Realtime publishers

Intelligently Reducing SharePoint Costs through Storage Optimization

Don Jones

sponsored by

AvePoint®

Unleashing the Power of SharePoint™

Copyright Statement

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.

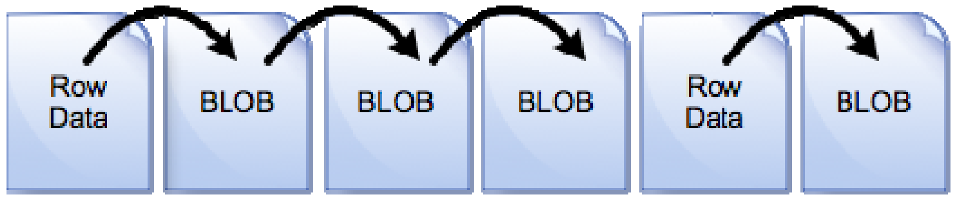

Figure 2.1: BLOB storage in a SQL Server database.

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.

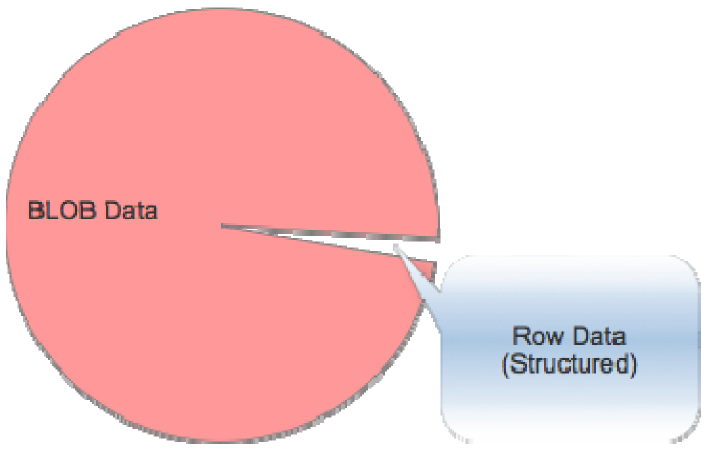

Figure 2.2: Around 95% of many SharePoint databases is BLOB data.

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.

Row Data
Row Data
BLOB

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.

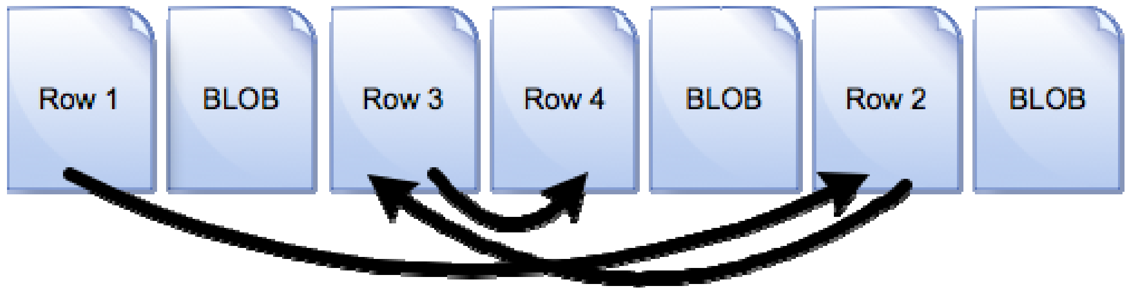

Figure 2.4: Fragmented databases require more reading effort.

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).

