Realtime publishers

Intelligently Reducing SharePoint Costs through Storage Optimization

Don Jones

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.

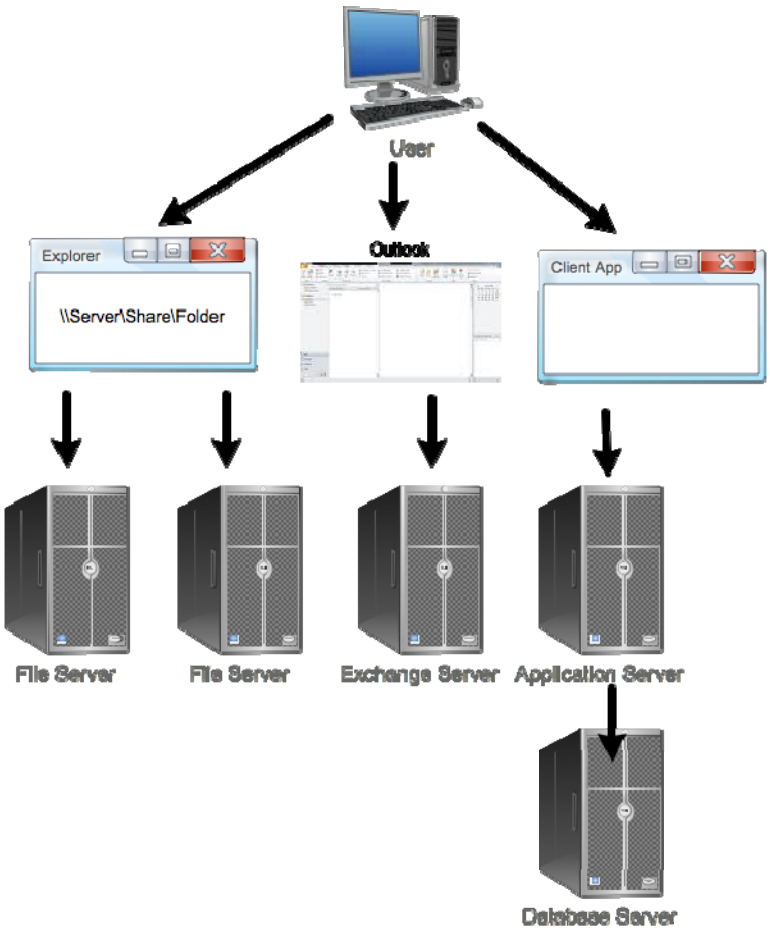

Figure 1.1: Users access information from too many places.

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.

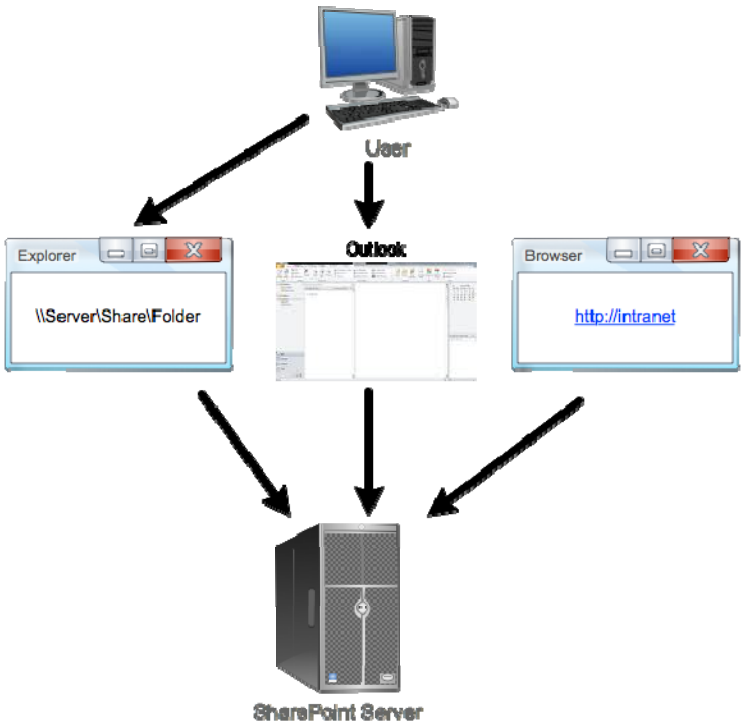

Figure 1.2: Centralizing everything in SharePoint.

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.

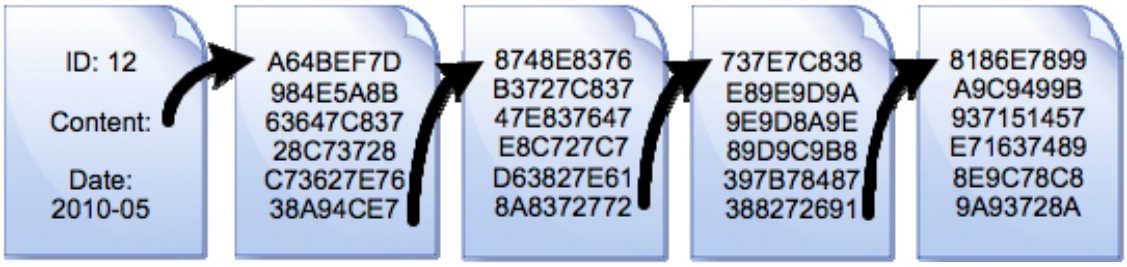

Figure 1.3: Storing data on pages in SQL Server.

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.

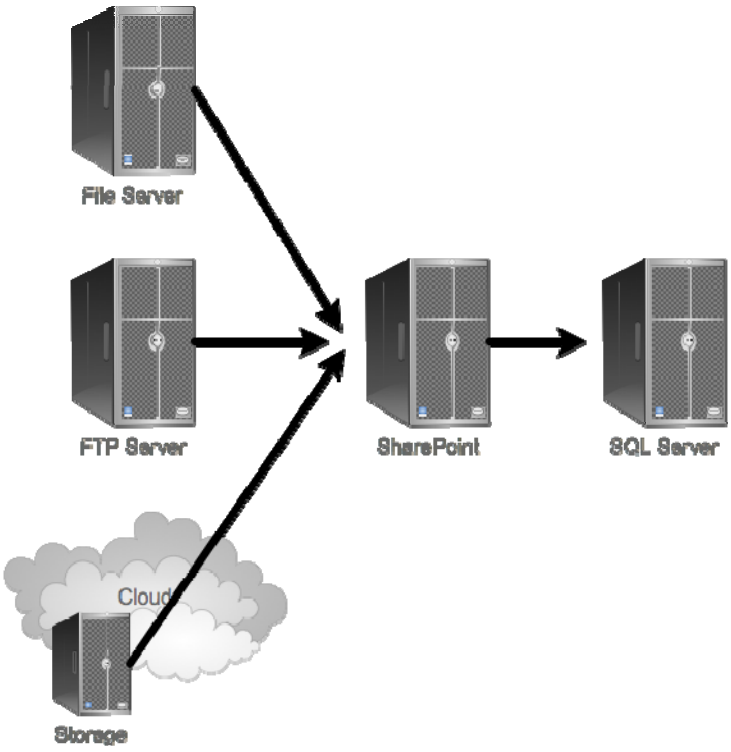

Figure 1.4: Migrating content into SharePoint.

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.

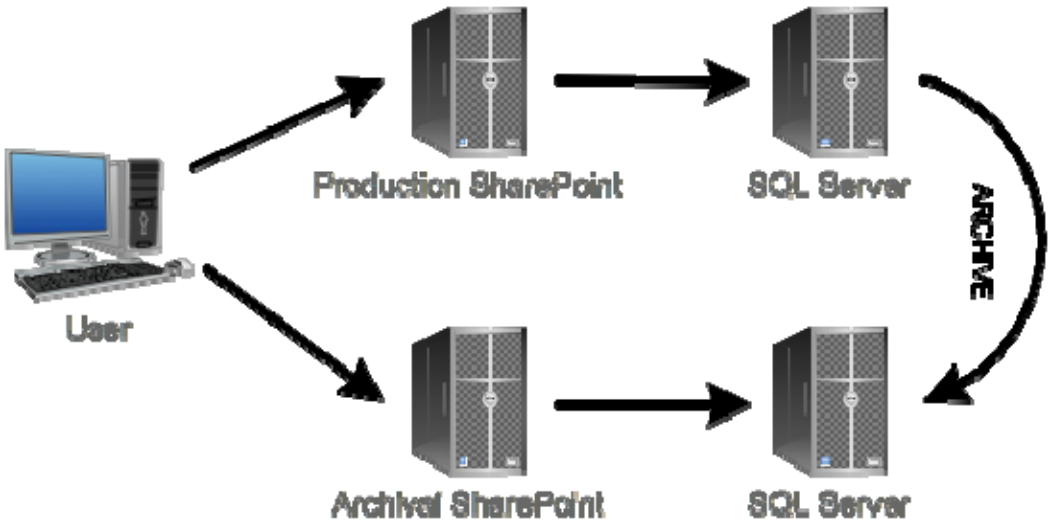

Figure 1.5: Tiered SharePoint storage.

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.

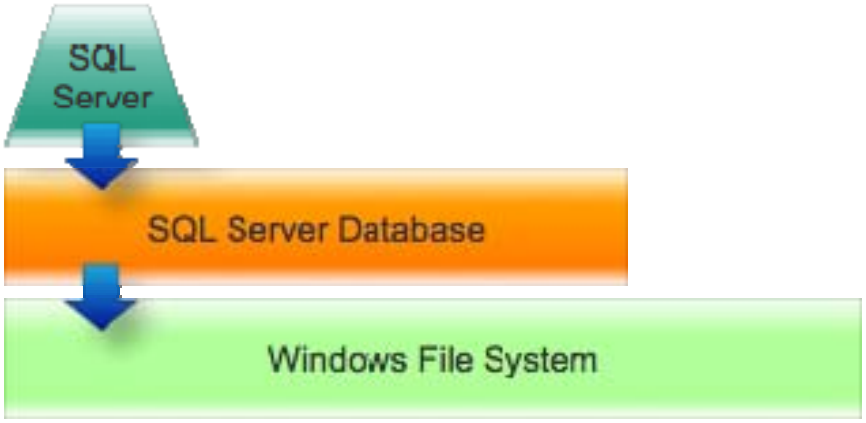

Figure 1.6: High-level SQL Server storage.

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.

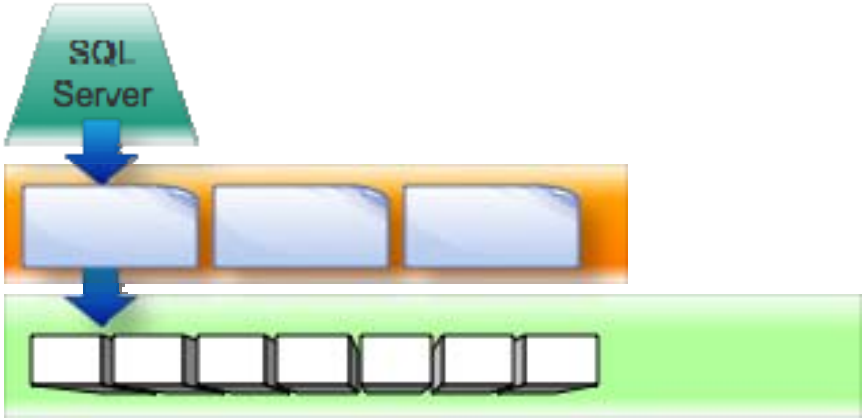

Figure 1.7: Storing data in pages and disk blocks.

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.

Chapter 2: Optimizing SharePoint Storage for Large Content Items

One of the biggest uses of SharePoint is to store *large content items*. Unfortunately, those are also one of the biggest contributors to massively-larger SQL Server databases, slower database performance, and other problems. One of the most important topics in today's SharePoint world is optimizing SharePoint to store these large content items.

What Is "Large Content?"

Large content, in this context, refers primarily to the file attachments stored within SharePoint. Microsoft refers to this kind of content as *unstructured data,* as opposed to the more structured, relational data normally stored in a database.

As outlined in the previous chapter, SQL Server's default means of storing this kind of data is as a Binary Large Object (BLOB), usually stored in a column defined with the varbinary() type. Physically, SQL Server keeps a pointer on the actual data page, and spreads the BLOB data across several pages. Figure 2.1 illustrates how the row data page provides a pointer to sequential BLOB pages.

