Realtime publishers

Intelligently Reducing SharePoint Costs through Storage Optimization

Don Jones

sponsored by

AvePoint®

Unleashing the Power of SharePoint™

Introduction to Realtime Publishers

by Don Jones, Series Editor

For several years now, Realtime has produced dozens and dozens of high-quality books that just happen to be delivered in electronic format—at no cost to you, the reader. We've made this unique publishing model work through the generous support and cooperation of our sponsors, who agree to bear each book's production expenses for the benefit of our readers.

Although we've always offered our publications to you for free, don't think for a moment that quality is anything less than our top priority. My job is to make sure that our books are as good as—and in most cases better than—any printed book that would cost you $40 or more. Our electronic publishing model offers several advantages over printed books: You receive chapters literally as fast as our authors produce them (hence the "realtime" aspect of our model), and we can update chapters to reflect the latest changes in technology.

I want to point out that our books are by no means paid advertisements or white papers. We're an independent publishing company, and an important aspect of my job is to make sure that our authors are free to voice their expertise and opinions without reservation or restriction. We maintain complete editorial control of our publications, and I'm proud that we've produced so many quality books over the past years.

I want to extend an invitation to visit us at http://nexus.realtimepublishers.com, especially if you've received this publication from a friend or colleague. We have a wide variety of additional books on a range of topics, and you're sure to find something that's of interest to you—and it won't cost you a thing. We hope you'll continue to come to Realtime for your educational needs far into the future.

Until then, enjoy.

Don Jones

Copyright Statement

Chapter 1: The Problem with SharePoint Storage

We've been promised a world where SharePoint, in many ways, becomes our entire intranet. At the very least, SharePoint is marketed as a means of centralizing all our shared data and collaboration efforts. Conference speakers tell us that we should migrate our shared folders into SharePoint, integrate SharePoint with back-end databases, and make SharePoint the "dashboard" for all our users' information needs.

In many regards, SharePoint can do all of that—but the price can be prohibitive. Why? That's what this chapter is all about: The *problems* that can arise when SharePoint becomes the centerpiece of your information sharing and collaboration. That's not to say we can't make SharePoint do the job. On the contrary, we can make SharePoint fulfill its marketing hype and much more—if we use the right techniques to overcome some of its inherent hurdles.

The SharePoint Vision: Everything, in One Place

Microsoft's vision for SharePoint is for it to become the central, single location for all your information-sharing needs. The problem with many of today's environments is the sheer amount of data that users need access to, and the fact that the data is scattered all over the environment. For example, consider Figure 1.1. Users access information from shared folders on file servers, from public folders in Exchange, from line-of-business application databases, and much more. Simply teaching new users *where* all this information lives is time consuming and challenging, and finding the right data at the right time can be bewildering for even experienced users.

All these different information repositories have their own means of access, too. Shared folders typically rely on Server Message Block (SMB) protocols, while Exchange public folders may be accessible via the Internet Mail Access Protocol (IMAP), Outlook Web App (OWA), Remote Procedure Calls (RPCs), and more. Line-of-business data—even basic summary data that you might want to glance at now and again throughout the day—might use entirely different protocols. Making sure users have access to everything from everywhere—from the office to their homes, including computers and mobile devices—is challenging and often impractical.

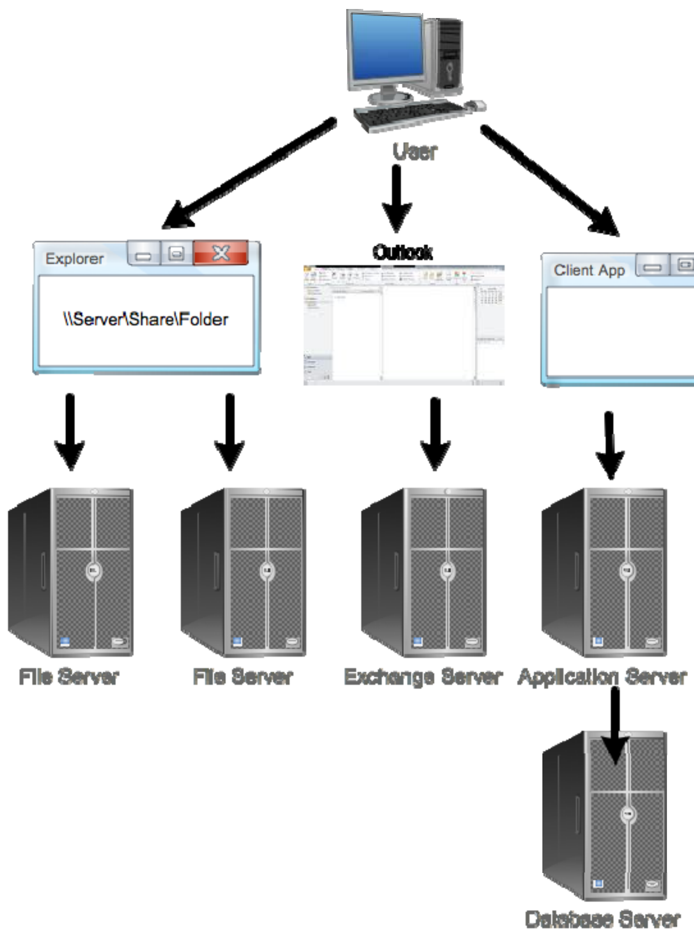

Figure 1.1: Users access information from too many places.

