaned SIDs if the Windows account is missing, but presumably that is a rare occasion.)
* **SQL Server authentication is considered less reliable.** SQL Server has to store the hash of a SQL Server-originated password internally, and the encryption is not as strong or as protected as Active Directory passwords are. SQL Server has to create a SID for the login at the server level, so when a database with users associated with that login is restored to another server; the SIDs will be initially orphaned and will need to be manually synchronized.

For a relational data warehouse, where users are usually not applications but rely on their Active Directory credentials for logins, Windows authentication is much better. Many enterprise data warehouses rely on Windows authentication exclusively and do not allow SQL Server logins.

**Use Windows Groups as Logins**

Another method of simplification is to use just Windows groups as logins. This involves adding individual users to Windows groups and then granting login rights to the groups and not the individual users. Using Windows groups can dramatically simplify permissions management because you can create users in the required databases that match the Windows group and assign permissions to the group’s database username, and members of the group will simply inherit those permissions.

Using Windows groups can be made much easier if you have either a domain or an organizational unit (OU) of the enterprise domain assigned to your systems people. Then you can manage adding and removing the users from your data warehouse groups without having to rely on overworked network administrators. The users you add to those groups can be in different domains within the enterprise, provided that the domains are fully trusted.

However, don’t allow groups to belong to other groups: This will only make the management more complex. Keep it simple and at one level, if at all possible.

You also need some way to retrieve information on a periodic basis about what users are currently assigned to the Windows groups. You may be able to get this from your network administrators, or you can use a PowerShell script to query Active Directory on the current domain.

**Assign Permissions to Groups of Objects, not Objects Directly**

When a relational data warehouse contains a large number of tables and other objects, assigning permissions to individual objects can be very tedious. It can also be error-prone because many of the objects “hang together” in natural groupings or domains based on the type of data they contain. The solution is to assign permissions to groups of objects. Here are a couple of approaches:

* **Assign permissions only to views and not to tables.** This can help hide joins among fact tables and even joins to dimension tables. This strategy can help simplify by reducing the number of table-type objects you must assign permissions to, but you still have stored procedures and functions that require individual assignments.
* **Assign permissions to schemas and not objects.** In this approach, you collect related fact tables, perhaps dimension tables, and views, stored procedures, and functions together into a single schema that describes the data domain. Often, this kind of organization can be coordinated with the data model, which may already have such groupings.

Assigning permissions for database users based on Windows groups and to natural groupings of database objects provides a useful method of keeping the number of permissions you must assign down to a manageable level.

### Querying Other Servers

A relational data warehouse is frequently part of a system of database servers. It often may collect data from one set of servers and distribute data to another set. When you query one database server, you may need to combine results from another server, either for actual data or quite often for metadata.

Such queries require linked servers to permit the use of four-part names to reference other SQL Server instances. You can find more about linked servers in the SQL Server 2008 BOL article [Linking Servers](http://msdn.microsoft.com/en-us/library/ms188279.aspx).

When creating a linked server, you should allow only Windows users and not SQL Server logins, if at all possible.

**Apply the Principle of Least Privilege to Linked Servers**

When defining the security for a linked server, explicitly list the users that can access the linked server, and do not allow other users to make connections implicitly. Figure 4-19 shows a screen shot of the New Linked Server dialog box in SSMS, where you set security for a linked server.



**Figure 4-19:** Limiting security on a linked server definition

To not allow implicit connections, simply select the option that if a user is not in the local server logins list, connections will not be made. The explicit list of local server logins requires specific Windows accounts only; Windows groups are not allowed, so you will have to explicitly define the users.

When defining the linked server, it is a good idea to set the RPC and RPC Out options to True (they default to False.) Figure 4-20 shows the options on the New Linked Server dialog box.



**Figure 4-20:** Setting the RPC and RPC Out options to True

Setting the RPC options to True makes it possible to execute stored procedures on the target server and receive data from the results of the stored procedure. This comes in handy when you need to “push” data from server A to server B via a linked server. You can call a stored procedure on server B that pulls the data from server A to a temporary table and then populates the local table accordingly.

When working with SQL Server Agent jobs that require the use of a linked server, and the SQL Server accounts are not the same between servers, you can set up a credential using a Windows account meant just for SQL Agent execution, list the account in the Security list of the linked server, and then assign the credential account to a SQL Agent proxy.

For more information about using credential accounts and SQL Agent proxies, see the following SQL Server BOL articles:

* [CREATE CREDENTIAL (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189522.aspx)
* [Creating SQL Server Agent Proxies](http://msdn.microsoft.com/en-us/library/ms189064.aspx)

### Maintaining and Monitoring Security

The primary tool for managing security in SQL Server is SSMS, which often provides the best way to visually inspect whether certain users have a specified set of permissions.

**Using Stored Procedures to Assign Permissions**

For assigning permissions, however, it is better to use stored procedures, which are T-SQL scripts that encapsulate often-repeated actions and perform all the required actions. For example, you may have a new group of finance users that needs access to a given database. The stored procedure might perform the following commands:

* CREATE LOGIN for the new group
* CREATE USER in the target database for that login
* GRANT CONNECT to the user in that database
* GRANT SHOWPLAN to the user in that database
* GRANT SELECT to the Dimension schema in the database
* GRANT SELECT to the Finance schema in the database.