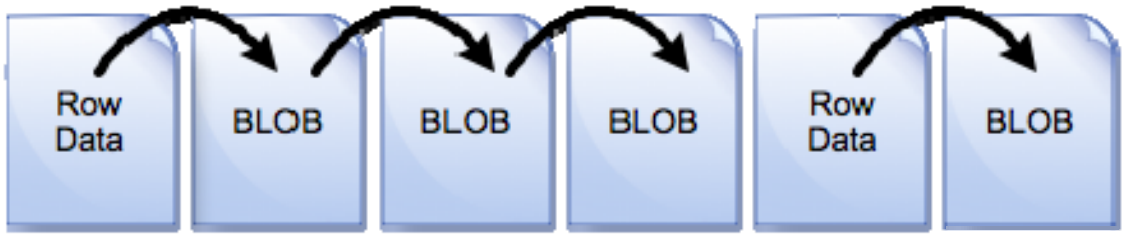

Figure 2.1: BLOB storage in a SQL Server database.

SharePoint can, of course, accommodate a *lot* of large content items. For example, if you enable versioning, then every new version of a file will be a new large content item—and the previous versions will remain in place. A typical file attachment might have less than 10KB of structured data associated with it—so much of your SharePoint database will be occupied by these BLOBs. For example, if your average Word document is half a megabyte, then you can expect about 95% of your database to be occupied by BLOBs, with just 5% being the actual SharePoint data used to track and provide access to those BLOBs. I examined a couple of SharePoint databases from consulting clients and found that number to hold roughly true for all of them. Figure 2.2 illustrates the percentage—it's pretty impactful to see a visual like this and realize that *most* of your database space is given over to BLOB data—essentially making SQL Server into a file server.

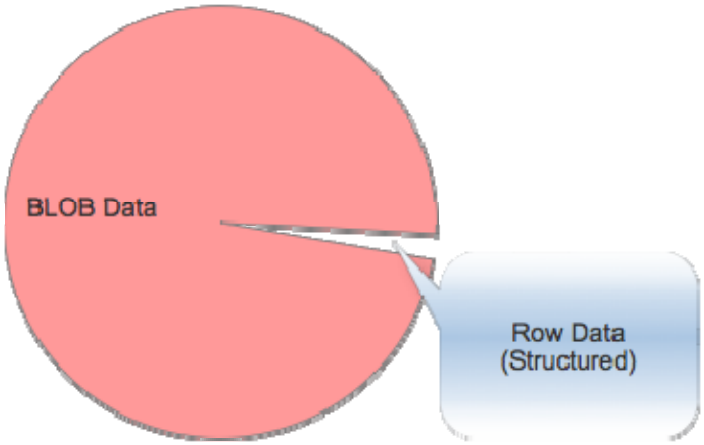

Figure 2.2: Around 95% of many SharePoint databases is BLOB data.

Of course this won't always be the case—I have clients who have SharePoint sites that don't contain any file attachments. Of course, those sites' databases are markedly smaller than the sites that *do* contain a lot of file attachments.

Pros and Cons of Large Content in SharePoint

Obviously, the SharePoint team didn't decide to load up SQL Server with BLOBs just for fun. There are excellent reasons to have that data there—just as there are some significant negative impacts. Understanding the pros and cons, however, is the key to finding a solution that lets us get what we want, with as few of the negatives as possible.

Everything in One Place

The main benefit of having BLOBs in the SQL Server database is that SharePoint can access the file data very easily. This is important for things like its workflow features, alerts, security model, and most importantly for its ability to index the file contents for search purposes. If SharePoint were to just dump all the data on a file server someplace, all of those things would be a bit harder to manage, and might well require additional layers and services to provide things like content indexing.

Having everything in the database also makes backup and recovery fairly straightforward: Just back up the SQL Server database and you're done. SQL Server features like database mirroring, replication, log shipping, compression, and transparent encryption all come for free when everything's in the database, giving you a lot of flexibility in how you work with your data, protect it, and so forth.

Keeping everything in the database helps with consistency as well. All the data is in one place, so it's always internally consistent. Delete a file entry in SharePoint and the file data—the BLOB—goes away immediately. If SharePoint kept the file in the normal file system, SharePoint would have to coordinate changes and deletions between the file system and the database, creating an opportunity for inconsistency that you and your users would probably not appreciate.

Negative Database Impact

The downside of having all of that BLOB data in the database is that, of course, it takes up a lot of space. That can actually have some subtle impact on SQL Server performance. For example, consider the data pages shown in Figure 2.3. Here, you can see that actual data rows are interspersed by BLOB data. That makes it harder for SQL Server to read the actual structured data because SQL Server has to skip over BLOB data pages in order to do so. This isn't a huge deal when you're talking about a page or two, but when reading a large number of rows in a heavily-populated database, it has a cumulative negative impact on performance.

Another issue is database fragmentation. In Figure 2.3, you can see that one of the large items has been deleted, leaving a big empty space in the database. SQL Server may not immediately re-use that space, so you'll wind up with wasted disk space—a database file that's physically larger than it needs to be. That results in the need for more frequent maintenance.

Figure 2.3: Deleting BLOB data can result in a lot of wasted space.

Worse, when SQL Server does start re-using that space, it'll lead directly to data fragmentation. As Figure 2.4 shows, SQL Server is now storing data pages out of order, meaning it'll have to hop back and forth within the database files to retrieve data rows. Pretty ugly, right? Again, not a huge deal on a piece-by-piece basis, but it does have a cumulative negative performance impact in a large, busy database.

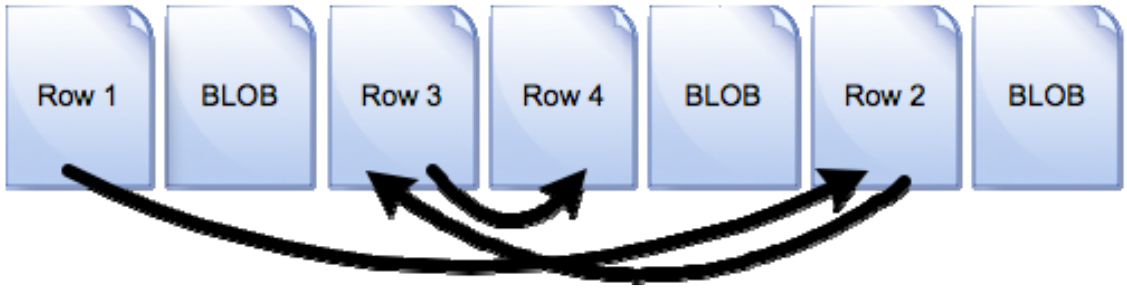

Figure 2.4: Fragmented databases require more reading effort.

Goals for Large Content in SharePoint

Optimizing SharePoint storage is, in many cases, led by an effort to get the large content items out of SharePoint. When we do that, however, we need to do so with an eye toward retaining all of SharePoint's features. We don't want to just shrink the database for the sake of doing so if at the same time we're losing access to features like security, workflow, alerts, indexing, and so on.

Remove the Data from the Database

So the primary goal is to remove *just the BLOB* data from the database. That leaves SQL Server with a cleaner, leaner database consisting entirely of data rows. Problems like empty space and fragmentation can still occur, of course, but they'll be much less severe, meaning you'll have to perform maintenance on the database less frequently.

Backups will *not* necessarily be faster, as you'll still want to back up the BLOB data, wherever it winds up living. In fact, one risk is that you'll have to come up with new backup routines, depending on how you do offload the BLOBs because your offloading technique won't necessarily integrate with SQL Server's native backup scheme.

> **Third-Party Backup Tools**
>
> Some third-party backup tools rely entirely on SQL Server to deliver the data to back up. If your offloading scheme takes the BLOB data away from SQL Server, then SQL Server won't know about it—and won't be able to deliver it to the backup tool.

Keep the Metadata in the Database

We don't want to pull *all* of the content out of the database, though. The *metadata* needs to stay—that's information like who owns the file, who has permissions to the file, what keywords are associated with the file, and so forth. SharePoint needs that information to manage the file itself, and having that structured data in the database is the best way for SharePoint to use it. The metadata, as I wrote, doesn't take up much space, and it's really what the database is *for.*

Keep the Content Searchable

We also want to keep the content searchable, meaning SharePoint has to be able to access the BLOB contents in some fashion. As you'll see in the upcoming sections, that means either SharePoint needs to be able to find the file on its own, or SharePoint needs to be able to transparently access the BLOB data through SQL Server—even if SQL Server isn't physically storing the BLOB data. Figure 2.5 shows the three ways this can work: storing the data in SQL Server itself (the middle path), using a SQL Server-side BLOB offloading mechanism (shown on the left), or using some SharePoint-based BLOB redirection (on the right).

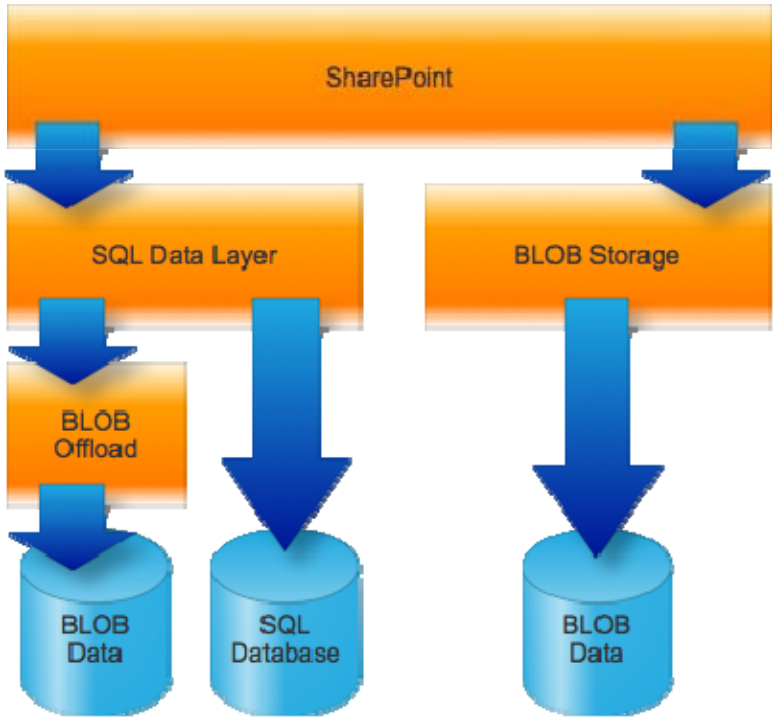

Figure 2.5 Keeping BLOB content searchable.

Note

As a convention in this chapter, green arrows will represent the flow of structured data, while blue represents the flow of unstructured BLOB data. A green/blue arrow will represent either type of data.

Any of these techniques will ensure that SharePoint still has access to the BLOB data so that SharePoint can crawl that data for search purposes.

Keep the Content Secured

We also want the content to remain secured using SharePoint's security system. That system is complex enough without layering some other security mechanism on top of it, so the BLOB data needs to be stored in a way that prevents access to it except through SharePoint, or in some other way that respects SharePoint's priority in controlling access to the file. What we want to *avoid* is a situation like that shown in Figure 2.6 where the BLOB data lives elsewhere, such as on a shared folder, and can be accessed directly—effectively bypassing SharePoint.

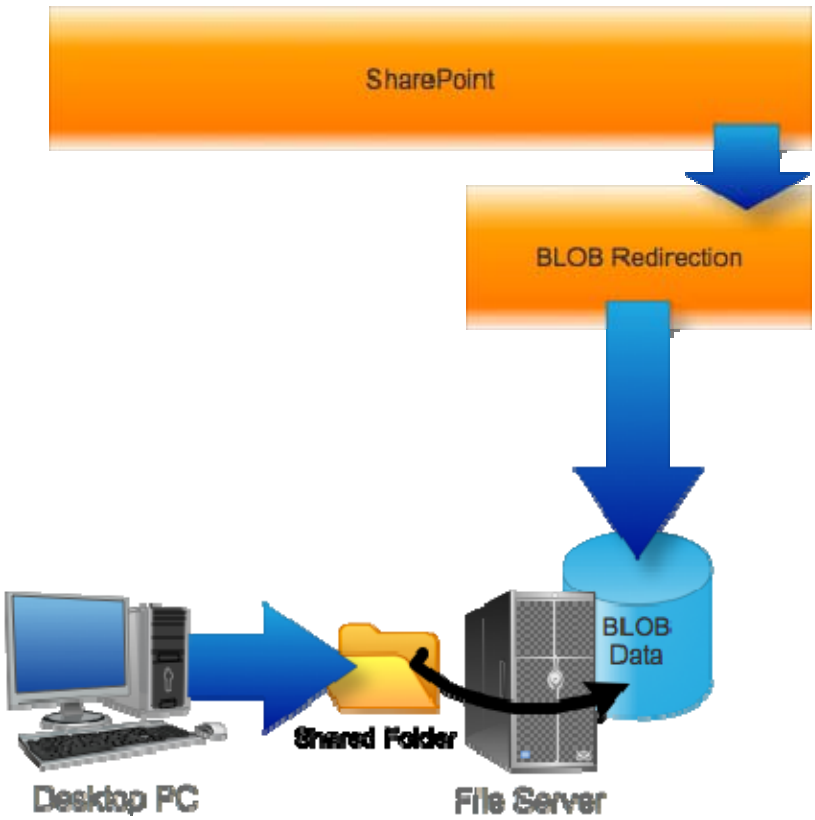

Figure 2.6: You don't want to be able to bypass SharePoint's security.