There are additional problem with this scattered access. For example, shared files living on a file server aren't version-controlled, making it all too easy for a user to accidentally delete or change something they shouldn't have. This mistake then forces an administrator to resort to a backup. Newer versions of Windows support a Volume Shadow Copy Service (VSS) feature that can help with the problem, but it's a time-based snapshot. That means it won't capture every version of a changed file, so you can still end up losing valuable information.

SharePoint proposes to solve this business problem by centralizing everything into a single location. As Figure 1.2 shows, users can continue to employ whatever means they like to access the data—including Microsoft Outlook—but the primary access is through a Web browser. The benefit of this technique is that Web browsers exist on nearly every modern computer and mobile device, and use a simple protocol that can be initiated from anywhere in the world. Suddenly, all that shared data is centrally available through a single interface.

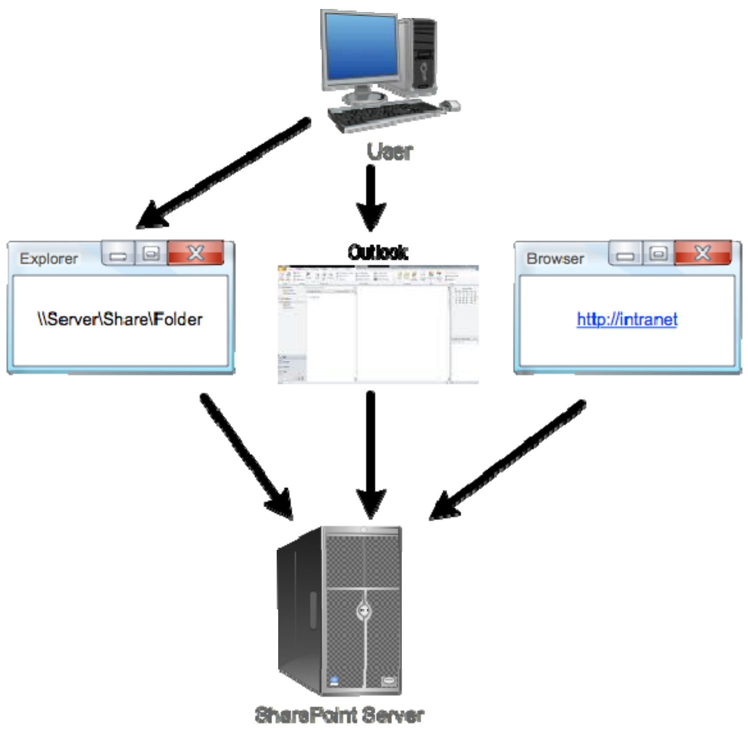

Figure 1.2: Centralizing everything in SharePoint.

Even business data from back-end databases can be integrated into SharePoint dashboards, while remaining in their original databases. This makes SharePoint a single "portal" for corporate data; in fact, the first version of SharePoint was called *SharePoint Portal Server*, an early suggestion of this all-in-one vision.

SharePoint can not only centralize all this information but also make it version-controlled, indexed, and searchable. Now, users can find data more easily, and the data is protected against accidental change or deletion through a version-controlled repository.

That's SharePoint's promise, and it's primarily delivered through the idea of having everything contained in a single, central repository. That repository, unfortunately, is exactly what introduces many of SharePoint's most significant challenges.

The SharePoint Content Repository: It's Just a Database

SharePoint's repository—where all its content lives, is indexed, and is version-controlled—isn't some special data construct. It's just a database—a SQL Server database, to be specific. Modern versions of SharePoint are well-tuned to support content databases in the multi-terabyte range, meaning SharePoint should be able to handle whatever you throw at it in terms of storage. In fact, the main reason that companies split their SharePoint content across multiple databases is to reduce things like backup and recovery time, not because SharePoint can't scale to handle their content storage needs.

So let's just be clear on one point: From a technical perspective, SharePoint can handle a *lot* of data. Probably in the tens of terabytes. Database size limitations are *not* the problem with the SharePoint content repository. But let's step deeper inside for a moment, and look at how that database works.

SQL Server stores data on disk in 8KB chunks called *pages.* When SQL Server needs to change a single byte of data, it loads—at a minimum—8KB of data off disk, makes the change in memory, then writes that 8KB page back to disk. Normally, SQL Server has to store an entire table row within a single page, meaning a single row of data can't exceed that 8KB limit (the actual number is slightly smaller, since each page has a small amount of overhead for management data). However, SQL Server does allow a row of data to contain a *pointer* to larger pieces of data, which can then be spread across multiple pages. Figure 1.3 illustrates this storage mechanism, with a single row of data on the first page, containing a pointer to several sequential pages that contain a large string of data—perhaps a photo, a Word document, or some other large piece of information.

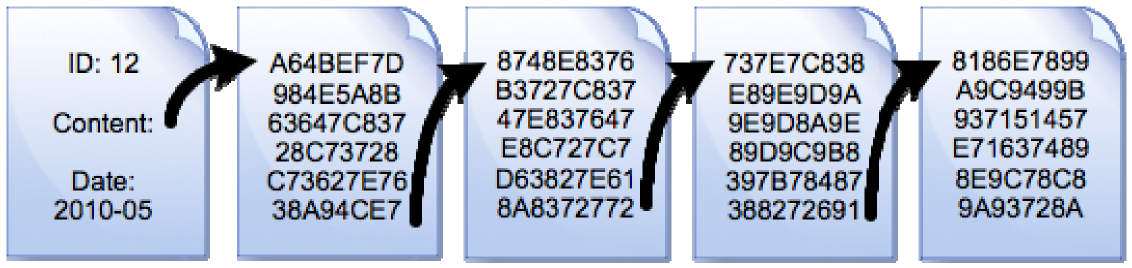
ID: 12
Content:
Date: 2010-05
A64BEF7D 984E5A8B 63647C837 28C73728 C73627E76 38A94CE7
8748E8376 B3727C837 47E837647 E8C727C7 D63827E61 8A8372772
737E7C838 E89E9D9A 9E9D8A9E 89D9C9B8 397B78487 388272691
8186E7899 A9C9499B 937151457 E71637489 8E9C78C8 9A93728A

Figure 1.3: Storing data on pages in SQL Server.