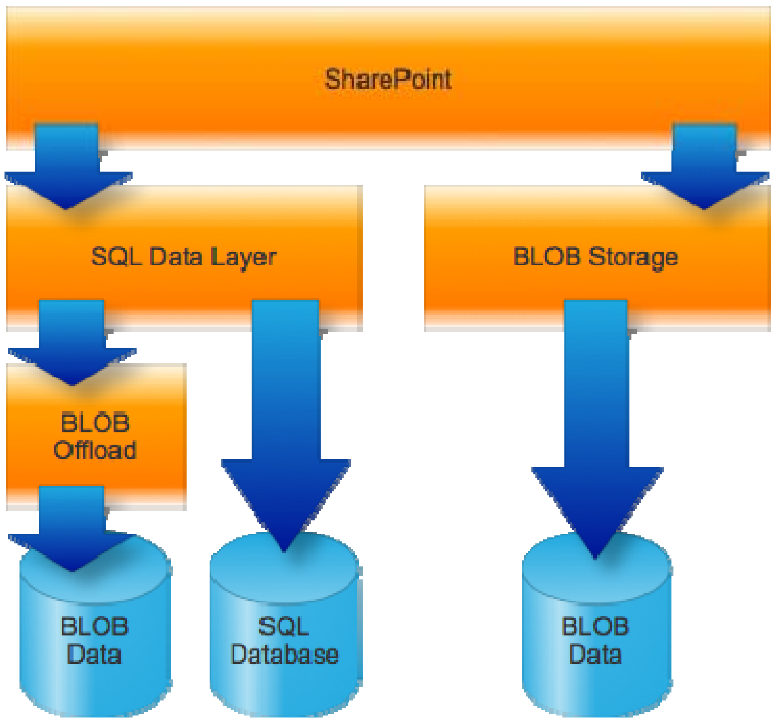

Figure 2.5 Keeping BLOB content searchable.

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.

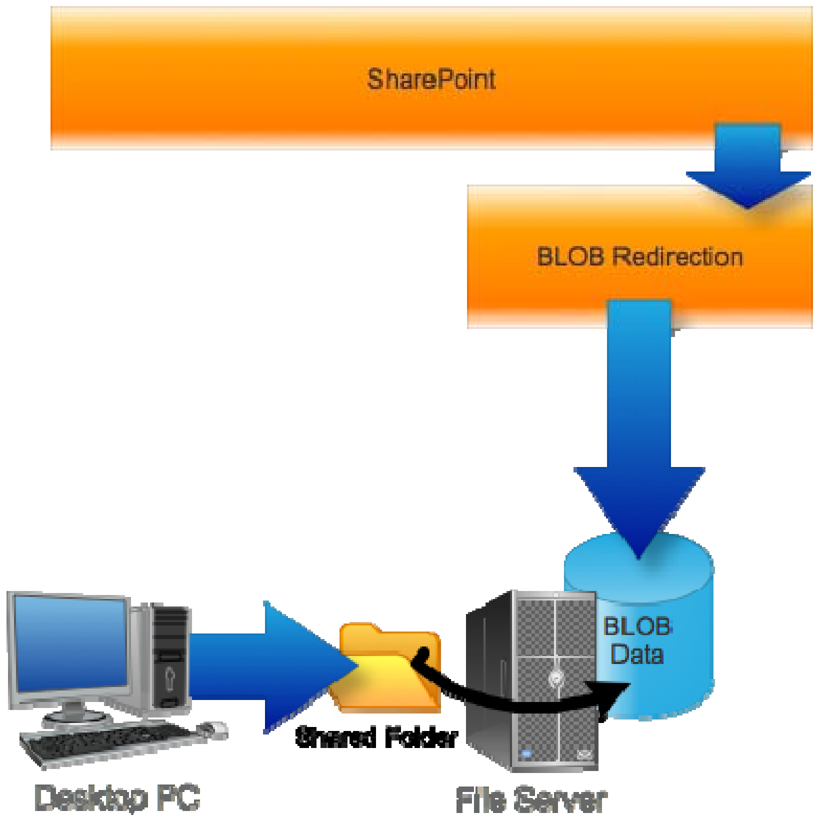

Figure 2.6: You don't want to be able to bypass SharePoint's security.

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.

