Performance basics for IBM® Lotus® Notes® developers

Andre Guirard
Technical Solution Architect, Lotus Notes
IBM Software Group, WPLC
Minneapolis, MN

May 2008

This white paper addresses the most important and most serious factors affecting IBM® Lotus® Notes® and Domino® application performance. It is intended for developers of Notes client applications, to help them maximize performance by identifying problem areas and offering solutions.

It is assumed that you already know how to create Notes design elements and how to set options on forms, fields, views, columns, etc. There's material here for developers of all levels of expertise.

Table of contents

1 Introduction

It's easy to develop simple applications in Lotus Notes and, if you have a few users and not too many documents, you're unlikely to have performance issues. However, if your application is successful, you may accumulate lots of users and lots of data. If you haven't designed it with performance in mind, your application will slow to a crawl.

This white paper discusses the main factors affecting Notes/Domino® application performance and explains what you, as a developer, can do to maximize performance. This is not an exhaustive guide; rather, we focus on the design problems that are most common and most serious.

The intent of this paper is to help you identify problem areas and point you toward solutions, mainly for Notes client applications. Web applications have largely the same design issues, but they have some additional performance concerns and opportunities that are addressed in the Appendices Group C of the IBM Redbooks publication, Performance Considerations for Domino Applications, and in the IBM Business Partner document, Performance Engineering Notes/Domino Applications.

2 General principles

The following factors have the most impact on application performance generally:

- **Number and complexity of views.** Remove unused views or merge similar views. Where possible, use a re-sortable column to combine views that contain the same documents with different sorting. Remove unneeded columns, and simplify selection and view column formulas. Check for “server private” and other views to which you might not have access.

- **Use of @Today and @Now in view selection formulas or column formulas.** Avoid if possible. Refer to the IBM Support Web site Techdoc, Time/Date views in Notes: What are the options?; also see the Views section of this article below.

- **Number of documents.** More documents make views slower to open. Consider archiving old documents or combining “main and response” documents into a single document. For instance, if your main document is an “order”, it might be a bad idea to make a separate document for each “line item” on the order. Lotus Notes is not a relational database, but a document-oriented database.

- **Number of summary fields stored in the documents.** Every field that's not rich text is called a “summary” field (though this is a slight oversimplification). Documents with more summary fields take more time to index into views (by up to about 30% if there are hundreds of fields). This is true even if the fields are not used in views. Sometimes using fewer documents requires more fields and vice versa; making the right choice for optimal performance requires thought.

- **Complexity of forms.** Attempt to limit forms to the number of fields you actually need. Long forms take substantially more time to open, refresh, and save (as well as contributing more fields that the view indexer must deal with).

- **Modifying documents.** Modifying documents unnecessarily slows view indexing by giving the indexer more work to do, and it also slows replication and full-text indexing.

- **Number of deleted documents.** When a document is deleted, it leaves behind a marker called a “deletion stub.” The replication program needs this to decide whether to delete the same document from another replica, or copy the “missing” document to this replica. Deletion stubs eventually age out (120 days is the default), so for a database with a normal number of deletions, you don't accumulate enough to cause a problem.

  However, we've seen applications in which there are many times more deletion stubs than documents. This usually occurs when there's a nightly agent that deletes every document and then creates all new documents from some outside data source. Don't do this. More advanced algorithms are available that compare documents with your source data and determine which ones need to be updated or deleted. See this Lotus Sandbox download for more information.

- **Reader fields.** If you need to use reader fields, then you need to – there's no other way to get that level of security. But be aware of the performance impact in views, especially if the user has access to only a few documents out of many. The Views section of this paper includes some tips for minimizing the impact, as does the developerWorks article, Lotus Notes/Domino 7 application performance: Part 2: Optimizing database views.

- **Number of users.** A large number of users on a server drags down performance of the application (and of the server). And, if the application already had marginal performance, adding users makes it much worse. Correcting design issues can help, but you might also create replicas on other servers, especially clustered servers, or encourage users to create local replicas, which are *much* faster.

3 Database-level performance considerations

Refer to the Domino Designer help document “Properties that improve database performance” for a list of database options that you can adjust to tweak performance. In most cases these options trade performance for functionality; hence, if you don't need the functionality for a specific application, disable it.

The options that are likely to have the most noticeable effect are:

- **Don't maintain unread marks.**
- **Don't maintain the "Accessed (In this file)" document property.** If you “don't maintain,” you can't tell when a document was last read. This information is handy in archiving documents that haven't been read for a while.
- **Disable specialized response hierarchy information**. If you disable this, you won't be able to use @AllDescendants or @AllResponses, which are occasionally useful in view selection formulas and replication formulas.
- **Disable transaction logging**. The effect of this on performance depends on how the administrator has set it up on the server and on the number of users. If there are many users, using transaction logging may be faster than not. Try it both ways and measure. Transaction logs are used for recovery.
- **Optimize Document Table Map**: This is most useful in situations in which an application contains roughly equal numbers of different types of documents, and most views only display one type (for example, SELECT Form = “xyz” & ...). If the view selection formula is written in just this way, with the form test first, view indexing is faster because it can immediately eliminate from consideration every document that doesn't use that form.