SQL Server refers to these large objects, which are stored as binary data, as *binary large objects* (BLOBs)—surely one of the most charming acronyms in IT!

Note
We'll dive deeper into SQL Server's storage mechanisms, and some of the other subtle problems that BLOBs can create, later in this chapter.

It turns out, however, that SQL Server isn't as amazing with BLOBs as it is with the smaller pieces of data it normally deals with. Streaming a BLOB into the database, or reading it out of the database, simply isn't what SQL Server is best at. That's not to suggest SQL Server's performance is horrible or anything, but even Microsoft has spent years trying to come up with alternative ways of storing the information that are faster and more efficient. In SQL Server 2008, for example, Microsoft added the FILESTREAM data type to SQL Server, which allows BLOBs to be stored as simple files on the file system, with a pointer inside the database. The idea is that Windows' file system *excels* at reading and writing large files, so why not let it do that? Of course, with some of the data living outside the actual database, tasks like replication, backup, and recovery can become more complicated, but the upside is increased performance.

A deeper problem with large SharePoint databases—whether they're full of BLOBs or not—is that they simply take up a lot of room on disk, and data center-quality disk storage still isn't cheap. You might be able to buy a dozen 1TB desktop-class hard drives for $1800, but just the *cabinet* for a 12-bay storage area network (SAN) can run $14,000—a fully-populated 12TB SAN can run to $30,000. So, although SharePoint's database might not flinch at storing that much data, doing so can cost a lot. Plus, you're going to have to find a way to back it all up, and be able to recover it quickly in the event of a disaster or failure.

A more subtle challenge with SharePoint storage is when you start enabling version control. Every time someone modifies a SharePoint-based file, you're creating a new version of that file—and the old version remains in the database. So the database can get quite large, quite quickly. SharePoint also needs database storage to index the file so that it can quickly locate files based on keyword searches by users. We *want* those features—it would just be nice if we could find a way to have them take up a bit less space.

The idea, then, is to identify the specific problems associated with specific types of "problem content," and to find ways to address those problems *while still meeting the SharePoint vision* of "everything in one place." The general phrase for what we're trying to do is *SharePoint storage optimization*, meaning we're seeking to optimize our use of SharePoint storage to reduce our storage costs, while still maintaining a fully-functional SharePoint infrastructure that offers all the benefits that SharePoint offers.

Specific Problems with Specific Kinds of Content

Let's begin by examining specific types of problem content. By "problem," I mean that these forms of content can bloat the SharePoint database—perhaps not reducing performance, but definitely increasing your storage costs and making tasks like backup and recovery more complicated. We're not going to take the "easy" route and simply say, "don't store this information in SharePoint;" our goal is to use SharePoint the way it's meant to be used—but to do so with a bit more control over our storage utilization.

Large Content Items

First and foremost are the file attachments stored in SharePoint, which I'll refer to as *large content items.* Word documents, PowerPoint presentations, Excel spreadsheets, Photoshop illustrations, Acrobat PDFs, you name it. Traditionally, we would just have dumped these onto a file server and let people access them from there, but with a file server, we're not getting integrated enterprise-wide searching, nor are we getting version control—which could certainly be beneficial for at least some of the files in your environment. SharePoint offers those features, but these large items can take up a lot of room in the database, increasing your storage costs. In addition, as I've already mentioned, SQL Server isn't necessarily at its best when working with these large content items; if there was a way to move them outside the database—and still have them be "inside" SharePoint, of course—then we could perhaps improve performance a bit as well as optimize our storage.

Note
I'll cover large content items in more detail, including storage optimization techniques, in Chapter 2.

Shared Folders and Media Files

Obviously, the information in shared folders qualifies as "large content items," so all the caveats I described in the previous section still apply. Media files—audio and video files—obviously fall under the same category, as video files in particular can be *very* large.

But they have some unique problems above and beyond their mere size. Simply getting this content *into* SharePoint can present an enormous challenge: You need to locate the data, copy it into the SharePoint database, create the necessary SharePoint items to provide access to the data, and—perhaps most importantly—apply the appropriate permissions to the content so that SharePoint's access permissions reflect the original permissions of each file. You'll be adding considerable size to your SharePoint database in the process, of course, but you'll get the advantages of SharePoint's features, including permissions management, workflows, alerts, and versioning, along with indexing and search. Figure 1.4 illustrates the logical migration process.

There are a number of vendors who offer tools to assist with, and automate, this kind of data migration. However, be aware that this kind of migration isn't always the optimal way to use SharePoint, at least in terms of storage optimization.