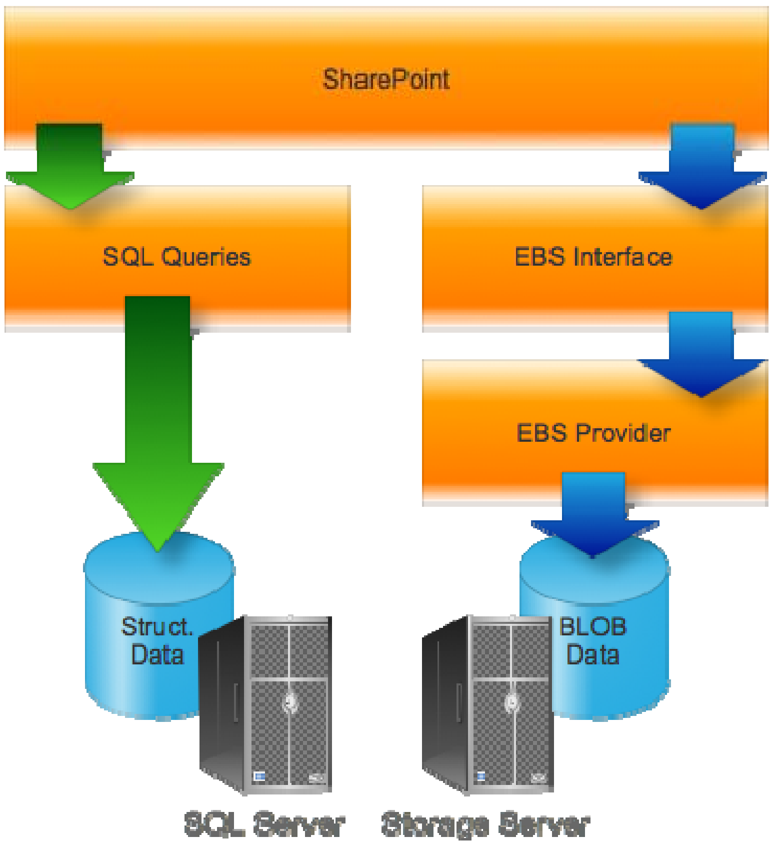

Figure 2.7: SharePoint EBS.

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.

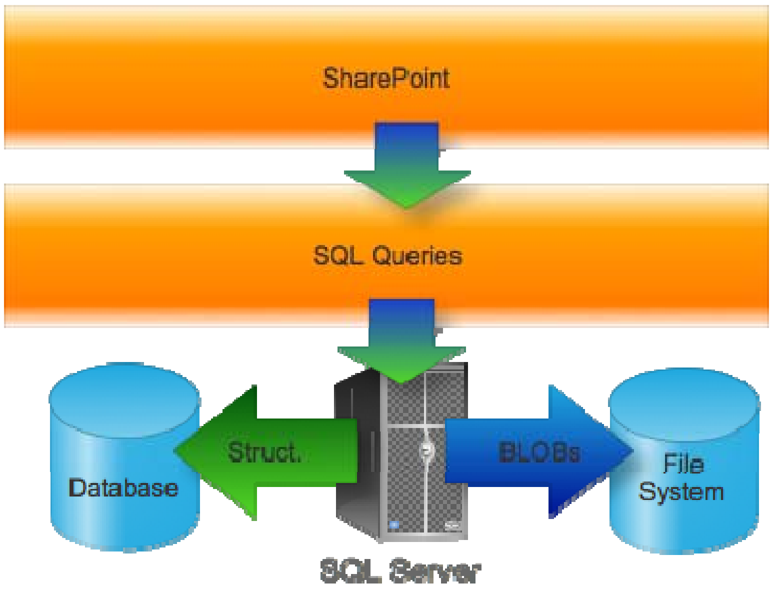

Figure 2.8: How the FILESTREAM type works.

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.