**Monitoring Tools**

You can also use SSMS to monitor some aspects of security in your relational data warehouse. But for getting summary data across many users or many databases, you will have to select from the system tables. The kinds of tables you need to query are:

* In the master database:
	+ *sys.server\_principals* contains a list of all the logins
	+ *sys.server\_permissions* lists the server-level permissions assigned to the login
* In each database:
	+ *Sys.database\_principals* contains a list of all the users
	+ *Sys.database\_permissions* contains the permission assignments to the database users
	+ *Sys.database\_role\_members* links the database principals that are users with the database principals that are database roles
* You can also use the built-in functions that show permissions for the current user:
	+ fn\_my\_permissions() shows permissions for a given securable
	+ HAS\_PERMS\_BY\_NAME() shows permissions by securable
* SQL Server 2008 provides enhanced auditing capability with SQL Server Audit.

What SSMS and T-SQL queries will not do is tell you effective permissions—that is, permissions that a user has by being in a particular role. However, some third-party vendors have SQL Server security tools that will detect and show effective permissions.

**Note:** Occasionally, you will need to query for orphaned security objects and remove them. Usually this involves users in databases when a login is no longer valid but the user remains in the database.

## Managing Database Change

Implementing changes to the storage structures of large tables is always a challenge, and enterprise data warehouses generally have very large tables. But changes to the data warehouse data model do occur, and occasionally fact tables must have their schemas modified.

Some changes to table structures are benign, such as renaming a column, enlarging a column’s varchar data type, or changing a column’s smallint data type to an int. Adding a nullable column to the end of a table’s schema can occur without taking the table offline. Changes to non-clustered indexes may also be non-intrusive, although user query performance might be affected while the index is being rebuilt.

However, many changes to table structures do have a significant impact. Perhaps the most significant is adding a new column or set of columns to a large table and significantly changing the position of the other columns. Effectively, the table must be reloaded because the new columns must be populated throughout the table.

In addition, changing objects such as a table or view, or the output of a stored procedure, may break applications that rely on those objects. Some impacts may require some time to detect, because they affect little-used areas of the application or require conditions that take time to reproduce. For a fuller discussion, see the SQL Server CAT Team blogs:

* [Managing Schema Changes (Part 1)](http://blogs.msdn.com/b/sqlcat/archive/2006/03/01/541550.aspx)
* [Managing Schema Changes (Part 2)](http://blogs.msdn.com/b/sqlcat/archive/2006/03/31/566046.aspx)

### Change Management Tools

Different types of change management tools can help data warehouse DBAs effectively manage such change.

**Source Code Control**

The first step in managing change is to ensure you are using a good source code control system for storing DBA scripts. For source code control, Microsoft provides the Source Control Explorer in Visual Studio Team Foundation Server (TFS). You can also use other third-party tools for storing T-SQL code, although TFS provides full support for SQL Server database scripts.

* For information about TFS, see the MSDN article [Team Foundation Server 2010](http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx).
* For details about Source Control Explorer and related features in TFS, see the MSDN article [Using Version Control](http://msdn.microsoft.com/library/ms181368.aspx).

**Database Change Management**

TFS also has an add-in called the Database Edition (known previously as Database Professional or DB Pro) specifically designed for managing database change. The Database Edition add-in contains tools for database comparison, change management, and deployment of database changes. For more information, see the MSDN readme file at [General Distribution Release (GDR) for Visual Studio Team System 2008 Database Edition](http://msdn.microsoft.com/en-us/library/dd819149%28VS.90%29.aspx).

### Version Deployment

Deploying the initial versions of objects to a database is straightforward. It is deploying the subsequent changes to systems that are already in production that is more challenging.

**Deploying Initial Versions**

Generally you can deploy initial versions of database objects using scripts that simply create the objects. The scripts may differ considerably between staging and production systems, due to the many transformations that ETL processes may make. Hence, in your source control system, you should typically keep staging and production script separate from each other.

**Deploying Subsequent Versions**

In the case of smaller dimension and fact tables, deployment may require outages that are within a tolerable amount of time, say a few hours at the most. So a planned downtime during a low-usage period such as a weekend is possible.

However, schema changes to the larger enterprise data warehouse tables could easily take days to complete, and taking the original table offline is simply not feasible. In such cases, it may be possible to load the new version of the table in place of the existing version. This process can be made much easier and transparent to end users by using a view to access the underlying table. For example, Figure 4-21 shows the current table as seen by users through a view, while a parallel table is being loaded.



**Figure 4-21:** A parallel load of a new version of a large table

Note that the user selects from a table called Finance.Sales, and the view points to an underlying table called Finance.Sales1, indicating this is version 1 of the table. The table is being updated nightly. The new table, Finance.Sales2, is created in the same database and is being loaded while users continue to access Finance.Sales. The user cannot see the new table because the user has permission only to the view.

Once the new table is loaded and current, the user’s view is recreated, and within seconds users are now pointed to the new table, as shown in Figure 4-22.



**Figure 4-22:** The new table is loaded and now being updated nightly

Users now see the new columns or columns with changed data types in their view and without any disruption. The new table, Finance.Sales2, is now being updated nightly with a new set of ETL packages.