Simply storing BLOB data on a file server is not inherently a bad thing; what matters is how the data is secured on that file server. You simply want to ensure that the data can't be accessed without SharePoint. Or, if it *can* be accessed without SharePoint, that any such access can be synchronized back to SharePoint—so that SharePoint remains nominally in control of that data.

Keep the Content Versioned, Alerted, Workflowed, Etc.

Finally, we need to ensure that all of SharePoint's other features—versioning, alerting, workflow, and so on—continue to operate on the data that we've offloaded. These features are the main reason for using SharePoint, after all, so if we lose these features, we've sort of made SharePoint useless. If we can't offload the data *and* retain these features, we probably wouldn't want to offload the data at all.

Extra Features

We might want to add features on top of what SharePoint and SQL Server currently offer. For example, we might want to direct different categories of BLOB data to different storage locations—high-risk data, for example, might live on redundant storage, while low-risk data (like the week's cafeteria menu) might live on less-expensive storage. Not every company will want or need that option, but if you do need it, it's something to keep in mind as you explore your options.

We might also want to implement some kind of tiered storage, so that older, less-used data can be moved to less-expensive storage but still remain generally accessible. We might need to observe data-retention policies, too, and it's possible an "outsourced" storage mechanism can address that. This is actually a bigger topic than just offloading BLOB data, and I'll spend all of Chapter 4 exploring it.

The Solution: Move the BLOBs

Ultimately, what we want to do is get the BLOBs out of the SQL Server database. That'll reduce the size of the database, which will help improve its performance. Of course, we need to do so in a way that still keeps the content entirely visible to SharePoint so that all of its features—alerts, workflow, versioning, metadata, security, and so forth—still work for us. There are two basic approaches to BLOB offloading: External BLOB Storage (EBS) and Remote BLOB Storage (RBS).

EBS

EBS was introduced with the previous version of SharePoint. It's a SharePoint-specific feature; it's not generic to SQL Server. Essentially, EBS was the SharePoint product team's way of addressing the BLOB problem without specific help from the SQL Server team.

By the way, you should know that EBS is deprecated in SharePoint 2010, meaning Microsoft doesn't plan to pursue the technology. Instead, they're pushing for RBS, which I'll discuss next. However, it's still useful to understand what EBS is and how it works.

How It Works

EBS basically provides a secondary storage mechanism within SharePoint: You still use SQL Server to store structured data, but those unstructured large items are directed to a different storage mechanism. A Component Object Model (COM) interface coordinates the two. EBS is an extensibility point; both Microsoft and third parties can supply you with EBS providers, and the provider makes the connection between SharePoint's EBS layer (that COM interface) and the actual external storage mechanism. Figure 2.7 illustrates how EBS is built.

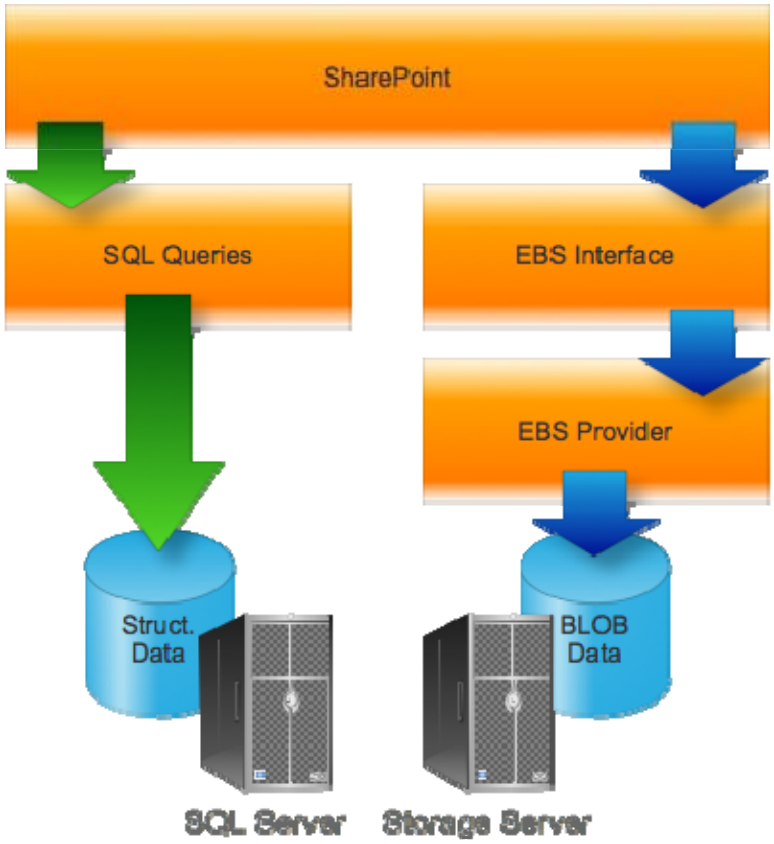

Figure 2.7: SharePoint EBS.

As you can see, structured data is still stored in SQL Server in the normal fashion; BLOB data is redirected at the SharePoint source through the EBS interface.

Pros

Obviously, the "pro" is that EBS gets BLOB content out of the SQL Server database, reducing its size and helping to improve its performance. EBS can work with any version of SQL Server supported by SharePoint, so you don't need to be on the latest versions of everything in order to use it.

Cons

EBS isn't the most sophisticated technology. For example, it doesn't automatically overwrite old BLOBs. If you update a file attachment, EBS creates a *new* store item, and redirects all of the structured SharePoint data—metadata and the like—toward the new item. Whoever wrote the EBS provider is responsible for cleaning up the now-orphaned "old" data item. In practice, Microsoft says most EBS providers will put that off until some scheduled "garbage collection" time, when orphaned BLOB data will be cleaned up.

EBS doesn't integrate with SQL Server directly, meaning SQL Server's internal backup/recovery mechanisms, log shipping, replication, and so forth are unaware that BLOB offloading is happening. That adds a layer of complexity to these operations.

EBS was always intended as a kind of stopgap solution; even when introducing it, the SharePoint team knew that something better would be on the way—and they did not design EBS to migrate to that better way (which turns out to be RBS, which I'll cover next). That said, some third parties *can* help migrate from EBS to RBS, and many third-party EBS providers have RBS equivalents, so you can often stay with the same storage vendor.

If you add EBS to an existing SharePoint installation, it won't convert existing BLOBs over to the new EBS storage; you'll have to do that manually—for example, by creating a new SharePoint site that uses EBS, then restoring your existing SharePoint data to that new site. Not exactly painless; some third parties may offer tools to help automate the process and make it less painful.

RBS

This is a technology implemented entirely by SQL Server (2008 and later); SharePoint has no clue that it's there or working, nor would any other application. You don't configure SharePoint at all—you simply enable this in SQL Server. Applications *can* be made RBS-aware, though (SharePoint is one such application), and those applications can make even more efficient use of RBS.

How It Works

RBS is designed as an extensible mechanism. To use it, you need an *RBS provider*, which is what actually handles dealing with the BLOBs on behalf of SQL Server. Microsoft ships a default provider, the FILESTREAM provider, that utilizes the new FILESTREAM data type in SQL Server's 2008 R2 Feature Pack.

Essentially, the FILESTREAM data type is something you assign to a column in a table. That is, instead of declaring the column as a varbinary() type, you add the FILESTREAM attribute to the varbinary() column. SQL Server then automatically writes the BLOB data to the file system rather than into the database. You still use SQL Server to add and retrieve BLOB data; all that's changed is where SQL Server physically stores it. Using a simple FILESTREAM column doesn't change the way you do backup and recovery, in fact; SQL Server "knows" about FILESTREAM columns and integrates them into the backup processes. They even work within SQL Server's security model and are fully supported by transactions. Figure 2.8 shows how it works: Essentially, the FILESTREAM type tells SQL Server to split out the BLOB data into normal files on the file system.

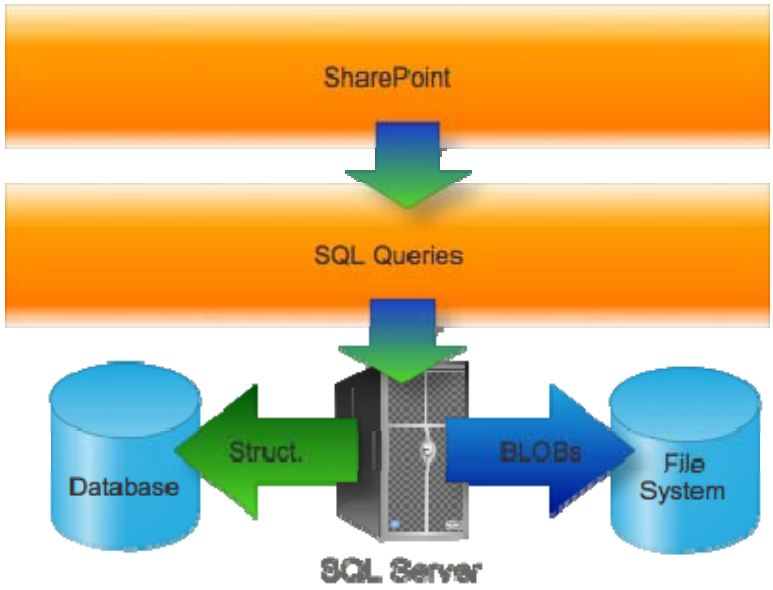

Figure 2.8: How the FILESTREAM type works.

Developers can *choose* to use a different technique to access BLOB data, which involves using Win32 streaming application programming interfaces (APIs), essentially meaning they can access the externally-stored files directly through a shared folder—taking some of the burden off SQL Server and instead using the Windows file system—which is really, really good at handling files. That does require a programming change in the application, but it can improve performance pretty significantly. Figure 2.9 shows how this works: The application, in this case, needs to do a little bit of extra work because it needs to access two data stores in order to deal with BLOBs. It isn't a massive undertaking from a programming perspective, but it isn't as transparent as just letting SQL Server do the work. However, SQL Server isn't a file system, so the extra programming work is generally rewarded by improved application performance. SharePoint doesn't necessarily use this approach; instead, you'll typically see it using RBS—which itself can use FILESTREAM.

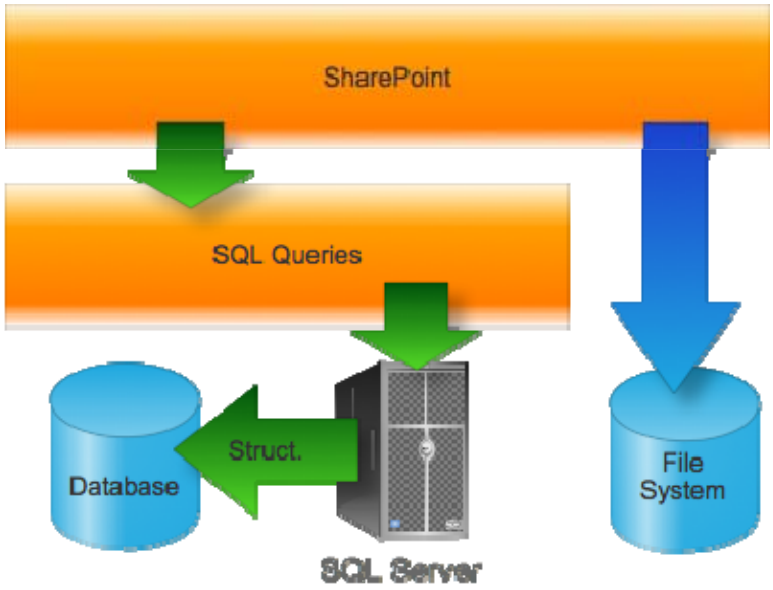

Figure 2.9: FILESTREAM-aware applications.

The trick with the default FILESTREAM implementation is that it can only store data on the local disks available to the SQL Server computer. Some SQL Server features—such as transparent encryption—won't work for FILESTREAM data. Tables containing FILESTREAM data can't be used in database snapshots or in database mirroring (although log shipping is supported).

As I said, SharePoint doesn't necessarily use FILESTREAM types directly. The next level up is the RBS API, which is what SharePoint 2010 *does* normally use. RBS recognizes that some companies have developed BLOB-specific storage systems, and RBS provides a way to offload BLOB data to those. RBS retains full awareness of the BLOB. That is, if you delete a row, SQL Server will delete the corresponding BLOB data even though it's stored elsewhere. RBS also doesn't provide quite the same level of data consistency that you get when storing BLOBs directly in the database or by using the normal FILESTREAM type. Figure 2.10 illustrates one way in which this can work. Note that in this example, SharePoint has been modified (there's a downloadable RBS component for it) to explicitly use RBS.