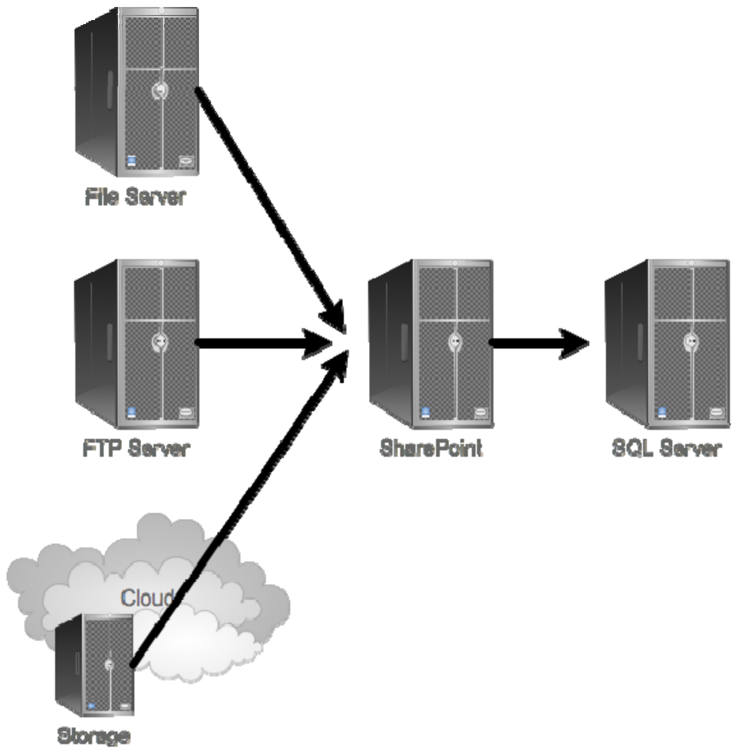

Figure 1.4: Migrating content into SharePoint.

Note

Chapter 3 will dive into this type of content in more detail, and suggest ways in which you can obtain the benefits of having the content in SharePoint, while optimizing your storage utilization.

Notice that the source repository for these migrations can come in a number of forms: typical Windows file servers, of course, but also cloud-based storage or even FTP servers. The basic idea is that *any* file, no matter where it's located, can become more valuable and collaborative once it's inside SharePoint—assuming, of course, that you want to devote enough storage to keeping it all in the repository, or that you have another way of incorporating the information *without* actually migrating it into the database.

Offsite Content?

Why in the world would we want to include FTP- or cloud-based content in our SharePoint infrastructure? Simple: There are a number of good business reasons to include the "primary copy" of content in a cloud-based storage system, on an FTP server, or elsewhere. Recoverability is one reason: Cloud-based storage can offer better protection against deletion or failure. Accessibility is another reason: We might have need for others to access the data, and cloud- or FTP-based storage both offer easy ways for anyone in the world to get at the information.

Sometimes data in a cloud- or FTP-based storage system might be *someone else's* data that our company has access to; being able to include that in SharePoint would make it easier for our users to access, without requiring us to actually "own" the data.

So there are definitely situations where we would want to bring in content from a cloud-based storage system, or even an FTP server, without actually "migrating" that data to live entirely within SharePoint. This may be a tricky requirement, as most of SharePoint's features typically require content to "live" in the database, but by identifying this as a potential need, we can be on the lookout for a solution, technology, or trick that might let us meet that need.

Aside from the storage implications, there might seem to be one other significant downside of moving content into SharePoint: retraining your users. For years, you've taught them to used mapped drives, or possibly even UNC paths, to get to their shared files. Now, they have to learn to find their files inside SharePoint document libraries. Newer versions of Office can help alleviate the retraining pain because users can directly access documents from those libraries, but for non-Office files—or if your users are used to older versions of Office—there's still some retraining to be done. There's good news, though: Usually, retrained users have an *easier* time working with documents that are in SharePoint, so there's definitely a benefit associated with that retraining investment.

I want to spend a few moments discussing the specific challenges associated with streaming media—meaning audio and video. First, these files tend to be *large,* meaning they'll take up a lot of space in SQL Server and place a greater demand on SQL Server to retrieve them from the database. They can also place burdens on SharePoint's Web Front End (WFE) servers, because those Web servers have to retrieve the content from the database *and* stream it—in a continual, literal stream of data—to users. In fact, this kind of media content is the one thing I often see companies *excluding* from SharePoint, simply out of concern for what it will do to SharePoint's performance. This book will have a specific goal of addressing this kind of content, and identifying ways to include it in SharePoint *without* creating a significant database or WFE impact.

Dormant or Archived Content

Perhaps one of the biggest drains on your SharePoint storage is old content that's no longer needed for day-to-day use or that hasn't been used in a significant period of time but still can't be permanently deleted. Most organizations have a certain amount of data that qualifies as "dormant" or "archival," such is particularly the case for organizations that have a legal or industry requirement to retain data for a certain period of time.

Even if all you have in the way of shared data is file servers, you probably know that the *majority* of the files they store isn't used very frequently. Think about it: If your SharePoint servers only needed to contain the data that people *actually accessed on a regular basis,* the database probably wouldn't be all that large. The problem is that you *also* need to maintain a way to access all that dormant and archival data—and *that* is often where SharePoint's biggest share of storage utilization comes from, especially when that dormant or archived data consists of large content items like file attachments. It'd be great to pull that information *out* of SharePoint, but then it would no longer be indexed and searchable, and when someone *did* need to access it, they'd have no version control, no alerts, no workflow, and so forth.