You should continue to update both tables nightly until the new table is considered stable, so that you can roll back to the old structure if necessary. But after users are comfortable with the new version, you can safely drop the old table.

This technique assumes that your database server has enough free space to store two versions of some of the larger fact tables at the same time. For the largest tables, this may not be an easy task. Nevertheless, the benefits of minimal downtime usually far outweigh the cost of extra storage. Because it is likely you would only use this technique for one large table at a time, and very rarely, you might be able to shrink the database after dropping the old table to free up space for the next large table or for other purposes.

## Business Continuity

In the context of a relational data warehouse, ensuring the continuity of your business involves more than just adopting technologies that provide high availability and disaster recovery. As a relational data warehouse DBA, your responsibility is to help ensure reliable and prompt recovery from any event that could impede the ability of your data warehouse to serve its customers.

Database servers are generally in service to applications as customers. However, relational data warehouses can be unique in that often the customers will directly query the server for their own purposes. As a result, you need to plan for and provide for business continuity for a wider range of customers than OTLP database systems usually have, including users who directly query in T-SQL as well as users who build cubes in SSAS based on relational tables.

### Planned Outages

The data warehouse may become unavailable or out of service to customers for a variety of reasons, but it is important to expressly plan for patches and updates to the data warehouse that you know will be required. It can help to categorize planned outages into those that are system-oriented and those that are related to the database.

Some examples of the types of system events you can plan for include:

* SAN upgrades and patches
* Security patches for the database server’s Windows Server OS
* Windows Server service packs
* SQL Server service packs and cumulative updates

Generally, DBAs do not like to apply cumulative updates (CUs) unless they have a direct benefit for the system in question. CUs are not as thoroughly tested as service packs (SPs), and most fixes in a CU will be rolled up into a later SP anyway.

Note that it is a good idea to test the service pack on a non-production system before applying it As soon as you have vetted the service pack, you should apply it. If you delay applying a service pack, you are exposing your system to problems caused by known bugs, and those problems could have been easily prevented by applying the service pack in a timely manner.

You can also plan for changes to the data warehouse that require a server outage but are not system-based. For a relational data warehouse, these may include:

* Loading fact tables via ETL processes
* Changing a table’s column structure
* Rebuilding a clustered index

Typically, data warehouse DBAs do not want users to access data during data loads. The ETL processes have a certain window in which to accomplish their tasks, so it is important to make sure that user queries do not interfere with them. For more information about ETL processes, see Chapter 3, “Integration Services.”

Sometimes changes must be made to table schemas, and that often requires some kind of outage. When the changes are minor, no visible outage may be required. But if the table must be rewritten, an outage may be needed. See the previous section on “Managing Database Change” for information about pushing database changes to production and minimizing their impact.

Maintenance tasks can present another potential for an outage, especially when a clustered index must be rebuilt. When tables in a data warehouses are load-only, with no updates, the table may not require a clustered index and the point is moot. However, when tables have rows that may move because the clustered key (such as a date) has changed, the index may in time require rebuilding. See the “Maintenance” section later in this chapter for a deeper discussion of maintenance issues.

### Unplanned Loss of Service

In addition to minimizing the impact of planned outages on your system, it is equally important that you prepare for unplanned outages—service losses that occur at unexpected times due to various types of failures. Such failures can occur at a variety of levels. Here is a sample list that starts at a high level and proceeds to the more specific data warehouse database level:

* Data center outage
* Power outage
* Internet outage
* Network outage
* Database server outage
* Storage system outage
* SQL Server instance unavailable
* Authorization failures
* Database unavailable
* Data corruption
* Accidental data deletion

You may find other items to list here or want to refine the list, but just make sure that you plan for addressing outages due to any of these types of events and document them in your Business Continuity plan.

When planning for unexpected outages, do not work in a vacuum. Instead, make sure you involve your management in knowing what to expect from any such event. In addition, make sure you are in conformance with, and have some say in, the enterprises’ Service Level Agreement (SLA).

### SLAs, RPO, and RTO

Ensuring business continuity means you need to respond to any interruption to your service within a specified period of time, usually defined in the SLA. SLAs usually contain two critical measures:

* **The Recovery Point Objective (RPO).** This is the point in time prior to an interruption of service within which you must recover data – in other words, it’s the amount of allowable data loss quantified by a time period.
* **The Recovery Time Objective (RTO).** This is the amount of time required to restore service to customers and applications after an interruption.

It is a good idea to explicitly define RPO and RTO times with your management so that you can test different scenarios and practice your response steps. A couple of notes to keep in mind about these measures:

* The RPO is the maximum amount of time prior to an interruption event up to which you can restore or recover data. In the context of a data warehouse that gets loaded nightly, you might stipulate that you will never lose more than one day’s worth of data after recovery from an interruption.
* The RTO measures the amount of time you expect to take to reach your RPO. Obviously, this will change dramatically based on the nature of the interruption or failure.

Be sure you include planned as well as unplanned outages when defining RPO and RTO times. Everyone involved—management, your colleagues in other departments, and your DBA team members—need to realize that sometimes planned outages are required.