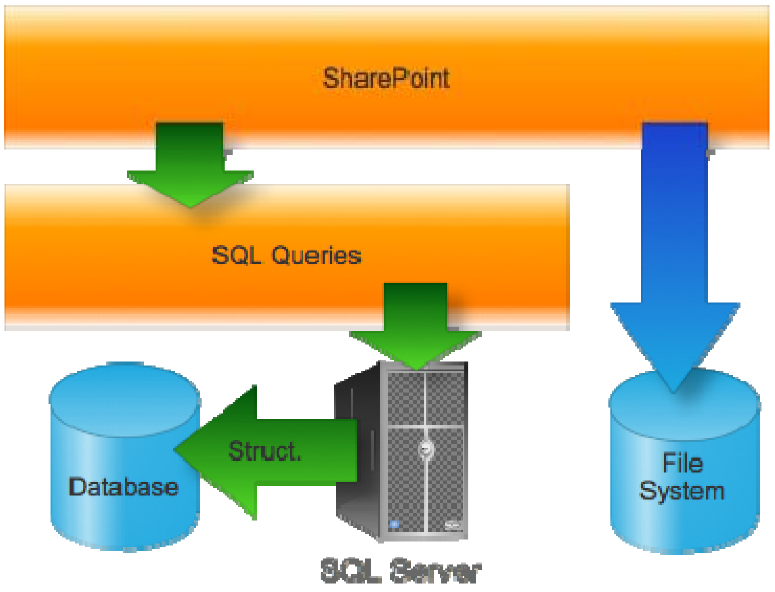

Figure 2.9: FILESTREAM-aware applications.

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.

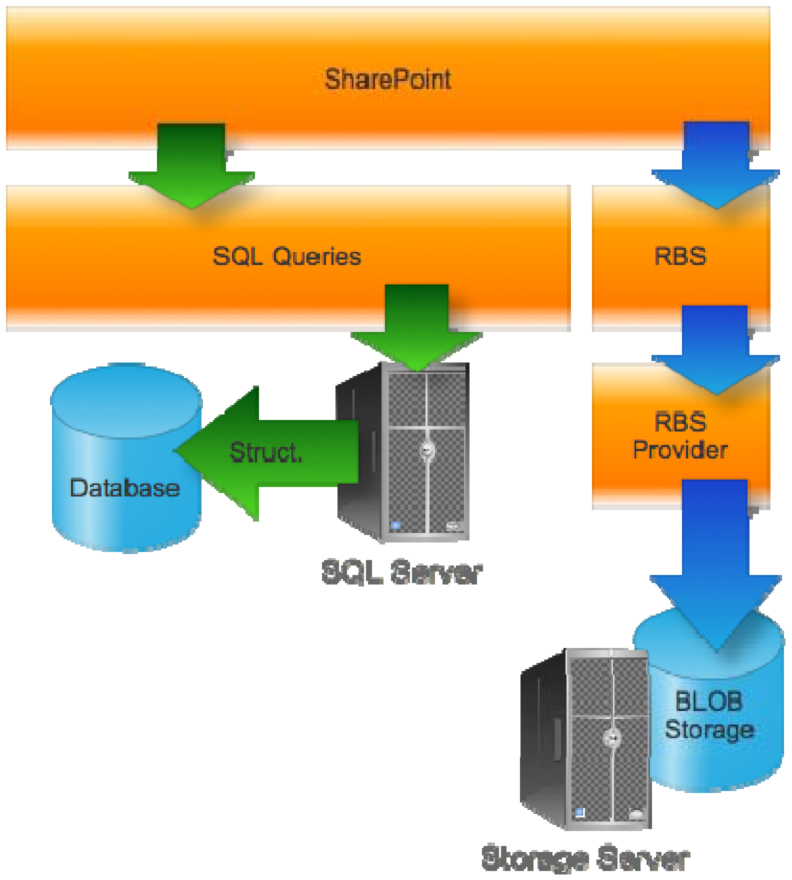

Figure 2.10: SharePoint and RBS.

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.