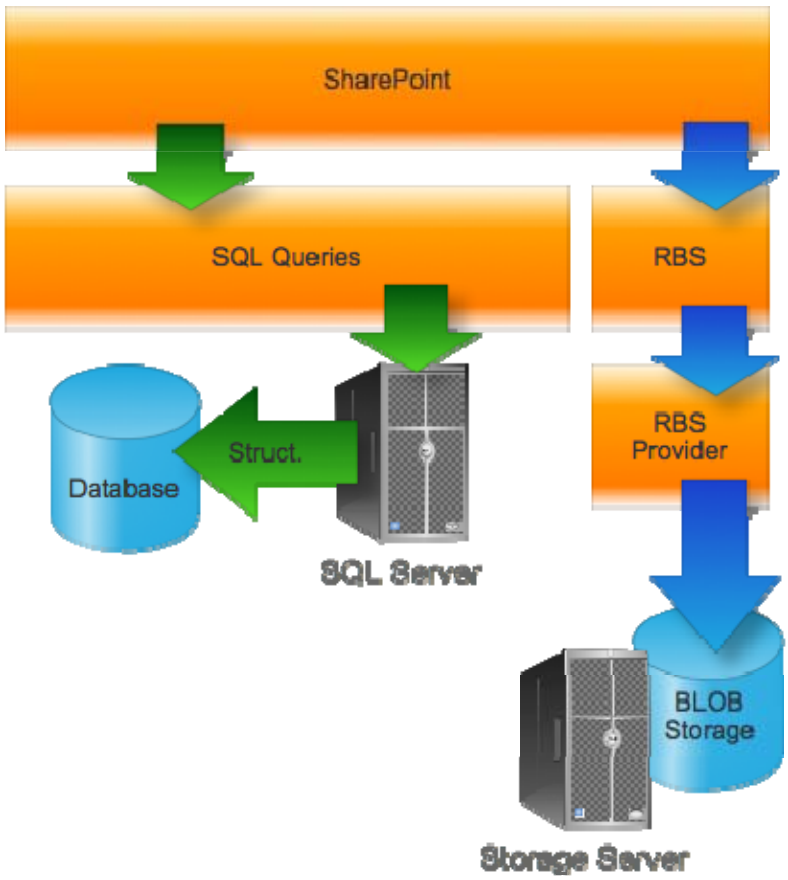

Figure 2.10: SharePoint and RBS.

Figure 2.10 is a bit of a simplification; the actual stack of components is a bit more complicated, but this illustrates the general idea. For completeness' sake, I should also note that RBS doesn't always require that an application be aware of it. It's something that can operate solely within SQL Server, as shown in Figure 2.11.

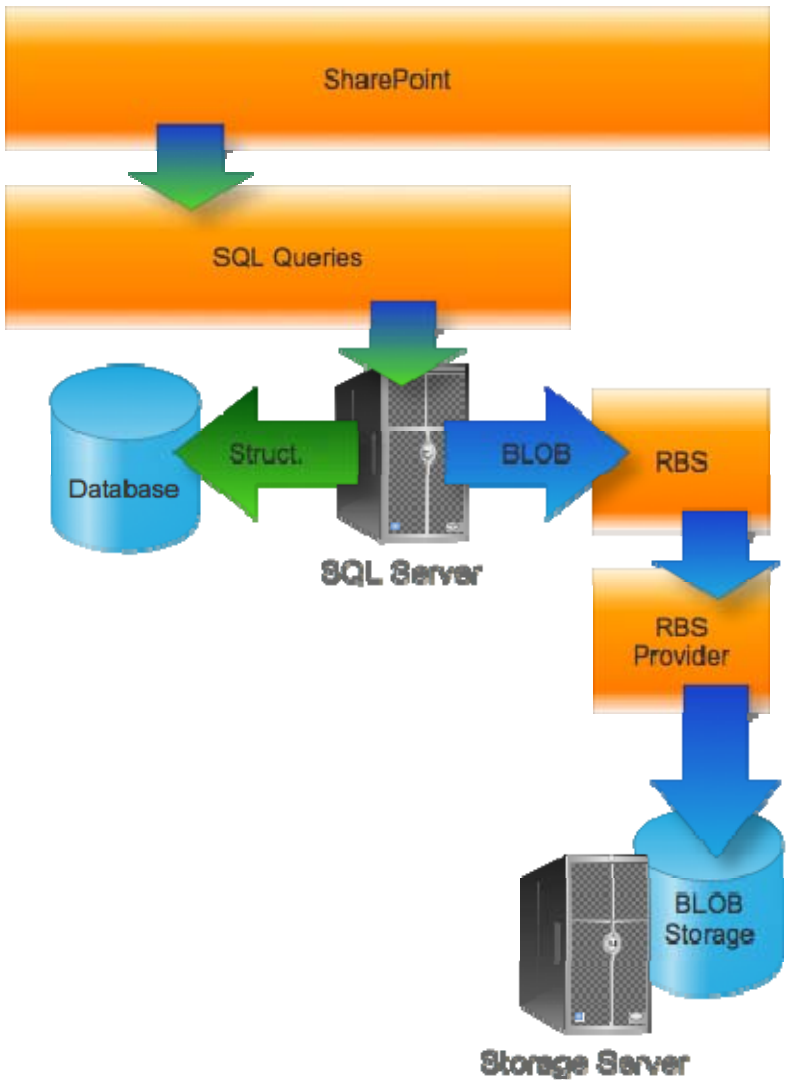

Figure 2.11: Transparent RBS.

Microsoft's general intention is that providers of BLOB stores—that is, vendors—will write RBS-compatible providers that tell SQL Server how to talk to that vendor's BLOB store. Microsoft offers an RBS provider that simply utilizes the FILESTREAM type, so you can get a basic RBS setup running out of the box (with SQL Server 2008 R2, at least). Essentially, what you do is create a *new* database in SQL Server that will be your "BLOB Store." You configure RBS on your SharePoint database to offload BLOBs to that BLOB Store; the BLOB Store in turn is set up to use the FILESTREAM type to push the BLOB data to disk. So the BLOB store winds up being nothing more than pointers to the actual BLOB data; the SharePoint database, in turn, contains pointers to the BLOB Store. Figure 2.12 shows how this all fits together.

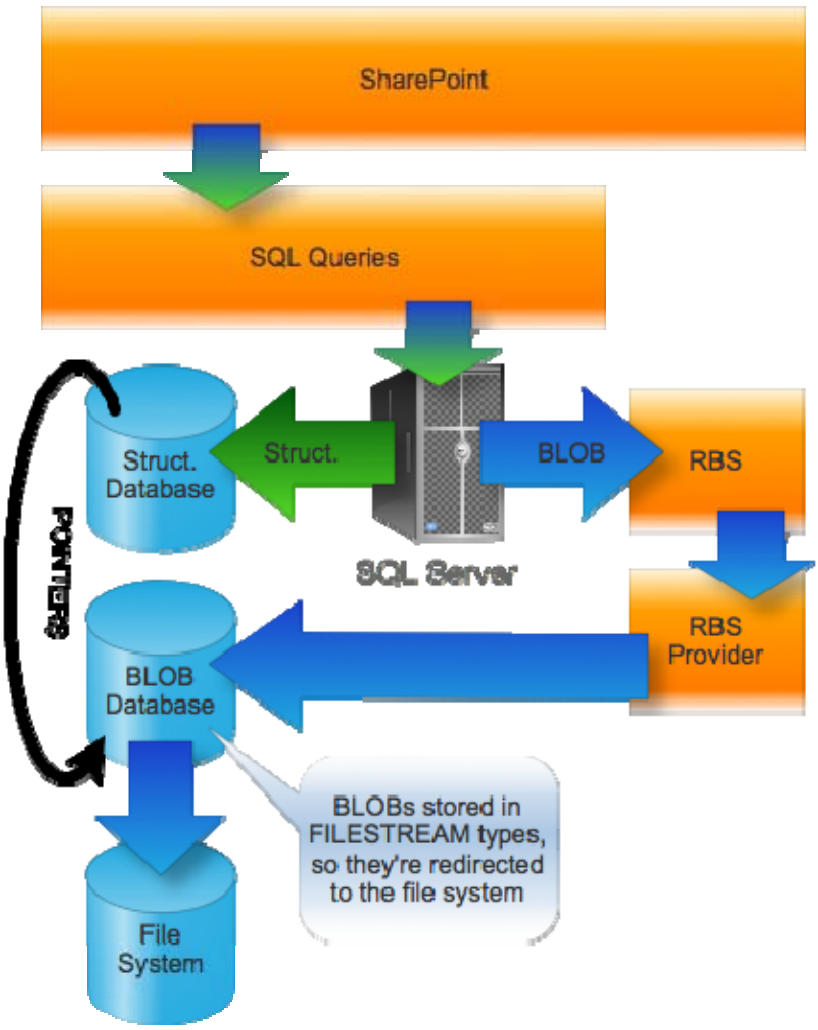

Figure 2.12: Microsoft's FILESTREAM RBS provider.

Again, all of this happens locally on the SQL Server—the files must be on the same physical machine as the BLOB database when using Microsoft's FILESTREAM RBS provider.

Pros

Obviously, the big advantage here is reducing your SQL Server database size—by as much as 90 to 95% in some instances I've seen. That's huge for maintaining SQL Server's efficiency as well as giving you a bit more flexibility in the backup and recovery department. Using strictly the Microsoft-offered FILESTREAM provider, however, does limit your flexibility: You're still stuck with only local storage, and there are complex interactions with other SQL Server features.

RBS is definitely the "way forward" for both SQL Server and SharePoint. Using RBS will provide a longer future for you than EBS will.

Cons

RBS obviously creates a somewhat more complex environment. Although normal SQL Server backups can work transparently, third-party backups may or may not work with RBS, depending on their exact approach—so it's something to investigate.

Not every RBS approach is created equal. You explicitly need to ensure that offloaded data can still contain metadata, still be indexed for searching, still be managed by SharePoint and its security rules, and so forth. You might want to offload different types of data to different locations, and the RBS provider would need to offer that kind of filtering functionality; if you just want to offload *everything*, you can likely use a simpler RBS provider.

Finally, RBS does require the latest versions of SQL Server and SharePoint. Thus, if you're stuck using older versions with no chance of upgrading, this might not be an option for you.

Third-Party Approaches

Many third parties are now offering their own RBS providers, as RBS is the officially-supported BLOB offloading mechanism. SharePoint only needs to understand that "RBS is in use;" it doesn't care what's actually handling the BLOBs in the background.

How It Works

Third parties can either write an RBS provider that is installed on SQL Server or even tap into the EBS architecture used by older versions of SharePoint. If you're in a mixed-version environment, an extender that can use either RBS or EBS might be desirable.

Added Flexibility

Third-party RBS providers can provide much faster BLOB offloading and can take more load off SQL Server. Keep in mind that the RBS FILESTREAM provider still places a load on SQL Server because SQL Server has to process the BLOBs into and off of the file system. Third-party providers can also offer other features:

- Offload to remote storage (such as a network-attached store or even to cloud storage)
- Cache BLOB data for transmission to off-premises storage (such as a cloud-based backup)
- Spread BLOB data across multiple storage locations, potentially storing different types of data in different places—Figure 2.13 shows one way in which that might work, with files of different categories (perhaps defined by SharePoint metadata) are offloaded to different storage mechanisms

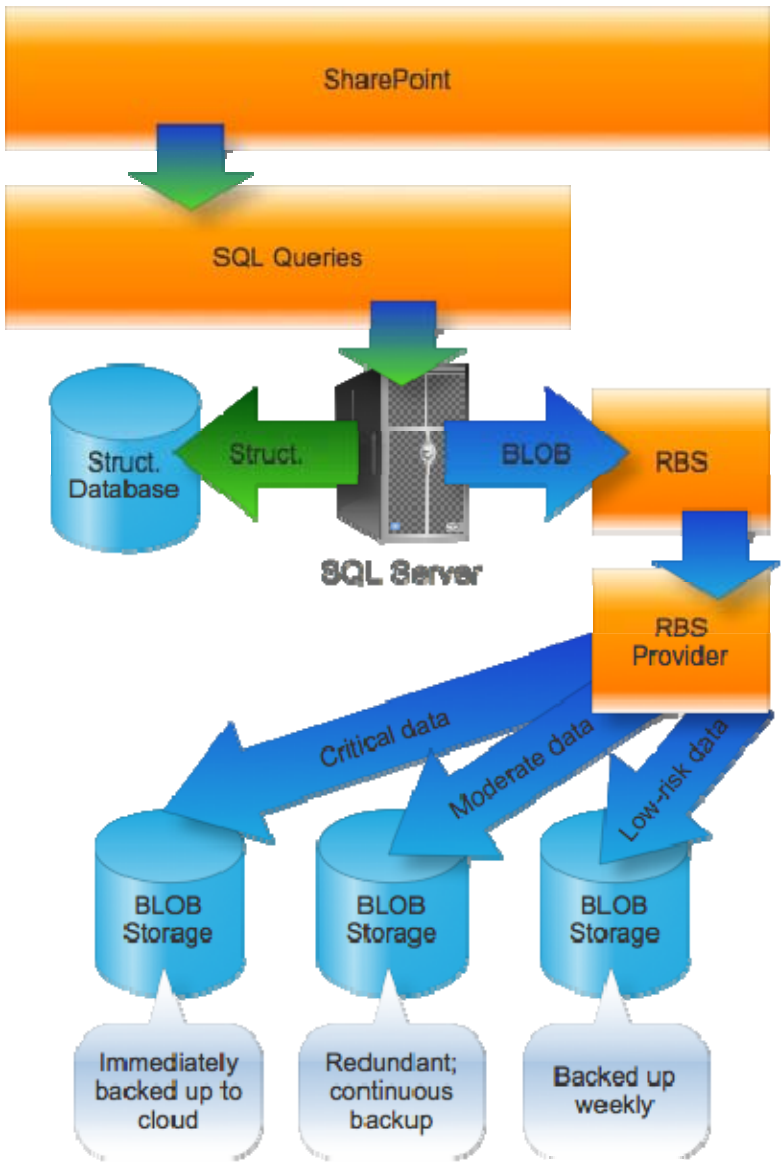

Figure 2.13: Offloading data to different locations.