### Reliability, Redundancy, and Recovery

When thinking about business continuity, it’s important to keep in mind the following best practices:

* **There is no substitute for reliable hardware and software.** You don’t want a data warehouse database server’s hardware or software to fail, so it’s important to invest in reliable and current products. Make sure that your database server is from a reputable vendor, that the Windows Server and SQL Server editions are correct, and that current service packs are applied. Just as important, make sure that the underlying SAN storage is reliable and adequately patched. All the hardware, including the server and the SAN, should be tested and baselined to make sure it performs properly at the initial stage.
* **Systems are kept available and recoverable by redundancy.** The list of redundancies is very long: SAN storage makes significant use of redundant disks, paths to disks, service processors, and switches. Database servers use multiple HBAs, multiple network cards, local disks in a RAID configuration, error-correction memory, and so on. When choosing among high availability and disaster recovery options, the technology choices usually involve deciding how you want to create redundancy.
* **Plan for recovery first, then backup.** When you define RPO and RTO times, you are defining what it will take for you to recover from an interruption in service. Therefore, when planning, it makes most sense to first work through your strategies for restoring from various types of failures and then select the appropriate mix of technologies (backups, clustering, mirroring, etc.) that allows you to meet your RPO and RTO goals.

With these points in mind, let’s take a look at high availability and disaster recovery options.

### High Availability and Disaster Recovery

The terms *high availability* and *disaster recovery* both refer to technologies that enable you to recover from an interruption in service. The terms overlap quite a bit in meaning, but there are some nuances and connotations that distinguish them.

* **High availability** refers to the set of technologies intended to enhance the availability of a system by preventing an interruption of service or at least allowing you to recover quickly from an interruption on the existing systems. The interruption could be due to anything, major or minor, but usually it is considered some kind of hardware or software fault on a server. Often, a second redundant server takes over to make the system available again.
* **Disaster recovery** refers to the procedures in place for recovering from a major interruption in service caused by some disaster. The most common scenario is the loss of a data center, and a common solution is to establish a data center located a safe distance away that can substitute for the main data center.

An effective disaster recovery plan normally assumes that underlying high availability technologies are already in place for interruptions in service that are not considered disasters. Then, the disaster recovery plan makes use of these technologies to establish and maintain a redundant data center location should a real disaster occur.

When a hardware or software fault on a database server causes a secondary server to come into place, the failover may be automatic or manual. That is, the other server may automatically take the role of the first server, or you may have to intervene in order to fully accomplish the transition.

From the standpoint of a relational data warehouse, it is useful to look at high availability technologies from the standpoint of whether they depend on the database recovery model. Because most relational data warehouses keep their databases in the Simple recovery model, let’s look first at the technologies that do not depend on the recovery model.

### Recovery Model–Independent Approaches to High Availability and Disaster Recovery

The reason most data warehouses keep their databases in the Simple recovery model is that the databases are normally updated by ETL loads on a daily or some other periodic basis. Thus, there is no need for the RPO time to be any less than the load period. Users only query the data and do not update it. The result is that you only need to restore to the last successful load of data.

**Failover Clustering**

Windows failover clustering is an excellent high availability solution for the data warehouse server. It does not require that the databases be in any particular recovery model, and all SQL Server databases, including the system databases, are protected. For a thorough white paper discussing SQL Server 2008 and Windows failover clustering, see [SQL Server 2008 Failover Clustering](http://sqlcat.com/whitepapers/archive/2009/07/08/sql-server-2008-failover-clustering.aspx).

Assuming you already know the basics of Windows failover clustering and SQL Server 2008, here are some things to keep in mind:

* Windows failover clustering uses shared data and redundant servers. A failover is from one server to another, using the same shared data but at different times.
	+ A key advantage of failover clustering is that the SQL Server instance is preserved with the original master and msdb databases.
	+ That means that the server keeps the same server name, the same (virtual) IP address, the same SQL logins, and the same SQL Agent jobs—no matter what server the instance is active on.
* The main use for failover clustering is as a high availability solution within a data center. It is not appropriate as a disaster recovery solution, where the failover cluster would span two data centers.
* Failover clustering servers share the same storage, which means that the storage location does not change during a failover.
	+ Storage is therefore a single point of failure in a Windows failover cluster.
	+ As a result, be especially cautious when SAN upgrades or patches are installed. If the cluster’s shared storage is not available, the cluster will not run.

If possible, install SQL Server 2008 clusters on Windows Server 2008 R2. Windows Server 2008 provides a simplified approach to clustering, and the SQL Server 2008 installer can take advantage of it. For more information, see [Six Failover Clustering Benefits Realized from Migrating to SQL Server 2008](http://sqlcat.com/top10lists/archive/2008/11/20/six-failover-clustering-benefits-realized-from-migrating-to-sql-server-2008.aspx).

Typically, Windows discussions about failover clustering for SQL Server consider the failover cluster with two nodes. However if you only have two nodes, when installing patches and upgrades to the server, there is only one other node available while the other node is being patched and availability is decreased in the event of a failure. The following case study by the SQLCAT team shows how one company, ServiceU, uses a three-node failover cluster to protect the SQL Server instance during patches: [High Availability and Disaster Recovery at ServiceU: A SQL Server 2008 Technical Case Study](http://msdn.microsoft.com/en-us/library/ee355221%28SQL.100%29.aspx).