I've seen organizations create tiered SharePoint libraries, like the one Figure 1.5 shows. The idea is that "current" content lives in a "production" SharePoint server, with its own database. Older or dormant content is moved—either manually or through some kind of automated process—into an "archival" SharePoint installation, with its own database. The archival database isn't backed up as frequently, may live on older, slower computers, and in general costs slightly less.

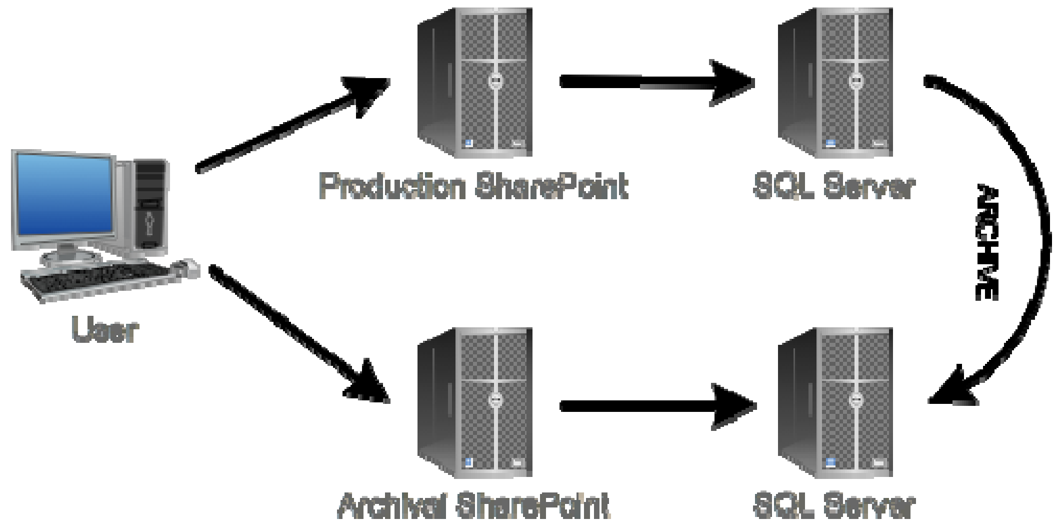

Figure 1.5: Tiered SharePoint storage.

Properly done, you can even maintain a single set of search indexes so that users can find older content. The problem is that older content becomes second-class, might be harder to get to in terms of performance, and *still* takes up space in a SQL Server database. This type of tiered storage isn't necessarily ideal for every company, although it's on the right track toward a better solution.

Note
Chapter 4 will dive into specific techniques for better managing dormant and archival SharePoint content.

There's a bit more to this dormant/archival content picture, and that's how you actually identify dormant or archival content and move it out of SharePoint—while somehow leaving it "in" SharePoint so that it's still searchable and accessible. Let's face it: If you expect users, or even administrators, to manually identify "old" content and mark it for archival in some fashion, it's pretty much never going to happen. So you need to create some kind of automated, non-manual process that can identify content that hasn't been accessed in a while, apply customizable business rules, and automatically migrate content into some other storage tier—without "removing" it from SharePoint, of course.

As you'll see in Chapter 3, "dormant" content can consist of a lot more than the odd rarely-used file. In fact, if you've really been *using* SharePoint, you might have entire *sites* that are dormant—perhaps ones associated with a now-completed project—and you want to dismantle them without making them permanently unavailable. You might want to treat old versions of files as "dormant," while leaving the current and most-recent versions in your "production" SharePoint site—but you don't want to permanently delete those old versions. You might even be *required* to maintain older content, for regulatory reasons, but you don't see any reason to bog down your day-to-day SharePoint operations to do so. There are lots of reasons to want to tier your SharePoint storage, and we're going to need to investigate some of the methods that will let you do so.

SharePoint Storage Technical Deep Dive

I've touched briefly on how SharePoint stores its data, but if we're going to make any headway in optimizing our SharePoint storage, we need to understand that storage mechanism in much greater detail. Here's a deep dive on how SharePoint storage works.

How SQL Server Stores Data

SQL Server consists of a service, which opens database files on disk. You can think of the database file as a kind of proprietary second-level storage system, meaning it is a way for SQL Server to organize data in a manner that facilitates SQL Server's job and performance goals. The database itself sits on a disk drive, and access to the file is made through Windows' own file systems. Figure 1.6 outlines the high-level structure.

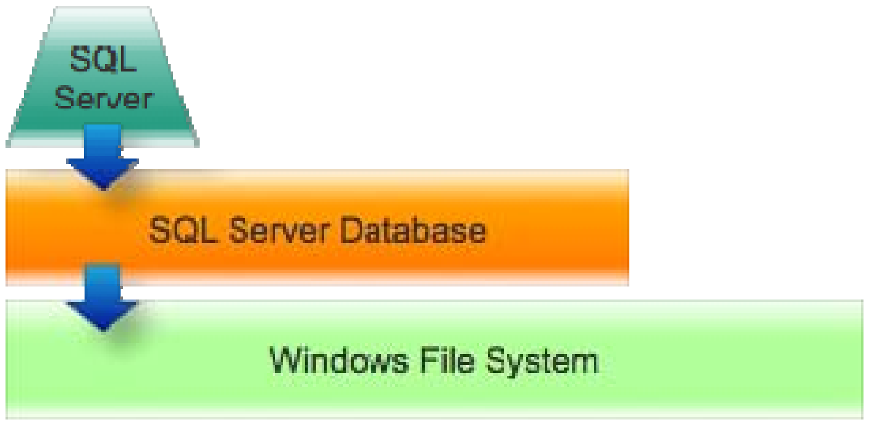

Figure 1.6: High-level SQL Server storage.