- Add secure deletion (erasing) to the BLOB deletion process to help comply with security requirements
- Compress and/or encrypt BLOB data—something that the native FILESTREAM provider cannot do
- Work with hierarchical storage mechanisms for tiered storage, enabling you to migrate older content (for example) into near-line storage; note that some vendors may implement this kind of functionality as a discrete product that works in conjunction with a separate RBS provider, while others may build the functionality into an RBS provider
- Transparently ensure that SharePoint can access BLOB data for search indexing

Note

A third-party RBS (or EBS) extension doesn't even have to be expensive—some companies offer these providers for free and intend for you to use them with the storage resources you already have or plan to implement. Try a Web search for "free sql rbs extender" and see what comes up.

Coming Up Next

So that's the large item storage taken care of. In the next chapter, we need to look at another way in which large items come into play: legacy, or external, content. In the perfect world, you'd consolidate all of your data-sharing activities into SharePoint, including all those old-school file servers that you have laying around. Unfortunately, you'd be exponentially loading your SharePoint servers, which might not be the best idea. Is there a way to get all of your shared files into one place—like SharePoint—while still maintaining great performance and smart utilization of your storage resources? We'll see—coming up next.

Chapter 3: Optimizing SharePoint Storage for External or Legacy Content

Shared folders. External databases. Even media files—audio, video, and so on. We want it all in SharePoint so that it's version-controlled, secured, and searchable—but can we afford to bloat the SharePoint database with that much content? Adding all that content will not only result in a pretty sizable SharePoint database but also take up a lot of expensive SharePoint storage. However, with the right tools and techniques, you can bring that content "into" SharePoint, while keeping it stored "outside," helping to optimize your SharePoint storage and maintain a smaller, more manageable SharePoint content database.

What Is "External Content?"

Today's businesses not only have a ton of data but they have it spread all over the place. Ideally, we could find a way to get *all* our content into *one* place, where users could search for it, read it, and potentially even submit changes as appropriate and allowed. SharePoint can't quite get you to the point where *all* your data is in once place—but with the right tools and techniques, it can come remarkably close to that goal.

Databases

Businesses keep more data than ever in databases. In fact, it seems as if every major corporate application written these days requires a database of some kind. We don't necessarily want people to be able to *use* that data from within SharePoint, but a lot of the information living in databases *can* be used to great effect in SharePoint-based dashboards and other visualizations. For example, why not display charts and goals right on SharePoint home pages, keeping employees connected with the company's top-level goals and letting them know how well they and the rest of the company are doing in achieving those goals?

Of course, you certainly wouldn't want to copy all that business data into SharePoint just to create charts and graphs, or to use the data in other ways. Ideally, you want SharePoint to be able to *access* the data where it sits—in a secure fashion—so that SharePoint doesn't become a means for users to gain access to data that they shouldn't be looking at.

Files in Shared Folders

The first corporate local area networks (LANs) were used primarily to connect users to each other—and to use each others' machines as file servers. The first server applications were primarily focused on sharing files. In fact, the first collaboration software—such as Microsoft Mail and cc:Mail—were essentially just elaborate file-sharing services. Shared files and folders have, in other words, been a key means of sharing information on the network since the very beginning of modern networks.

As the amount of data we store continued to grow, however, shared folders became less and less efficient. We had to map an increasing number of network drive letters to increasingly-complex shared folder hierarchies in order for our users to find all of that data. And keeping it organized was (and is) often a nightmare.

Shared folders don't typically support versioning, workflow, and all the other great features present in SharePoint. In fact, it would be great if we could just migrate all of that shared folder content into SharePoint. Doing so, however, isn't always feasible. The sheer amount of data may make a complete migration away from file servers impractical; in some cases, other processes depend on files being located on file servers, not within SharePoint.

In other cases, migrating shared folders into SharePoint can seem, well, wasteful. After all, they're just files, right? Why put them into a database, which is stored on a file system, when we could just store the files on the file system directly? Of course, but *not* having that content in SharePoint, we don't have access to features like versioning, workflow, advanced permissions, and so on—so there definitely *are* benefits to having the content "in SharePoint" somehow. However, perhaps a straightforward migration isn't the best means of achieving that goal.

Media Files

Media files, more than other kinds of shared files, can highlight the worst about file servers *and* be the most difficult to bring into SharePoint. Media files—audio and video files, primarily—are difficult to index when they're just living on a file server. After all, a search engine can't (yet) "listen" to an audio file and extract keywords that can be used in searching. The file system doesn't, unfortunately, provide us with many ways to manually add those keywords as metadata, either. File servers make it tough for users to find the media files they want and need.

SharePoint could improve the situation with its more-robust metadata capabilities and search features, but moving media files—which tend to be pretty large—into SharePoint can exponentially increase storage utilization and even reduce database performance. Moving media into SharePoint can even reduce the performance of the media itself: Most media files are meant to stream, something that can be done quite readily from the file system. Streaming data from a database, however, goes pretty much against everything that our databases are designed to do. It can be burdensome for a database to retrieve streaming content properly, and when the database can't keep up, media playback is affected.

Again, we're faced with a bit of a dilemma: It would be nice to have media in SharePoint for versioning, workflow, indexing, and searching, but it seems as if doing so would negatively impact storage utilization and might even result in poorer user experiences.

Files from the Cloud

Recently, businesses have started storing more and more data in a variety of cloud-based repositories. Those repositories take the form of simple FTP servers to cloud-based databases, and from Web sites to dedicated cloud-based storage. We're basically distributing our old file server model across the Internet—and we're not doing much to make it easier to find and manage that data.

But can we just pull all of that cloud-based data into SharePoint? Doing so would make it easier to access and manage, but it seems as if that would defeat the whole point of putting the data into the cloud. If we're just going to pull the data back into our SharePoint data center, then we really don't need the cloud at all, right? Again, a dilemma: How can we make that cloud-based data appear to live in SharePoint without moving it out of the cloud?

> **Note**
>
> You're probably thinking, "SharePoint in the cloud!" as a solution. Not quite. Although it's true that many companies offer SharePoint hosting, their SharePoint installations can only access data living in the hosting company's data center—just as *your* SharePoint servers would seem to be able to access only data on *your* network. The ideal would be to have data that could live in one part of the cloud, yet still be "in" SharePoint—either in your own SharePoint servers or perhaps even those you've outsourced to another part of the cloud.

Today: Data Chaos

Frankly, it's amazing that our users can find anything—and it's no surprise that so much disk space is given over to the copies of data that users make into their "home" folders—the only place they can organize data in a way that makes sense to them. We've created "data chaos," illustrated in Figure 3.1, where our users are running all over the place (usually virtually, but not always) looking for files, media, data, and more.

SharePoint seems to offer a solution—a way to organize data that is accessible to everyone who needs it: rich metadata, tagging, and search capabilities, and the ability to version-control information, rollback to prior versions, and even apply workflow rules to information updates. But to really benefit from SharePoint's capabilities, we need *as much of our information as possible* inside SharePoint. Simply migrating everything, however, could have a negative impact on expensive SharePoint storage resources, and on SharePoint's own performance. It's possible that migrating might be the perfect choice for some types of information but less than perfect for others. But what can we do *besides* migrate the information, if our goal is to take advantage of SharePoint's features?

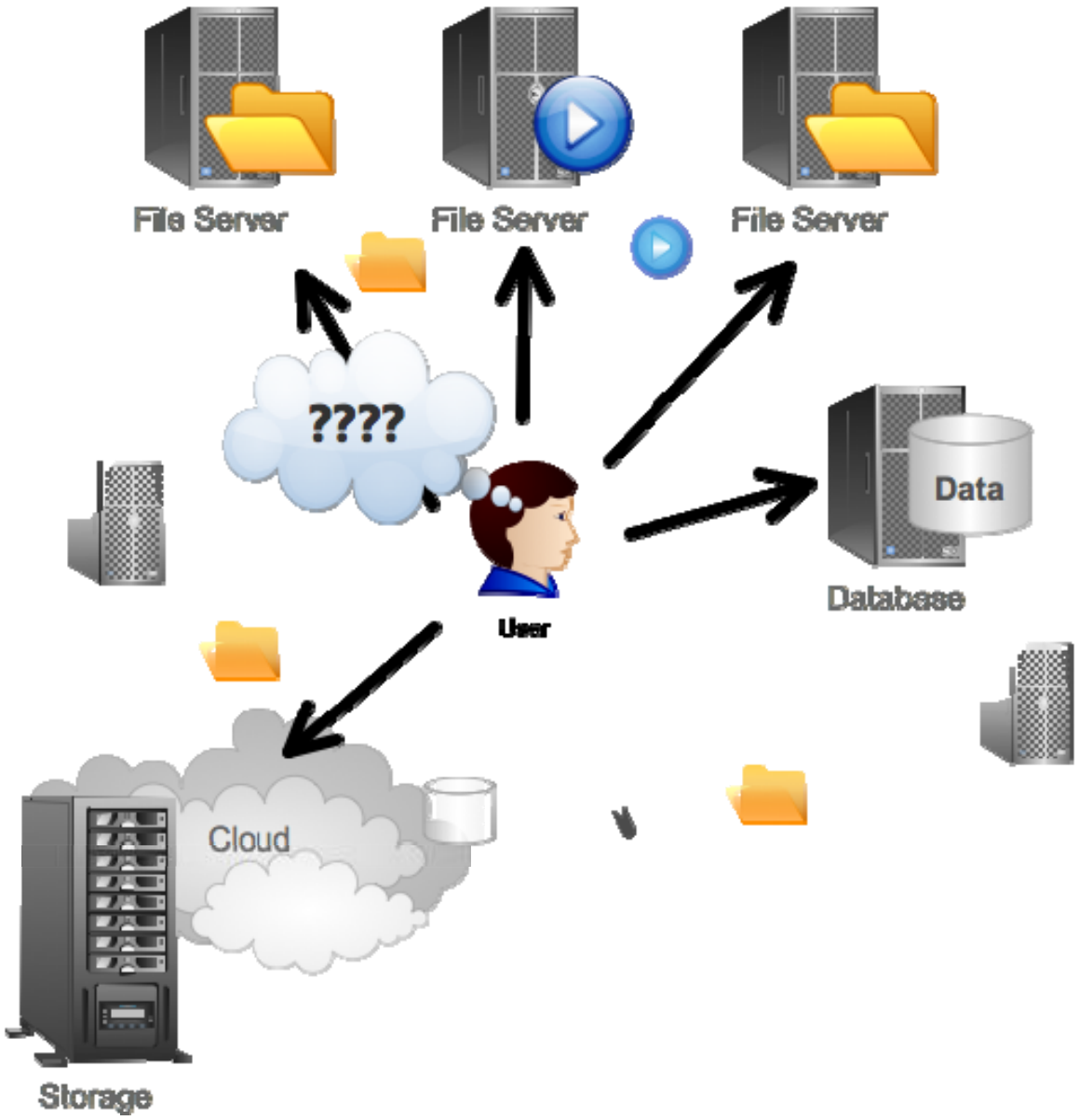

Figure 3.1: Data chaos—our users are forced to look everywhere for the information they need.

What's the answer? There are actually a few approaches, and they differ depending on exactly what kind of content you're working with.