Although this case study is not of a data warehouse, it does show how using a three-node failover cluster maintains the ability to automatically fail over to another node in case of failure of one of the functioning servers during the patching process.

**Secondary ETL Loads**

Another way to provide high availability for enterprise-level relational data warehouses is to run ETL loads to a second server. This option also accommodates databases that are in the Simple recovery model because it relies on ETL processes to populate and maintain a second data warehouse server. Figure 4-23 shows a simple diagram of how a secondary server can work.



**Figure 4-23:** Using ETL processes to load a secondary server for redundancy and scale out

The main requirement in this scenario is that the data warehouse data in both database servers remain in sync with each other. The key mechanism involved is the ETL process, which must therefore have some way to recover and repopulate the data if an ETL process should fail. In addition, you need to have some ways to query both systems to see if the data remains synchronized. You may be able to rely on existing data profiling processes to ensure that both servers remain in sync.

With this approach, both the servers and the storage can be redundant if the secondary server has a separate storage system or SAN from the main server. In addition, the secondary server can be queried and used to scale out the data warehouse system to support more users. If you use this strategy to scale out users, then consider having a third server that can serve as a backup in case any of the other servers should fail.

Note that with this scenario, however, failover is not automatic. Users will have to change their references to the alternate server in the event the main server fails, and you will have to ensure that the logins and SQL Agent jobs, as well as other types of server configurations, are the same on both servers.

**SAN Mirroring**

Another approach to storage redundancy is SAN mirroring, where data on one SAN can be mirrored to another SAN, either locally or in a different data center. Using SAN mirroring with SQL Server is possible, provided that the SAN vendor and your SAN engineers understand the requirement for SQL Server to have transaction log writes secure before data writes. This requirement, called Write-Ahead Logging (WAL), is described in the KB article [SQL Server 7.0, SQL Server 2000, and SQL Server 2005 logging and data storage algorithms extend data reliability](http://support.microsoft.com/kb/230785#appliesto).

Often, SAN vendors can provide something called consistency groups or the equivalent to guarantee write behavior that’s compatible with SQL Server. If the SAN engineers do not understand SQL Server’s WAL protocol, or the SAN vendor is not willing to guarantee it, be very cautious about using SAN mirroring.

### Recovery-Model-Dependent Approaches to High Availability and Disaster Recovery

To use some of the other high availability technologies, it is necessary to keep the databases in the Full or Bulk-logged recovery models. This requires you to take transaction log backups in order to manage the transaction log. Here are some high availability and disaster recovery technologies that fall into this category.

**Database Mirroring**

SQL Server database mirroring is a popular technology that can be used in both high availability and disaster recovery scenarios. Database mirroring has the following characteristics:

* Both the database servers and their storage are redundant because database storage is not shared between servers.
* Provided you have the bandwidth in place, database mirroring can take place over much longer distances between servers than failover clustering, and so it is suitable for disaster recovery.
* Failover from a failed database to a mirrored database can be automatic.
	+ The secondary mirror database can recover automatically if a witness instance is involved.
	+ Client applications can also fail over automatically if they are using the appropriate SQL Server client connectivity components. Note, however, that the individual pool of a connection must be cleared for the redirect to work properly. See the MSDN post, [Client-side Redirect problem](http://social.msdn.microsoft.com/Forums/en-US/sqldatabasemirroring/thread/5783bbe7-bfc8-4342-94a7-e83b07111cbb).
	+ Automatic failover requires high-safety mode, i.e. synchronous mirroring, which is only available in the SQL Server Enterprise Edition.
* Failover is at the database, not server level, so one database can fail over independently of other databases.
* Database mirroring requires the Full recovery model.
* You must manually ensure that the proper logins and SQL Agent jobs are created and kept current on the secondary mirror server.

Because database mirroring requires that you initialize the secondary server from a database backup, initializing mirroring using the VLDB databases of a data warehouse can be a challenge and must be planned out carefully. This issue will also arise if you need to re-initialize the mirror from a full backup.

**Log Shipping**

Log shipping is similar to database mirroring, except that it transfers log data to a secondary server through files and not through direct streaming. Unlike database mirroring, log shipping can work with both the Full and Bulk-logged recovery models.

Log shipping has a couple advantages over database mirroring:

* You can delay applying the logs on a secondary log-shipped server for a period of time, perhaps a few hours, so that in the case of an accidental deletion of data, you can query the log-shipped secondary database to get the deleted data, if it is in standby mode. This is different from using a database snapshot on a mirror database, which presents a consistent view of the database at a particular point in time that stays constant until the snapshot is dropped.
* You can log ship to multiple servers, whereas SQL Server 2008 only allows database mirroring from one server to a single mirrored server.

Some DBAs like to combine database mirroring and log shipping to have the automatic failover of database mirroring as well as the ability to recover deleted data that log shipping provides. For more information about combining log shipping and database mirroring, see the BOL article [Database Mirroring and Log Shipping](http://msdn.microsoft.com/en-us/library/ms187016.aspx) as well the SQLCAT white paper [Database Mirroring and Log Shipping Working Together](http://sqlcat.com/whitepapers/archive/2008/01/21/database-mirroring-and-log-shipping-working-together.aspx).