As I already described, SQL Server stores data in 8KB chunks called *pages*. That is strictly a SQL Server data-management paradigm; the actual data is still written to the disk in the form of *disk blocks,* which are usually smaller than 8KB. For example, if the drive was formatted to use 1KB disk blocks, SQL Server would be writing eight of those blocks to the file system each time it saved a page to the database. Figure 1.7 illustrates this deeper-level look at the storage architecture.

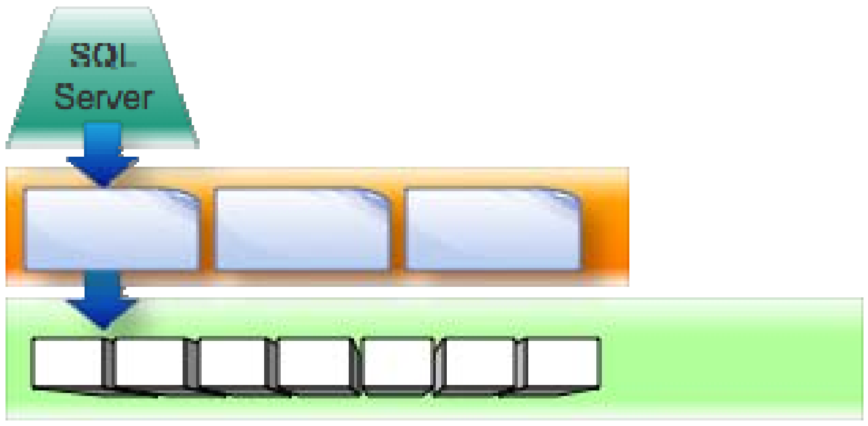

Figure 1.7: Storing data in pages and disk blocks.

This form of storage can have ever-deeper impacts on SQL Server's performance. As I already explained, SQL Server generally requires that a single row of database data live within a single 8KB page; smaller rows can share a page. Because SQL Server reads data in 8KB chunks, storing more rows per page means that SQL Server can read more data in a single operation. Conversely, a data row occupying 4.1KB can result in a lot of wasted disk throughput because SQL Server can only fit a single such page on an 8KB row but must read and write that entire 8KB, even though only slightly more than half of the 8KB actually consists of data.

How Windows Stores Data

Windows' NTFS stores data in disk blocks, or *clusters,* and their size is determined when you format a new logical disk. The sizing theory goes something like this:

- Smaller disk blocks mean less wasted space but require more work for Windows to read and write when large, multi-block files are involved.
- Larger disk blocks mean the potential for more wasted space but allow Windows to read and write larger files in fewer discrete operations.

For a volume containing SQL Server databases, it's almost ideal to use an 8KB cluster size, as this aligns SQL Server's own storage with the file system's smallest unit of work. Some experts recommend a larger cluster size of 64KB, meaning every file system-level disk read will pick up eight SQL Server pages.

The point is that the Windows file system is *really good* at reading large blocks of disk space into memory, and at writing changes to large blocks of disk space. You have a lot of flexibility to optimize the file system's behavior in this regard, depending on the size of files you're working with. Really, the high-level lesson is that *the file system is very good at storing files.* It does not necessarily need another layer added atop it if all you're doing is storing large items like file attachments. Yes, if you're going to be storing *relational* data, such as an actual database, a different means of organizing that data can vastly improve performance—which is why SQL Server has its own database structure rather than just storing data in tiny little files all over the disk. But that doesn't mean an intermediate storage layer like SQL Server is *always* going to offer the best performance.

SharePoint: All About the BLOBs

When SQL Server needs to store a large mass of data, such as a file attachment, it does so in the form of a BLOB. BLOBs consist of a link, or pointer, within the actual row data. That link or pointer then connects to one or more 8KB pages that store the actual BLOB data. So, in SharePoint, suppose that a single document entry takes up a single 8KB page for the entry itself. If a document entry includes a 50MB PowerPoint file attachment, the total entry will consist of more than 6000 pages in the database.

SQL Server will never need to read a *portion* of those 6000 pages—when you retrieve the PowerPoint file, you're going to retrieve *all* of it. That means those 6000 pages will only ever be read sequentially, all at once. The problem is that they might not be *stored* sequentially. SQL Server writes pages to the database beginning in the first available spot, so those 6000 pages may require SQL Server to jump around a bit, finding free spaces for all of them. The result is a *fragmented database,* meaning SQL Server will need to jump around within the database file to re-assemble that PowerPoint attachment. Further, the actual disk blocks storing those pages might not be contiguous (and that's often the case when a database grows beyond its initial size), so the operating system (OS) may have to jump around quite a bit to piece together those 8KB pages. All that disk I/O, at both the database and file system level, can slow SQL Server a bit. Although SQL Server is *designed* to perform this kind of operation, when it has to do so for hundreds or thousands of users at once, its performance definitely sees an impact.

The fact is that *most* of SharePoint's storage will be given over to BLOBs. If you import 5TB of files, you're going to be using 5TB of BLOB-style storage (potentially more, actually, because having to store the data in 8KB chunks will usually result in some "wasted space" at the end of each BLOB sequence). You'll also be adding the SharePoint data entries to keep track of those files, but as a percentage of the total data, those entries are negligible. In most SharePoint installations, upwards of 90% of your database size will be given over to BLOBs, so figuring out how to optimize that storage can have a significant impact.