Traditional Approaches for Getting External Content into SharePoint

Let's start by looking at some of the traditional ways of incorporating external content into SharePoint. None of these techniques are bad, but they're not always the *best*. They offer compelling features and they help to meet certain business goals, so it's important that we consider them as potential tactics.

Integration

When you're dealing primarily with data that lives in external databases, you're *not* going to migrate it into SharePoint unless your plan is to create a SharePoint-based application and eventually eliminate that external application (and its database) entirely. For this discussion, I'll assume you have data that you want to *keep* separate from SharePoint, but that you want some means of accessing that data from within SharePoint. Traditionally, Microsoft's answer—introduced with SharePoint Server 2007—has been the Business Data Catalog (BDC).

The BDC

The BDC is a shared service within SharePoint, enabling SharePoint to grab business data from back-end server applications—ideally without any coding on your part (prior to the BDC, integrating back-end data almost always required some form of custom programming, which could be expensive to both create and maintain). The BDC allows data to be integrated into user profiles, custom applications, Web Parts, lists, and even search. It also provides support for displaying data drawn from both databases and from Web services—such as SAP, Siebel, or other line-of-business (LOB) applications.

Figure 3.2 shows how back-end data can be integrated into SharePoint portals. Here, marketing campaign and other information is drawn from a back-end database or Web service.

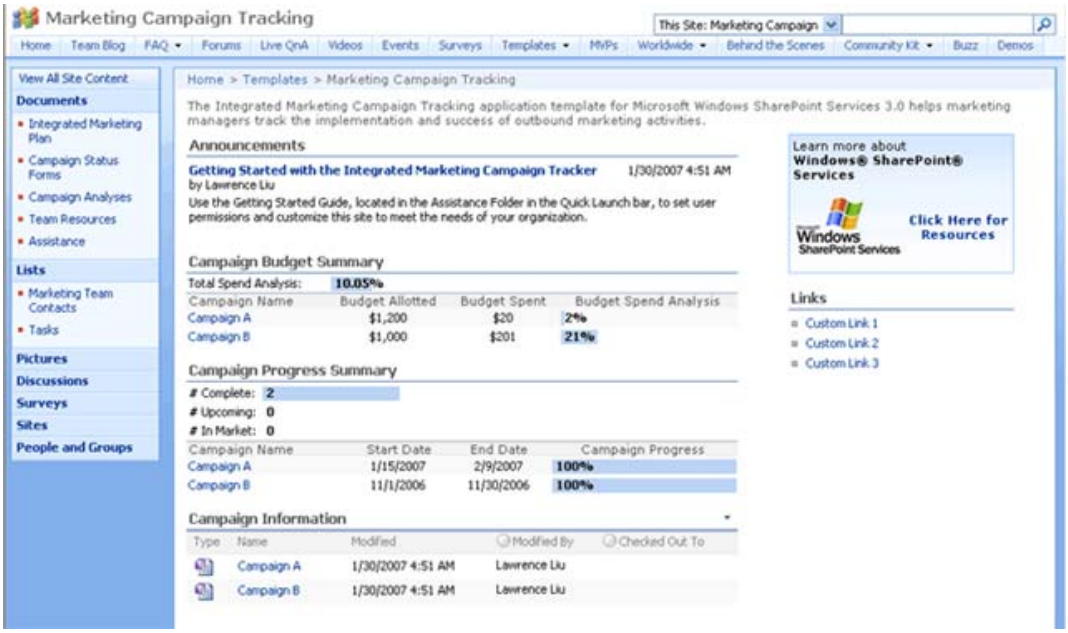

Figure 3.2: Using the data accessed by the BDC.

From an architecture standpoint, the BDC uses ADO.NET and other data-access techniques—such as a Web service proxy—to access back-end data from a variety of sources. It then makes that data available to a variety of SharePoint features. XML-based metadata is stored in SharePoint itself to help manage that data, and some of the back-end data is cached within SharePoint to enable features like search. Figure 3.3 illustrates the basic architecture.

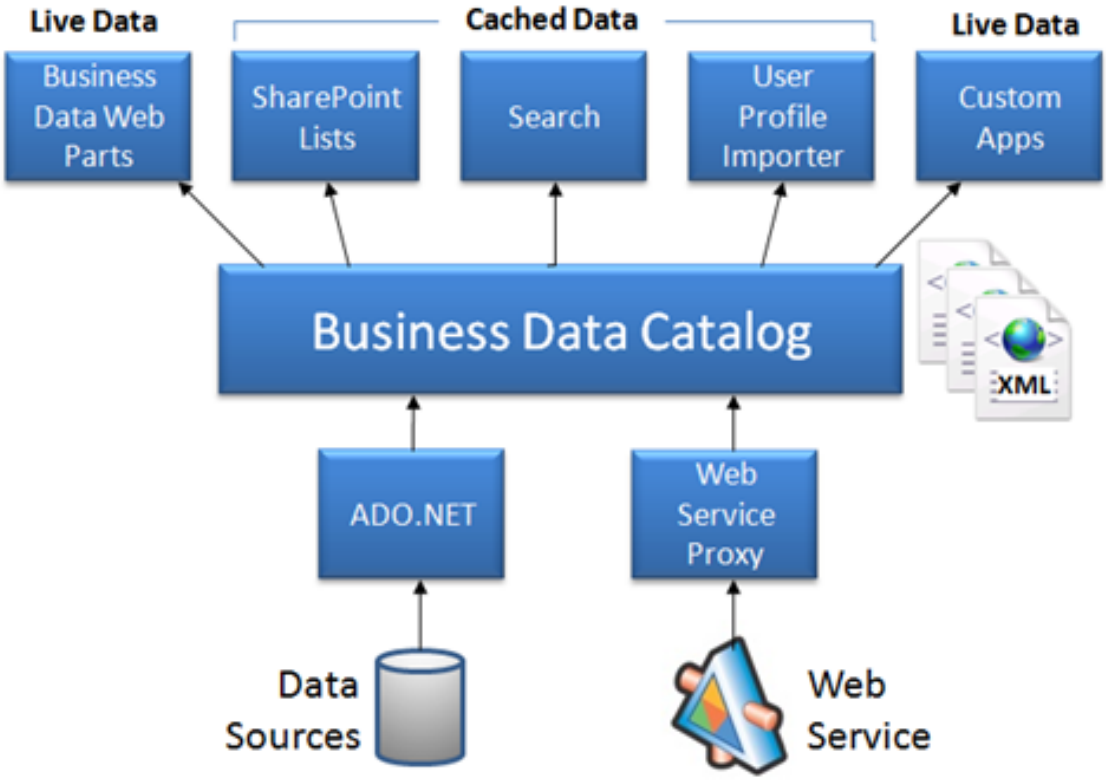

Figure 3.3: BDC architecture.

That XML-based metadata is the real functionality of the BDC. It defines the "mapping" between SharePoint and the back-end data, defining back-end entities, data relationships, and so on. You have to define this metadata when connecting the BDC to back-end data; SharePoint includes tools for doing so.

Business Connectivity Services or "Two-Way BDC"

For SharePoint 2010, BDC was renamed Business Connectivity Services (BSC), and given the ability to do more than just read data from back-end services. Now, it can also *write* data to the back-end. BCS can also be used from within Office 2010 applications. You might use Word to perform a mail merge from a customer database, for example. In SharePoint 2010, "BDC" now stands for "Business Data Connectivity," the service that makes BCS work.

You still have to create XML models of your back-end data, and those models help control the ability to write data back to the back-end. As Figure 3.4 shows (drawn from http://msdn.microsoft.com/en-us/library/ee557658.aspx), the number of ways in which this data can be consumed is even larger than it was with the old BDC. It even includes robust caching so that users—say, someone using Excel 2010—can access data even if the back-end data source isn't accessible at the moment. BCS can sync read and write operations with the back-end server, once it does become available.

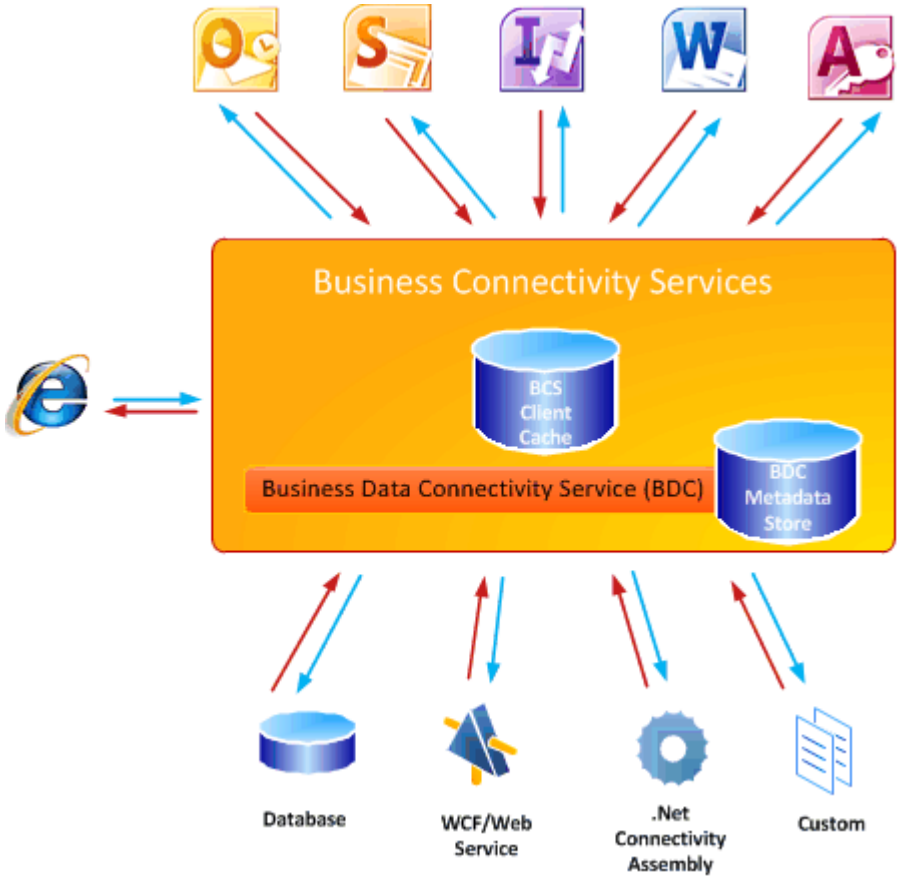

Figure 3.4: BCS architecture.

As shown, you can also write custom .NET Framework assemblies to broker connections to back-end data, enabling you to impose and enforce business logic between SharePoint and your back-end data.

> **Resource**
>
> You can read more about BCS at http://msdn.microsoft.com/en-us/library/ee557658.aspx.

BCS remains a solid way of integrating back-end data into SharePoint. It isn't exactly "no coding" as sometimes implied because you do need to be able to create your data models, but you won't have to actually write programming code in order to create those models.

Migration

Migration: Moving content from place to place. In this case, permanently moving content from file servers or wherever into SharePoint. This isn't an option with back-end data; I'm not presenting migration as an alternative to the BDC/BCS. But for files, media, and so forth, migration is an option.

How It's Done

Migration is typically performed using a tool of some kind—and a plethora of them exist from third-party vendors, although migration isn't a space that Microsoft has directly addressed with any major effort.

At its simplest, migration can involve simply copying a folder hierarchy into SharePoint, while maintaining whatever access permissions the file had on its file server. A more complex migration might also involve restructuring the data because SharePoint offers organizational metaphors and techniques that aren't present in a simpler, hierarchical folder structure on a file server. Migrations can also help automatically populate document metadata, which helps make those documents easier to find through SharePoint's search facilities.

Benefits of Migrating Content

The benefits of migrating content into SharePoint are varied and significant. Your content becomes indexes and searchable. You gain the ability to add version control to documents, and to enforce workflow rules around document updates. You can in many cases employ simpler permissions structures, using SharePoint groups rather than individual user permissions—making long-term permission maintenance easier. You can even make it easier for content to be accessed from a variety of places, since SharePoint not only integrates with the Office suite of applications, but also exposes content via a Web-based interface.

But perhaps the biggest benefit of migration is that you can start to end data chaos. Along with BDC/BCS, migration can help bring all of your content *into one place,* as shown in Figure 3.5. Users have *one* place to go to find everything they need. You can start to manage permissions in this one place, report on utilization in this one place, and so on. You won't need to spend days teaching new users where to find everything on the network, and you can stop mapping a dozen network drives in logon scripts. Users begin to utilize SharePoint as—well, as a replacement for file servers, perhaps. Everything lives in one place.

By keeping authoritative copies of data in a single location, you can also start to avoid the bloat that often accompanies file servers. Users won't have as strong a need to make copies of data into their home folders; they can simply keep smaller shortcuts to SharePoint-stored documents. Everyone will be working from a single copy of information, rather than having dozens of copies spread across everyone's home folders.