### Backup Planning

No matter what technology you choose for high availability or disaster recovery, you still need to take backups of your data. Database backups are the first and last line of defense against any and all types of disruptions and disasters. Because relational data warehouse databases can become quite large, creating a workable backup plan can be challenging.

If the databases in question are in the Simple recovery model, you cannot take transaction log backups. The only remaining options are full and differential backups. If the databases are in Full or Bulk-logged recovery models, you must take transaction log backups in addition to periodic full database backups.

The key lesson of this section is that you should choose your backup schedule to fit your RPO requirements. If you must provide the ability to recover to within one day (a common scenario for a data warehouse), then you need to schedule your backups accordingly. For example, you might choose to take full databases backups of the relational data warehouse data weekly—say, Sunday morning—and then take only differential backups nightly during the work week.

In addition, consider taking the following steps:

* Enable page checksums on the server. You do not want to back up a corruption.
	+ Also add WITH CHECKSUM to the backups so that the page check occurs there as well.
* Use backup compression. The cost is low and the benefits enormous.
* If your database does not have too many filegroups, consider taking full backups over a period of time using filegroup backups.
	+ Filegroups marked as read-only do not need to be backed up more than once.
	+ Databases with partitioned tables usually have large numbers of filegroups, and filegroup backups may not be practical.
	+ You can restore individual filegroups from a full backup as well as from filegroup backups.

Make sure that your backups are offloaded to secondary media such as a tape system or disk-to-disk-to-tape systems on a regular basis and that the tapes are kept with a reasonable rotation schedule.

**Note:** If you are taking transaction log backups and you have a database failure, always try to back up the tail of the transaction log. For more information, see the BOL article [Tail-Log Backups](http://msdn.microsoft.com/en-us/library/ms179314.aspx).

Database backups should always be tested by restoring them to a SQL Server instance. *There’s nothing worse during a crisis than to discover that your backup set will not restore.* One useful approach is to designate a secondary server just to test full and differential or transaction-log backups on a periodic basis. This testing can be justified as part of your business continuity plan.

## Maintenance

As a relational data warehouse DBA, you must constantly seek a balance between user access, data loads, and maintenance. Data loads are critical for keeping the data current and usable, so the maintenance window often has to make do with whatever time is left over in the day. Figure 4-24 shows how a typical workday might be divided up.



**Figure 4-24:** How a typical workday might be segmented for user access, data load, and maintenance cycles

Assume that users have access from 7am to 6pm, and that ETL loads must start after midnight to make sure their data is current as of the previous day. That leaves 6pm until midnight—or just 6 hours—for the maintenance cycle.

Unfortunately for many enterprises, the data warehouse may have users across several time zones, or possibly around the entire globe, which would mean that ETL loads and maintenance would have to cut into user access time. In those cases, decisions must be made about running all three operations in parallel. There may be ways to have maintenance operate just prior to loads for a given region that is in only one or two time zones, for example, so that those tasks can actually rotate around the clock.

In the maintenance cycle, there are three primary kinds of activities: backup, index maintenance, and integrity checks.

### Backup

Database backups are the primary activity that must complete within the maintenance window. As mentioned in the previous section on backup planning, database backups are your last line of defense. It’s quite common in data warehouse scenarios to take full database backups on weekends and then only differential backups on a nightly basis, to help minimize the time required for backups during the weekday maintenance window.

### Index Maintenance

The remaining time of the maintenance cycle can be devoted to index maintenance and integrity checks (DBCC). It is not realistic and a waste of time to rebuild indexes constantly. Instead, indexes should be reorganized (defragmented) when their fragmentation levels exceed a certain threshold.

Normally in data warehouses, the largest fact tables either have no clustered index or have a clustered index on an identity column that’s a surrogate key that will never be updated. As a result, clustered indexes on the largest fact tables should never have to be rebuilt.

Non-clustered indexes also should ordinarily only require reorganizing (defragmenting.) You can implement defragmentation of non-clustered indexes by running a script or stored procedure that will selectively check for levels of fragmentation that must be addressed. What’s nice about defragmentation is that you can interrupt the process and not lose the work done up to that point in time. For partitioned tables, you can reorganize just the partitions that exceed a given threshold of fragmentation percentage.

If you need to access an index during a rebuild, be sure to use the Online option and utilize parallelism using the MAXDOP option, which can dramatically reduce index rebuild time. If not, you can rebuild offline, which should be somewhat faster.

**Note:** There are many online resources for stored procedures and scripts to selectively rebuild and reorganize indexes based on configurable thresholds. For example, see the CodePlex project [SQL Server 2005 and 2008 - Backup, Integrity Check and Index Optimization](http://olahallengren.codeplex.com/releases/view/52224). For more information about rebuilding and reorganizing indexes, see the SQL Server 2008 R2 BOL article [Reorganizing and Rebuilding Indexes](http://msdn.microsoft.com/en-us/library/ms189858.aspx).

### Integrity Checks (DBCC)

It is equally important to run DBCC checks on your most valuable databases on a regular basis. However, on very large tables, even if you choose minimal options, running DBCC can take quite long time, even days. It’s just not acceptable to run such an I/O-intensive process outside the maintenance window.