Why BLOBs Are Bad for Databases

Keep in mind that SQL Server's *main* design point is to read and write *single pages* of data, or at the worst, a *few* pages of data, at once. Most SharePoint operations—updating a document's permissions, or changing its name—require that very few database pages be modified, and SQL Server does a great job of it. When SQL Server has to start behaving like a file system, however, it's not working in its best "comfort zone," and so you can start to see performance differences.

In fact, when it comes to dealing with large, sequential data structures—like file attachments—SQL Server is essentially adding an unnecessary second layer to the equation. Because those file attachment BLOBs aren't the kind of structured, relational data that SQL Server is deigned for, SQL Server really *is* taking on some of the attributes of a file system—but it's also *sitting on top of* a file system.

One trick, then, is to offload the BLOBs *to the file system,* which excels at moving huge lumps of data from place to place. The file system *is already involved* in SQL Server's BLOB manipulation, so taking SQL Server "out of the stack" can help improve performance. In fact, in the next chapter, I'll discuss some of the techniques Microsoft has created—including External BLOB Storage and Remote BLOB Storage—to help offload BLOB storage from SQL Server and into the file system or other storage mechanisms that are better suited to mass-storage.

Why We Put BLOBs into SharePoint

Let's consider every file in a SharePoint document library to consist of two main parts: The file *metadata* and the file attachment itself. The metadata consists of things like keywords, permissions, update dates and times, and so forth; the file itself is the actual file data stored in a Word file, a PowerPoint file, or whatever.

The metadata provides most of SharePoint's features, allowing it to coordinate workflows, maintain version information, send alerts, and so forth. But SharePoint also needs access to the actual file because its search indexing engine wants to open those files and scan them for keywords. By doing so, it builds a search index, which is employed to help users quickly locate files by using keywords. So although it is technically possible to separate the metadata from the file itself, it isn't desirable to do so unless that separation can be done in a way that still provides SharePoint's indexing engine access to the file.

In fact, as you'll see in the second chapter, Microsoft's official BLOB-offloading technologies seek to do just that. They essentially wedge themselves into SQL Server's brain so that when SQL Server needs to store or retrieve a BLOB, the BLOB data goes elsewhere rather than into the database itself. Because this "wedge" occurs within SQL Server, applications—like SharePoint—don't need to realize that it's happening. They "think" the BLOBs live in the database, and SQL Server makes the BLOBs available as if they *were* in the database, so SharePoint can continue working with all the files as if they were in the database—even though they're not. But that's not necessarily the only approach to reducing the size of the SharePoint database and improving SharePoint's database performance. In fact, one reason BLOB offloading isn't the "perfect solution" is because it still requires that content be migrated *into* SharePoint to begin with—and that migration project might be one that you want to avoid, if possible.

Goals for Content

Now that you're aware of the technical underpinnings of SharePoint's storage, and of some of the general directions that a solution might take, let's lay out business goals for SharePoint storage. Some of these might seem contradictory at this point, but that's okay—we're after the "perfect world" set of capabilities, and we'll work out in the next few chapters whether we can have them all.

Keep in mind that these goals apply, potentially, to *all* the shared and collaborative data in your environment. Right now, it might not all be in SharePoint. Whether it is, isn't, or should or should not be is not the consideration right now. SharePoint is a means, not a goal in and of itself. What we're going to review now are our *goals,* and we'll determine later whether SharePoint can be made to meet these goals.

Location-Unaware

As I wrote earlier, there are some advantages to keeping content elsewhere, especially off-site. We want to be able to include content in SharePoint *regardless* of where the content is located, and in some cases—as I'll outline in a bit—we may have good business reasons for *not* migrating that content into the SharePoint database.

Alert-Enabled

We want all of our content to be alert-enabled. Alerts provide a way for users to "subscribe" to an individual document and to be notified of any changes to it. This might allow a document owner, for example, to be notified when someone else has made changes to a document; it might allow users who rely on a document—such as current sales specials or personnel policies—to be notified when changes have been made that they ought to review and become familiar with.

This is something that SharePoint offers, but we need to figure out whether we can practically and affordably include *all* of our content in SharePoint. Ideally, we *do* want all of our content included in SharePoint in some way so that any piece of content can be subscribed for alerts.

Metadata- and Tagging-Enabled

SharePoint allows for content to have predefined and custom metadata attached to it, along with user-defined tags. Companies use these features to attach additional meaningful keywords to content, and to classify content. For example, companies might use metadata to identify a content item as “confidential” or to associate it with a particular project. Of course, the content has to live in SharePoint’s database in order for this support to exist—but we want these features for *all* of our content.

These feature in particular can make long-term content management easier, and can enable users to locate content more easily and quickly by using common keywords that might not appear within the content body (especially for media files like videos, which don’t have a written “body” for keywords to appear within).

Workflow-Enabled

We don’t necessarily want every single document modified by everyone in the environment, but we might be open to everyone *suggesting* changes. One way to achieve that is to apply workflow to our documents. Workflow enables a user to modify a document and submit it for approval, either to a group of reviewers or to a single reviewer. Before the modified document becomes the official “current” version, the modifications would have to be approved by some predetermined set of approvers.

SharePoint offers this functionality but only for content that resides within its database. In other words, we again need to see whether it’s practical and affordable to include *all* of our content inside SharePoint so that we can enable workflow on whatever pieces of content we feel require it.