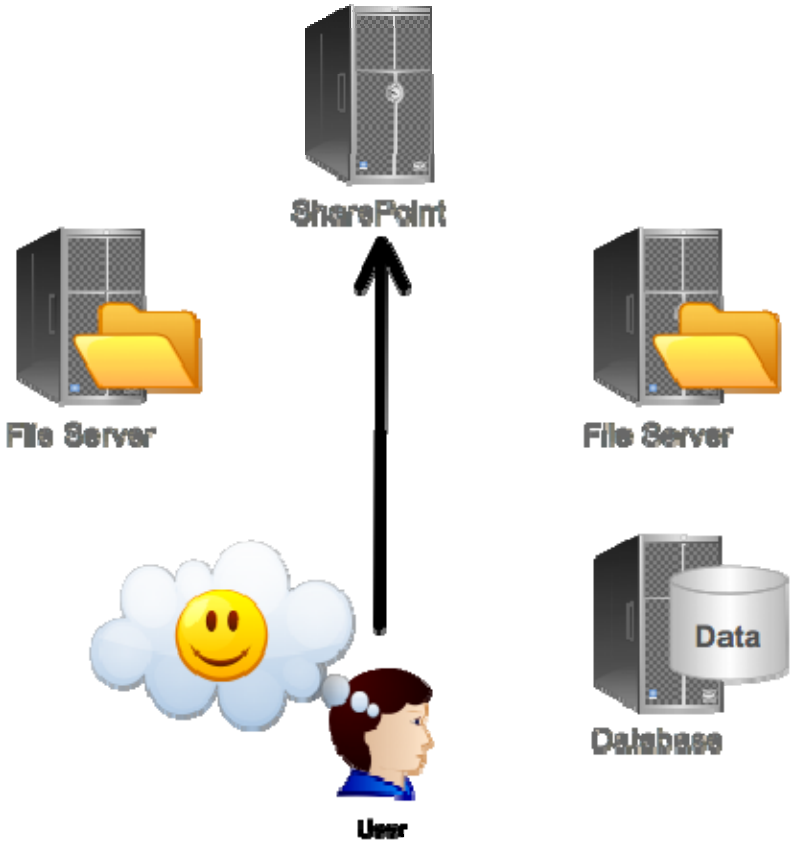

Figure 3.5: SharePoint migrations can help reduce data chaos.

There are a *lot* of strong arguments in favor of migrating *everything* into SharePoint.

Downsides of Migrating Content

There are, however, downsides to migrating content, many of which I've outlined already.

- Large content items are going to take up a lot of room in the SharePoint database. That's going to complicate SharePoint performance management, as well as database maintenance tasks like backups, defragmentation, and so on.
- Media items don't work as well from a database—which as I've pointed out isn't designed for streaming data—as they would from a file system.
- Some file server-based data, as I suggested earlier, *needs* to be on a file server in order to interact with other processes. For example, you might have a process that "watches" a given folder for new files, and does something with them; that won't work if the file "lives" in SharePoint.

Ideally, there's some middle ground, where we can get the advantages of having content in SharePoint—without the downsides.

Goals for External Content

Let's establish some business goals for all of this external content. Let's define—at a business level—what we'd like to do, and then see if it's possible to achieve that.

Keeping the Content External

Our first goal will be to *keep external content outside of SharePoint,* simply because there are benefits in doing so. We'll maintain our file servers for those processes that rely on them, and we'll be keeping our SharePoint database from becoming overly-bloated with large content items. File servers *are really good at storing files,* after all, so let's let them continue to do so.

Let's start building a picture of what we want this to look like. Figure 3.6 will be our starting point: External content (represented as page icons) will remain external. They'll be accessible to external processes that might need them, and they'll remain secured by whatever security (such as SQL Server or NTFS permissions) that's in place.

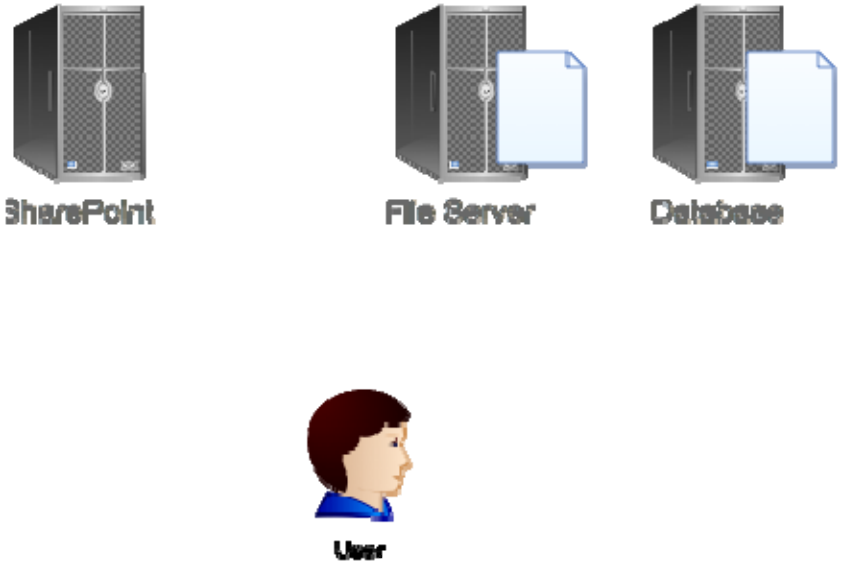

Figure 3.6: Leaving external content where it is.

Of course, now we're facing some downsides, because it's like we don't even *have* SharePoint. So we need to move quickly on to our next business goal.

While Surfacing the Content in SharePoint

We *want* that external content to appear in SharePoint. In the case of database data, that might be through something like a custom application, or a BDC/BCS type connection. The data stays in the database, but SharePoint becomes an interface to it.

Similarly, we want SharePoint to be *aware* of our external files and folders—including media content. We want users to be able to find that content from within SharePoint. SharePoint will thus remain a single point-of-access for our users—helping to kill data chaos—but it will just be "keeping track" of the external content's location, rather than necessarily storing it in the SharePoint database. Figure 3.7 shows this updated goal, with shortcut icons indicating that the content is *surfaced* in SharePoint, but not actually *present* within it.

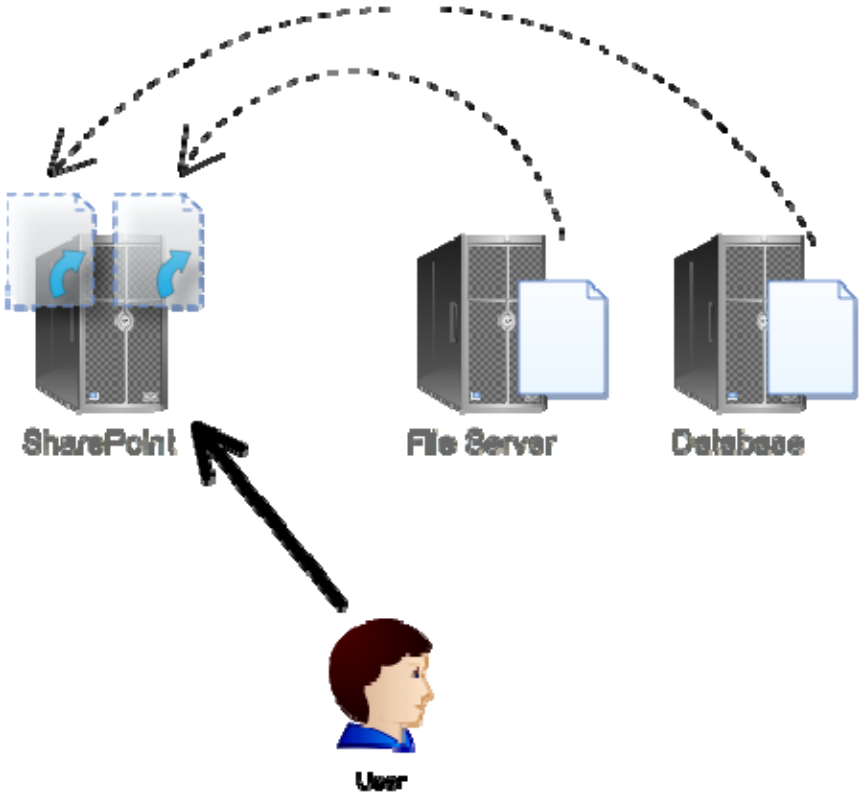

Figure 3.7: Surfacing external content in SharePoint without migrating it.

Now, we need to be very specific about our business requirements for how this content is treated within SharePoint. It's not that tough to just create a little shortcut, within SharePoint, to external content; we want a bit more functionality out of the situation than just that.

And Making External Content a Full SharePoint Citizen

We want our external data to be a "full citizen" of the SharePoint environment. Specifically, we want external data:

- Included in a SharePoint Document Library, not just as a list of external links
- Included in workflow operations
- Available for alerts, so that users can be notified when content changes
- Added to SharePoint's search index

- Taggable with metadata to improve searchability
- Controlled by SharePoint's permissions (at least, all access via SharePoint should be controlled that way)
- Accessed directly from within SharePoint by SharePoint-aware applications like Microsoft Office. We should be able to have Word open up a SharePoint-based Word document, even if that document is technically still living on a file server someplace. We want to remove users' awareness of those file servers, in other words.

Basically, we're asking a lot: We want our content to act as if it lives in the SharePoint database, even if it doesn't.

And we want this for more than just file servers. We also want content located on local and, possibly, remote FTP servers treated this way, along with content that may be located in outsourced, cloud-based storage. In fact, let's update our drawing—in Figure 3.8—to reflect that added requirement.

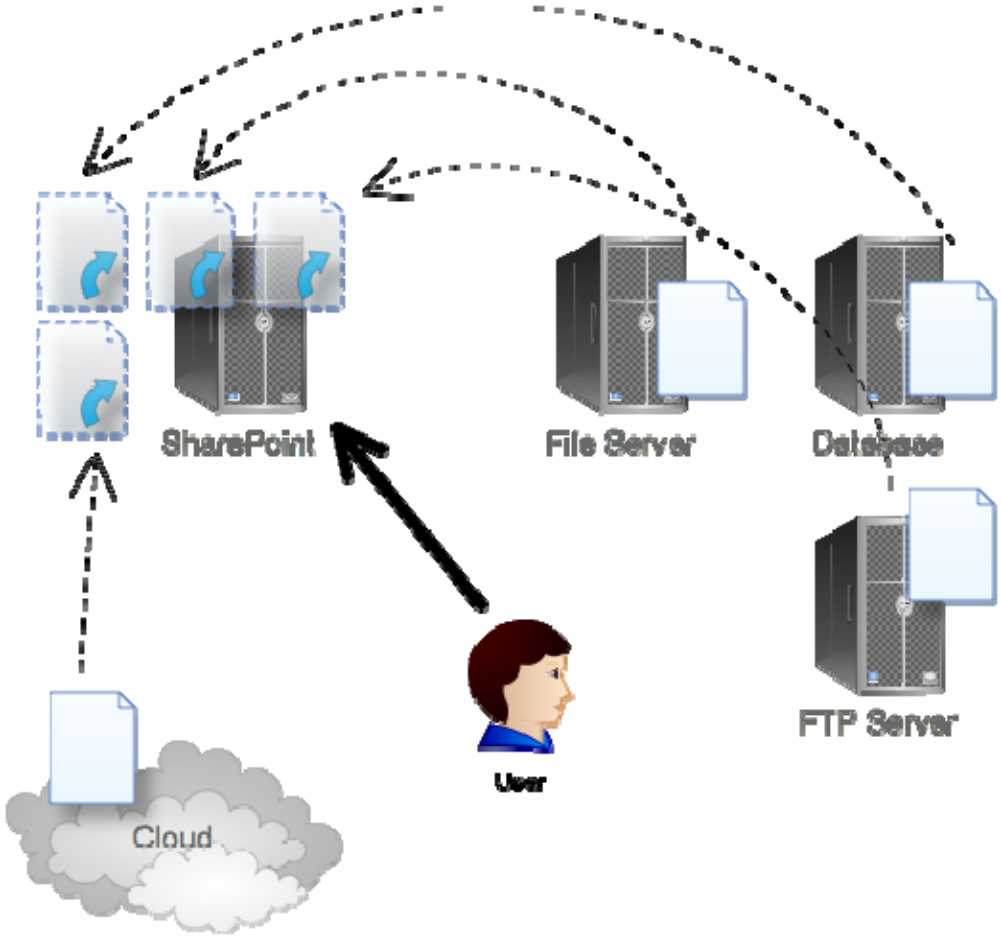

Figure 3.8: Including FTP- and cloud-based content in SharePoint.

We *can* do all of this. We just need to take a more creative approach for getting external content into SharePoint—more creative than a simple migration, that is.