Fortunately, DBCC checks do not need to occur nearly as often as backups or index maintenance. In fact, many DBAs are content to run DBCC checks on tables or databases on a rotational schedule. For example, on a database that contains several large fact tables, you might run DBCC on one table per night, using the NOINDEX and PHYSICAL\_ONLY options to speed up the operation. You can also check integrity one filegroup at a time using DBCC CHECKFILEGROUP to subdivide the task.

**Note –** **A Brief Word about Maintenance Plans.** For smaller databases and non-production databases, SQL Server’s database maintenance plans can provide a quick and easy way to set up SQL Agent jobs for backups, index rebuilds, and integrity checks. However, sophisticated as the dialog boxes are, they can never give the same amount of flexibility as T-SQL scripts. In addition, those dialog boxes may hide occasional bugs that are not easily detected. Consequently, most experienced DBAs prefer to write and schedule their own T-SQL scripts and stored procedures and not rely on database maintenance plans for production data.

## Monitoring Tools

Most but not all monitoring of SQL Server instances is performance-oriented. Chapter 5, “Querying, Monitoring, and Performance Tuning,” has extensive coverage of the various tools, counters, and techniques for monitoring SQL Server instances. In this section, we will end with a discussion of monitoring tools that are of interest from the standpoint of database administration.

A full discussion of all the monitoring tools available would take us beyond the scope of this chapter, but here is a partial list of tools at your disposal:

* **Utility Control Point.** SQL Server 2008 R2 has a significant new tool for monitoring servers: the Utility Control Point (UCP). The UCP makes it possible to organize various servers in the Registered Servers window of SSMS and apply scripts and policies to those servers as groups. For a fuller discussion of the UCP, see Chapter 5.
* **Management Data Warehouse.** The Management Data Warehouse (MDW) uses the Data Collector to assemble data from various servers into a central database and apply built-in reports. It can be installed on the same server as the UCP. See Chapter 5 for more information about using the MDW to collect performance data.
* **Central Management System.** The Central Management System (CMS) is a CodePlex project that uses the MDW, PowerShell, and the Policy Management Framework to collect system data from a variety of servers to a central server. Like the MDW, it can be installed on the same server as the UCP. For more information, see [CodePlex Project Functional Specification - Central Management System](http://sqlcms.codeplex.com/).
* **System Center Operations Manager**. The Microsoft System Center Operations Manager (SCOM) can report server-based information that cannot be reported by SQL Server-based tools. With each new release of SCOM and SQL Server, the SQL Server team releases a new Monitoring Management Pack (or “MOM pack”) that adds a selected mixture of queries and Perfmon counter collections suited to monitoring SQL Server and SQL Agent. For more information, see [SQL Server Monitoring Management Pack](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=8c0f970e-c653-4c15-9e51-6a6cadfca363&displaylang=en).
* **Third-party tools.** There is a healthy market in third-party monitoring tools for SQL Server. Most have an evaluation version that you can use to try the tools for a period before purchasing the full version. Usually these tools go beyond what is available from Microsoft for either gathering additional information or at least presenting the information in a friendlier way, so they are always worth checking out.

## Conclusion and Resources

Database administration for a relational data warehouse involves all the same tools as for an OLTP system, but adapts them to the differences required for a data warehouse. For example, a relational data warehouse presents read-only data that is updated on a periodic (usually nightly) basis. As you’ve seen, this means that those databases need to be treated specially, especially in these areas:

* Data warehouse databases are usually directly queried by users, using T-SQL, SSAS cubes, or SQL Server Reporting Services reports. As a result, security planning can differ considerably from that of an OLTP system.
* Data warehouse databases are usually in the Simple recovery model, which has significant consequences for business continuity and maintenance.
* Managing database object changes on very large fact tables presents significant challenges.
* The maintenance window for relational data warehouses is constrained by the need to allow for a data load window for ETL processes.

However, by following the approaches and best practices we’ve covered in this chapter related to these special challenges, the relational data warehouse DBA can effectively manage these challenges.

### Resources

The following is a list of all links referenced in this chapter.

[Performance Tuning Guidelines for Windows Server 2008](http://www.microsoft.com/whdc/system/sysperf/Perf_tun_srv.mspx).

[Fast Track Data Warehouse 2.0 Architecture](http://msdn.microsoft.com/en-us/library/dd459178.aspx)

[SQL Server support in a hyper-threaded environment](http://support.microsoft.com/kb/322385)

[SQL Server Editions](http://www.microsoft.com/sqlserver/2008/en/us/editions.aspx)

[Disk Partition Alignment Best Practices for SQL Server](http://msdn.microsoft.com/en-us/library/dd758814.aspx)

[SQL Server support for mounted volumes](http://support.microsoft.com/kb/819546)

[Pre-deployment I/O Best Practices](http://technet.microsoft.com/en-us/library/cc966412.aspx)

[Microsoft SQLCAT Team: Storage Top 10 Best Practices](http://technet.microsoft.com/en-us/library/cc966534.aspx)

[SQL Server I/O Basics](http://technet.microsoft.com/en-us/library/cc966500.aspx)

[SQL Server I/O Basics, Chapter 2](http://technet.microsoft.com/en-us/library/cc917726.aspx)

[Analyzing I/O Characteristics and Sizing Storage Systems for SQL Server Database Applications](http://msdn.microsoft.com/en-us/library/ee410782.aspx)