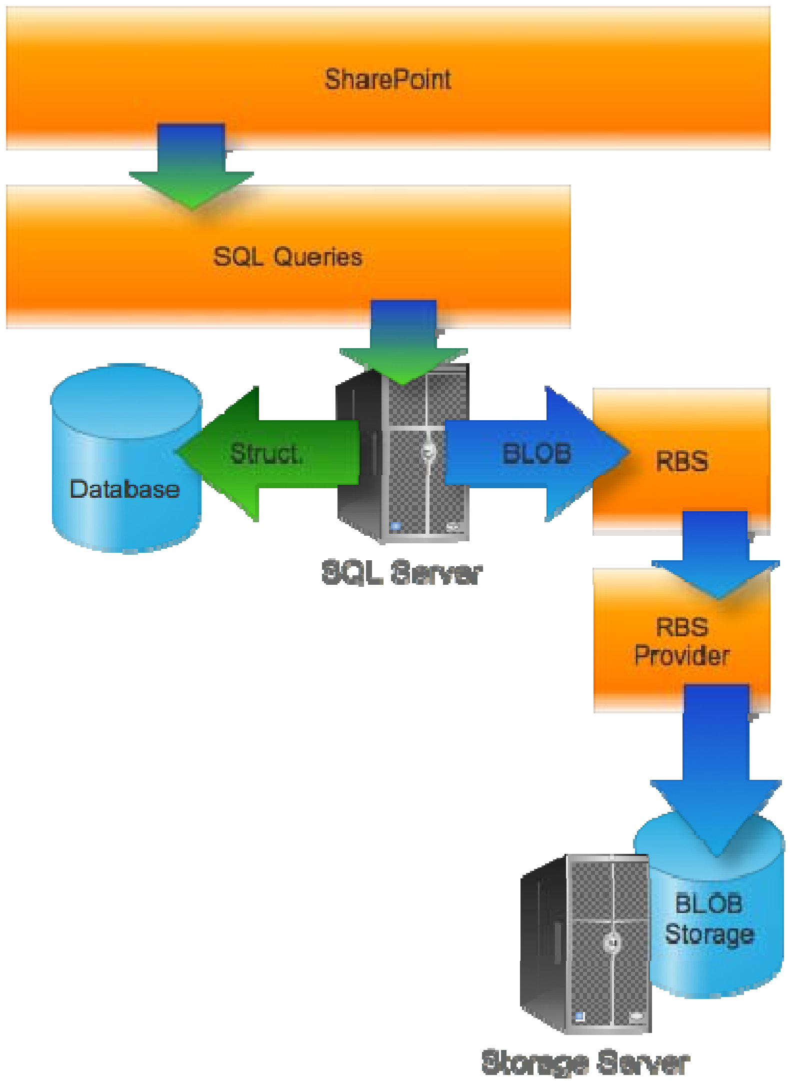

Figure 2.11: Transparent RBS.

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.