Note

From here on out, I'm going to sort of drop the "external content from a database" aspect of the discussion. SharePoint's BDC/BCS functionality serves that need well, and custom applications are needed if you exceed the capabilities of BDC/BCS. I'm going to focus on external file-based content (including media files) living on FTP servers, cloud storage, or file servers. I'm not implying in any way that database-based data isn't important, just that it's a problem with a clearly-defined native solution.

Creative Approaches for Getting External Content into SharePoint

So let's talk about creative approaches. In general, these are things you'll need third-party software to implement; we're not talking about native SharePoint capabilities. In general, there are two things to discuss: Content connectors and the special considerations necessary for those large, streaming-media files.

Content "Connectors"

A content connector is designed to take external data—living, perhaps, on a file server, FTP server, or some cloud-based storage—and *present* it through SharePoint. Done correctly, this meets all of our requirements: We get to leave our external content outside SharePoint, for a trimmer SharePoint database, but we get all the benefits of having the content "in" SharePoint, like search, workflow, alerts, and so on.

There's a trick, here: It's still possible to access the content from its original location. That means there has to be some kind of two-way synchronization, so that changes in either SharePoint or the native location will be synchronized to the other. For example, if someone changes a filename in SharePoint, we want that mirrored on the file server where the file actually lives. One way to accomplish that synchronization is to have a tool that can monitor both SharePoint and the content's native location for change events. This is a superior approach to time-based synchronization, which simply checks for changes at a fixed interval.

Architecturally, this is accomplished through an extension to SharePoint itself. That extension receives or gathers information about external content, and populates SharePoint's database with the information. Figure 3.9 shows how this might work. Note that I'll use "N" to designate content natively stored in SharePoint's database, and "X" to indicate external content.

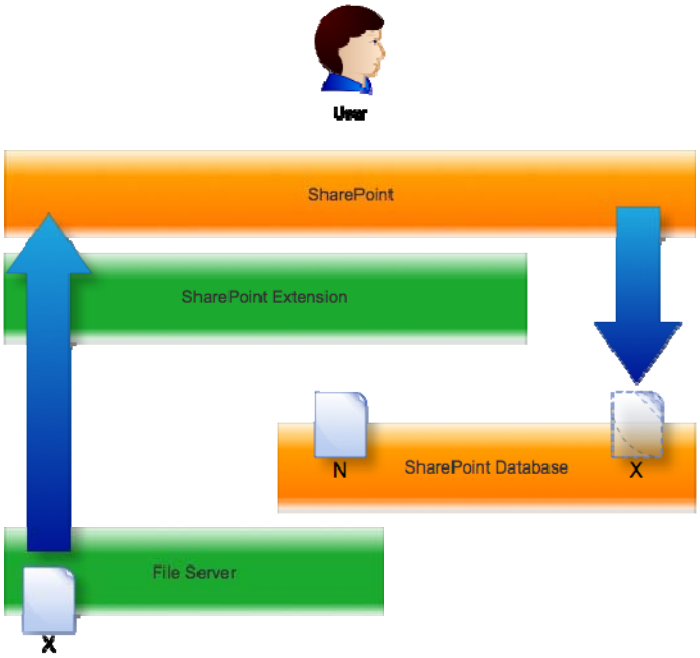

Figure 3.9: Incorporating external content into SharePoint.

That information includes an identifier that the content is, in fact, external, so that when a user attempts to access the content, the SharePoint extension can physically retrieve it. Figure 3.10 illustrates.

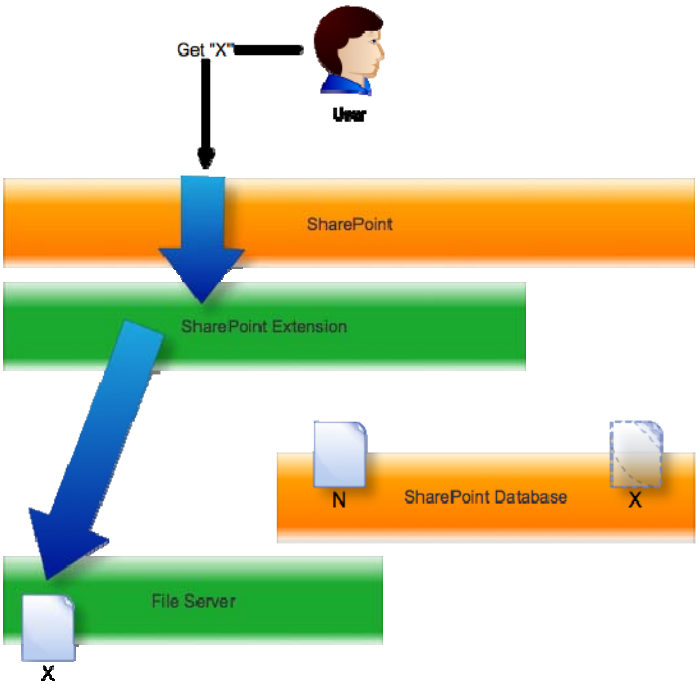

Figure 3.10: Accessing external content from SharePoint.

The important fact to remember here is that the user is *only dealing with SharePoint.* Any application—like the Office suite—that works with SharePoint will work with this external content, because the content *appears* to be living in SharePoint. The extension simply makes the fact that it's really located externally transparent.

Special Considerations for Media Files

You could do that exact same "connector" trick for media files. However, there's a danger in doing so: You'll be putting a decent burden on SharePoint's Web Front End (WFE). Essentially, the WFE is going to become responsible for streaming the media content to the Web-based user. There's nothing *wrong* with that, since it's what Web applications are more or less designed to handle, but there *is* a better way. Simply bypass the WFE.

You still have a SharePoint extension operating, here. It gathers your external content, be it located in a file share, FTP server, or in the cloud—FTP and cloud being more important when it comes to media, since so much of it may be located externally.

> **Note**
>
> Make sure the SharePoint extension you select supports a variety of media types—WMV, WMA, MP3, AAC, VP6, MP4, MPEG, MPG, AVI, WAV, and so forth. You don't necessarily want to be "stuck" just doing Microsoft-friendly media types, or any other vendor's media types for that matter. You want support for them all.

By having the extension stream content *directly to the user's browser,* you offload some work from the SharePoint WFE. That helps keep performance high for other users, and makes SharePoint itself more scalable. Figure 3.11 shows how that might work.

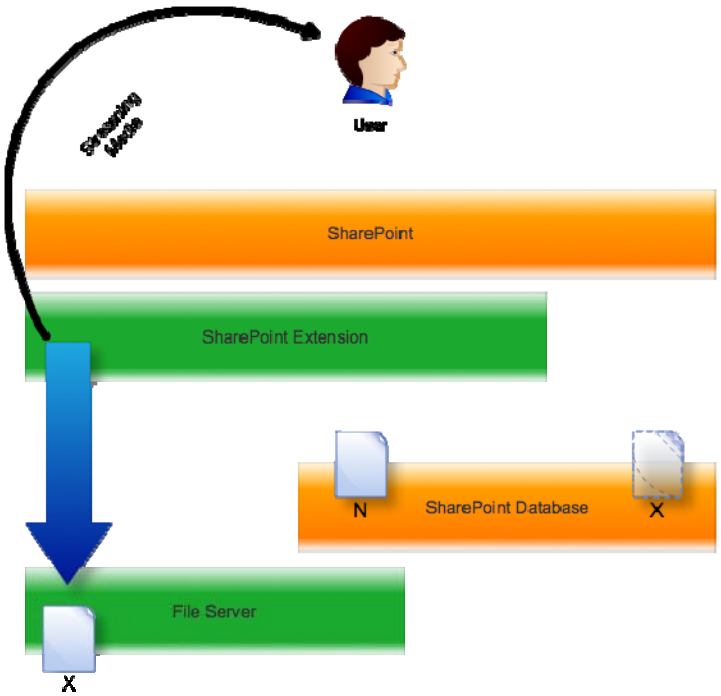

Figure 3.11: Streaming media directly to the user, bypassing the WFE.

How Do You Do It?

There are a number of ways that you could physically implement such a connector. One way would be to simply install an extension of some kind on to each SharePoint server. That might not be optimal in high-load environments, though, since each server would be responsible for maintaining its own list of external content.

A better approach, shown in Figure 3.12, might be to adopt a client-server model. A central "external content server," shown in the middle, would be responsible for loading "stubs" for external content into SharePoint databases. A client agent, running on each SharePoint server, would handle that communication with the central external content server. By populating SharePoint with the appropriate stubs, the system would enable SharePoint to discover, index, view, and manage the external content. The central server would also be responsible for detecting changes and synchronizing them as I described earlier.

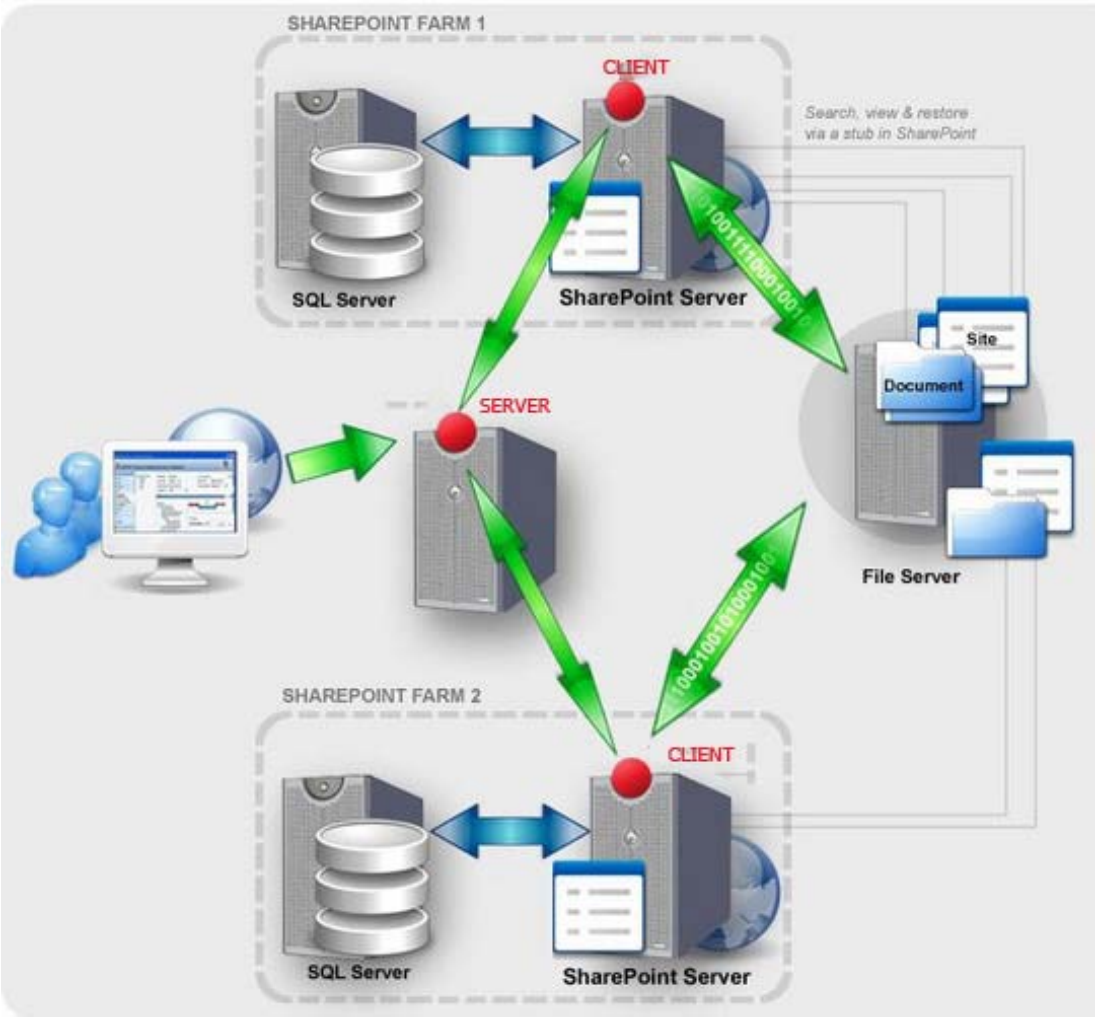

Figure 3.12: Implementing an external content connector.

Obviously, this is just a rough sketch of how the implementation might actually work; each vendor producing such a solution would no doubt have a unique approach.

Coming Up Next

Most businesses are awash in old data, much of which they're required to keep handy. It's a good thing we're closer to the "paperless office" than a decade or two ago because otherwise we'd be drowning in archived content. But "old data" doesn't mean "unneeded data;" you still may have to refer to that dormant, older content from time to time. Keeping it all in SharePoint is a great way to bloat your database, reduce performance, and consume costly storage resources—but removing that data from SharePoint makes the data more difficult to find and access. In the final chapter of this book, we'll look at ways to incorporate dormant and archived content into SharePoint, while still maintaining good performance and responsible utilization of storage resources.