[Database File Initialization](http://msdn.microsoft.com/en-us/library/ms175935.aspx)

[How to reduce paging of buffer pool memory in the 64-bit version of SQL Server](http://support.microsoft.com/kb/918483)

[How to troubleshoot the "Cannot generate SSPI context" error message](http://support.microsoft.com/kb/811889)

[How to use Kerberos authentication in SQL Server](http://support.microsoft.com/kb/319723)

[Registering a Service Principal Name](http://msdn.microsoft.com/en-us/library/ms191153.aspx)

[How to: Install SQL Server 2008 R2 from the Command Prompt](http://msdn.microsoft.com/en-us/library/ms144259.aspx)

[How to: Install SQL Server 2008 R2 Using SysPrep](http://msdn.microsoft.com/en-us/library/ee210664.aspx)

[Using the Resource Governor](http://msdn.microsoft.com/en-us/library/ee151608%28SQL.100%29.aspx)

[Moving System Databases](http://msdn.microsoft.com/en-us/library/ms345408.aspx)

[Optimizing tempdb Performance](http://msdn.microsoft.com/en-us/library/ms175527.aspx)

[AdventureWorks 2008R2 RTM](http://msftdbprodsamples.codeplex.com/releases/view/45907)

[Microsoft SQL Server 2008 R2 Feature Pack](http://www.microsoft.com/downloads/details.aspx?FamilyID=ceb4346f-657f-4d28-83f5-aae0c5c83d52&displaylang=en)

[CodePlex Project Functional Specification - Central Management System](http://sqlcms.codeplex.com/)

[The Impact of Changing Collations and of Changing Data Types from Non-Unicode to Unicode](http://blogs.msdn.com/b/sqlcat/archive/2007/12/06/new-best-practices-whitepaper-published-the-impact-of-changing-collations-and-of-changing-data-types-from-non-unicode-to-unicode.aspx)

[Working with the New SQL Server 2008 Collations and Earlier Versions of the Native Data Provider](http://sqlcat.com/whitepapers/comments/1018.aspx)

[Ownership Chains](http://msdn.microsoft.com/en-us/library/ms188676.aspx)

[sp\_estimate\_data\_compression\_savings (Transact-SQL)](http://msdn.microsoft.com/en-us/library/cc280574.aspx)

[Top 10 Best Practices for Building a Large Scale Relational Data Warehouse](http://sqlcat.com/top10lists/archive/2008/02/06/top-10-best-practices-for-building-a-large-scale-relational-data-warehouse.aspx)

[Partitioned Table and Index Strategies Using SQL Server 2008](http://technet.microsoft.com/en-us/library/dd578580%28SQL.100%29.aspx)

[Linking Servers](http://msdn.microsoft.com/en-us/library/ms188279.aspx)

[CREATE CREDENTIAL (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189522.aspx)

[Creating SQL Server Agent Proxies](http://msdn.microsoft.com/en-us/library/ms189064.aspx)

[Managing Schema Changes (Part 1)](http://blogs.msdn.com/b/sqlcat/archive/2006/03/01/541550.aspx)

[Managing Schema Changes (Part 2)](http://blogs.msdn.com/b/sqlcat/archive/2006/03/31/566046.aspx)

[Team Foundation Server 2010](http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx)

[Using Version Control](http://msdn.microsoft.com/library/ms181368.aspx)

[SQL Server 2008 Failover Clustering](http://sqlcat.com/whitepapers/archive/2009/07/08/sql-server-2008-failover-clustering.aspx)

[Six Failover Clustering Benefits Realized from Migrating to SQL Server 2008](http://sqlcat.com/top10lists/archive/2008/11/20/six-failover-clustering-benefits-realized-from-migrating-to-sql-server-2008.aspx)

[SQL Server 7.0, SQL Server 2000, and SQL Server 2005 logging and data storage algorithms extend data reliability](http://support.microsoft.com/kb/230785#appliesto)

[Client-side Redirect problem](http://social.msdn.microsoft.com/Forums/en-US/sqldatabasemirroring/thread/5783bbe7-bfc8-4342-94a7-e83b07111cbb)

[Database Mirroring and Log Shipping](http://msdn.microsoft.com/en-us/library/ms187016.aspx)

[Database Mirroring and Log Shipping Working Together](http://sqlcat.com/whitepapers/archive/2008/01/21/database-mirroring-and-log-shipping-working-together.aspx)

[Tail-Log Backups](http://msdn.microsoft.com/en-us/library/ms179314.aspx)

[SQL Server 2005 and 2008 - Backup, Integrity Check and Index Optimization](http://olahallengren.codeplex.com/releases/view/52224)

[Reorganizing and Rebuilding Indexes](http://msdn.microsoft.com/en-us/library/ms189858.aspx)

[CodePlex Project Functional Specification - Central Management System](http://sqlcms.codeplex.com/)

[SQL Server Monitoring Management Pack](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=8c0f970e-c653-4c15-9e51-6a6cadfca363&displaylang=en)

[SQLCat whitepaper: Running SQL Server 2008 in a Hyper-V Environment - Best Practices and Performance Recommendations](http://sqlcat.com/whitepapers/archive/2008/10/03/running-sql-server-2008-in-a-hyper-v-environment-best-practices-and-performance-recommendations.aspx)