Version-Controlled

Another SharePoint feature is the ability to keep past versions of documents. Unlike a tape backup or even Windows’ VSS, SharePoint doesn’t create a new version on a scheduled basis. Instead, it creates a new version of a document whenever someone modifies the previous version—ensuring that we have every version of the document that ever existed, if desired. Users with appropriate permissions can access older versions of a document, compare it with other versions, and even make an older version the current, “official” version for other users to access.

In an ideal world, we’d have the option for versioning for every document in the entire enterprise—but normally, SharePoint can only do this for documents that live within its repository. So once again, we need to decide whether we can afford to include *everything* within SharePoint.

Secured

Most of today's data repositories support some kind of security. The Windows file system, for example, has a very granular security system. What would be nice is if we could manage access to *all* of our shared data *in a single place.* Obviously, SharePoint is a candidate to be that place because it too supports a robust and granular security system. In fact, because its security information lives in a database rather than being distributed across individual files and folders, SharePoint is arguably a *better* way to manage storage, offering the potential for easier security reporting, auditing, and so forth.

Again, however, we can only get those advantages if *all* of our content lives in the SharePoint database—which may or may not be practical or affordable.

Indexed and Searchable

One of SharePoint's biggest advantages is its ability to index the content in its database, and make that content searchable for your users. It's like having your own private Google or Bing search engine that is accessible only to employees and that includes all of your enterprise data. SharePoint's indexing system is security-aware, meaning it won't show users search results for things they don't have permission to access in the first place. Of course, in order to be indexed, SharePoint needs all your content to *move into the database.* Even if you've decided that you'll pay whatever storage costs are needed to make that happen, there's still the significant project of getting your data into that database.

Minimal Database Impact

Here's where our business goals start to contradict each other. We want all of the above capabilities—searching, security, alerts, workflow, and so on—but we want minimal impact on the SQL Server databases that support SharePoint. We want those databases to perform at maximum efficiency at all times, and we ideally want them to take up as little space as possible—simply because "space" costs money to acquire and to protect and maintain.

Minimal WFE Impact

I described earlier how streaming media files can sometimes have a negative impact on SharePoint's WFE, so we want to avoid that impact. We still want our media files "in" SharePoint somehow—so that they can be indexed, searched, and managed just like any other form of content—but we don't want to do so in a way that will create a burden for the WFE.

Minimal Migration

We also want all of the above goals *without* having to spend time and money migrating data into SharePoint. Although great migration tools exist, any migration project *is* a project. We might decide that some degree of content migration is acceptable, but we don't want migration to be a hard-and-fast pre-requisite for gaining the above capabilities for *all* of our content.

In other words, we want to be able to use all SharePoint's features *without* necessarily putting all of our content into SharePoint's database. Seem contradictory? Sure—and it's a big part of what we'll be examining in the next three chapters.

Transparent to Users

Based on the above, seemingly-conflicting goals, we're likely going to be looking at some kind of hybridized system that involves SharePoint, SQL Server, perhaps some kind of BLOB offloading, and likely other techniques. With that possibility in mind, let's make one last, formal business requirement of whatever solution we come up with: *Our users can't know.*

The goal here is to get *all* of our content into a centralized SharePoint infrastructure so that our users can access *all* of their content in *one* consistent fashion. That's SharePoint's high-level vision, and we have to maintain it. We can't start throwing wrenches into the system that require users to go *here* for some content, *there* for other content, and *over there* for still more content; it all needs to be in one place. Whatever we're doing to archive old content, for example, still has to make it *look like* that content still lives in SharePoint, even if it really doesn't. This is perhaps our ultimate business goal, and any solution that doesn't meet at least this goal is one that we will have to set aside as unsuitable.

Coming Up Next

Now that we've defined the major challenges and goals for SharePoint content, it's time to start looking at specific solutions. In the next chapter, I'll focus on large content items, which often create the first and most difficult challenge that companies face with SharePoint. We'll look at the advantages of including this content in SharePoint, and the difficulties that arise when you do so. We'll also focus on the core techniques that can allow large content to be integrated within SharePoint, while avoiding most of those difficulties.

So let's quickly review what's ahead: In Chapter 2, I'll look at the specific techniques we can use to reduce the size of the SQL Server database while still leaving our content "in" SharePoint. We'll dive into the details of the BLOB-offloading technologies I introduced in this chapter, along with other approaches. We'll get pretty technical because it's the subtle details in each approach that really make a difference.

Chapter 3 will be about optimizing SharePoint storage for external or legacy content. We'll look at all the content living on file servers and other "old school" forms of storage as well as content living in external stores like FTP servers or cloud-storage systems. I'll also focus in on those streaming media files that can be so problematic for some SharePoint environments. We'll look at ways to include this content in SharePoint, while still trying to meet our business goals for transparency, database impact, and so forth.

Finally, in Chapter 4, we'll look at optimizing SharePoint storage for dormant and archived content. This will be a tricky chapter because we have to not only find a way to keep our SharePoint databases trim and efficient but we also concoct some means of automatically identifying and moving dormant data into another storage tier—while of course keeping it transparently accessible to SharePoint users. The industry has worked up some clever techniques for this, and I'm excited to share some of them with you.

There's sort of a theme for this book, and it's this: *Getting all of your content into SharePoint without blowing your storage requirements through the roof.* If you're already using SharePoint, you'll find that this theme also helps you *lower* your existing SharePoint storage requirements, hopefully without giving up any SharePoint features for any of that content.