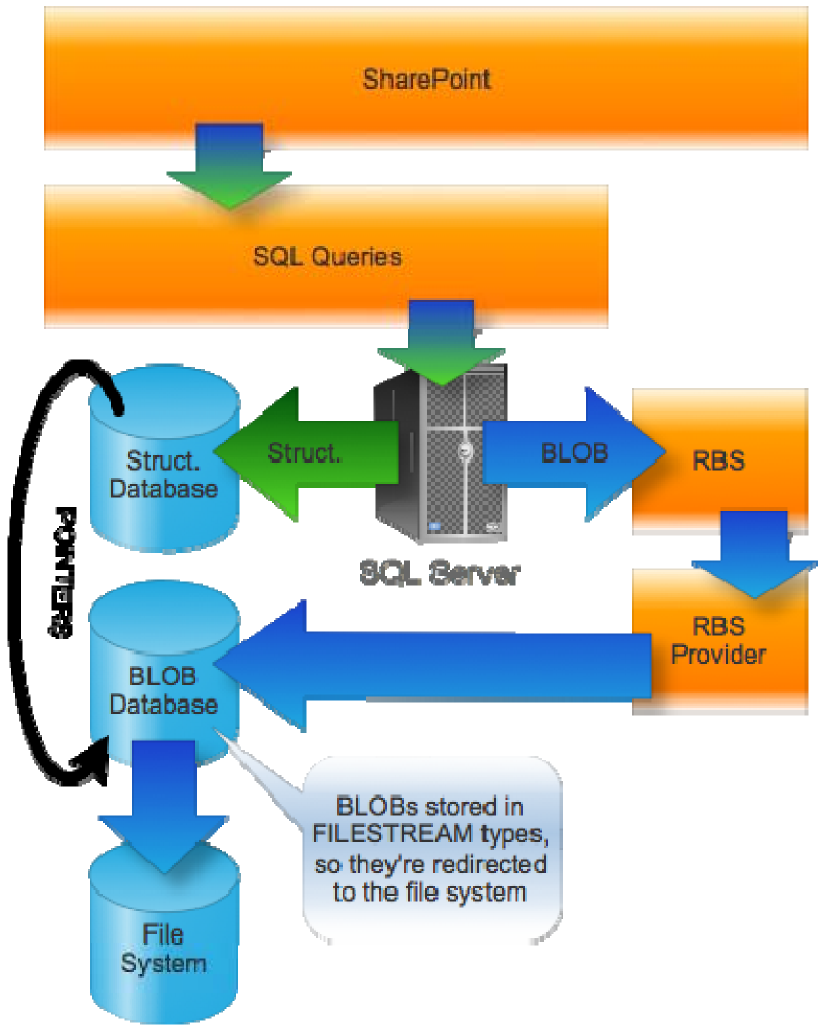

Figure 2.12: Microsoft's FILESTREAM RBS provider.

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

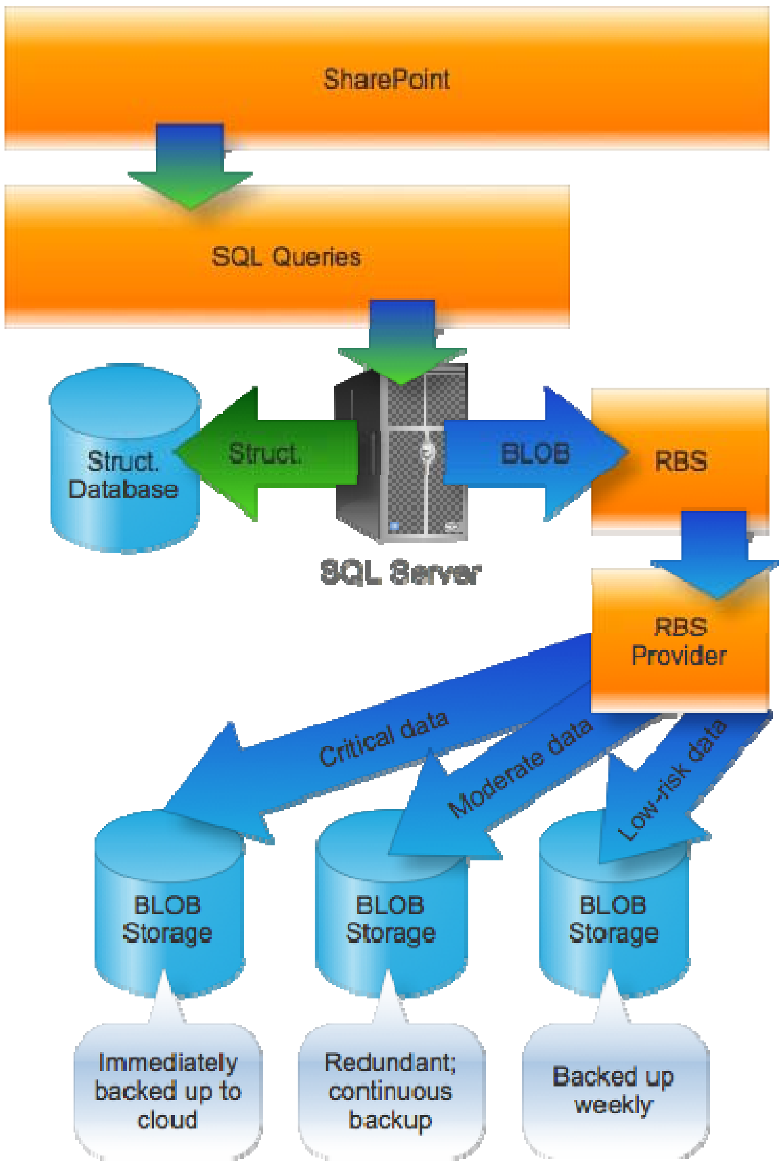

Figure 2.13: Offloading data to different locations.

- 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.