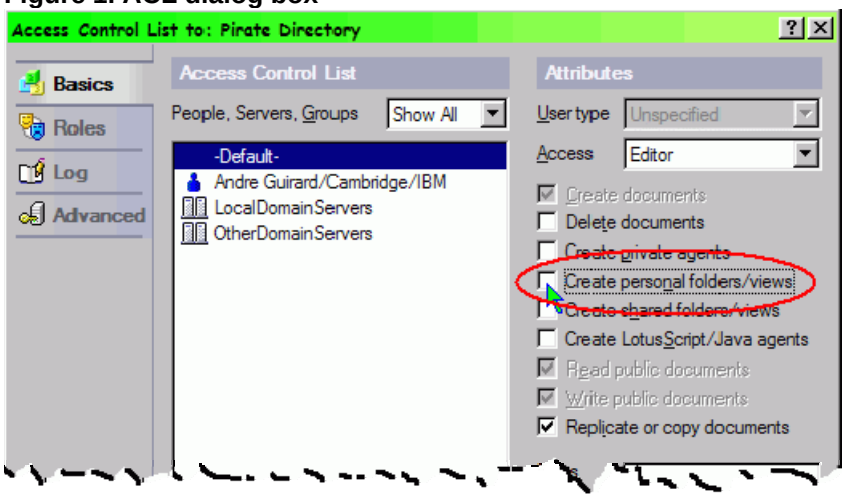
Using NSFDB2 (storing your Domino data in a DB2® database) does not help performance and, in fact, is usually a bit slower than a traditional NSF file. The goal of NSFDB2 is adding functionality, not boosting performance.

A full-text index can take up a lot of room on disk, but it's usually worth it. You can leverage this to do faster searches in agents, and if you don't have the index, users are forced to use slower searching methods that tie up the server and take longer to yield results.

> **NOTE:** A new database property in Notes version 8.0 lets you turn off "full-text" searching of a database if isn't full-text indexed. Generally, this is a good idea even if you have a full-text index; it guarantees that if the index is deleted accidentally, users will see a message rather than just suddenly having poor performance with no explanation.

Another thing you can do at the database level is not in the property box, but in the ACL dialog box. Limiting users' access to create personal views and folders reduces the load on the server (see figure 1).

Figure 1. ACL dialog box



If you uncheck the “Create personal folders/views box,” users will still be able to create private views, but the views will be stored locally in their desktop file, instead of on the server, so they will not affect performance of the application as much.

Desktop private views do have performance implications because, to index them, users must pull data from the server in real time. So heavy use of desktop private views can also bog down a server. For this reason, avoid automatically creating personal views for the user with the “Private on first use” view option. (This is covered in more detail below.)

4 Formula performance

Most @Functions are fairly fast, but there are a few that take longer to evaluate. Be aware of these, and use them wisely:

- **@Contains** isn't a very expensive function, but it's often used to test whether a list contains an exact value, which is both inefficient and incorrect. For instance, the expression @Contains(Cities; “Lansing”) returns True if Cities contains the value “East Lansing”. If this is what you want, fine; but if you were actually looking for entries that contain the *exact* value “Lansing”, use =, *=, or @IsMember instead. These are faster because they don't need to scan the whole string if the first characters don't match.

- **@For** and **@While** can often be replaced with the more efficient @Transform, or with other functions that operate on the whole list at once.

- **@Unique**: Since this function must compare each value of a list with each other value, the execution time is proportional to the square of the number of items in the list. It's better if you can retrieve a list whose values are already unique. More about this later.

- **@NameLookup** is similar to @DbLookup, but for directory information only.

- **@DbLookup, @DbColumn**: Overuse and misuse of these functions accounts for the vast majority of delays on forms, so they warrant a section of their own (below).

Looping functions in macro language are often used unnecessarily. *Although it doesn't state this in the Domino Designer help docs,* nearly all the macro functions that accept string arguments can also operate on a list. For instance, @Left(x; ","), where *x* is a list, returns a list in which each element has been "lefted".

> **NOTE:** In the past, the functions **@UserRoles** and **@UserNamesList** caused a big performance hit, but beginning with Lotus Notes 6.0, the results of these functions are cached.

4.1 @DbLookup and @DbColumn

Three main factors that affect performance of @DbLookup and @DbColumn are whether:

- you use the cache
- the view you're looking up to is efficient
- you use them unnecessarily

4.1.1 Using cache

Many developers overuse the "NoCache" option, particularly in keyword formulas. It's easy to see how this can happen, because during development and initial testing the keywords tend to be edited often, and NoCache "makes them work right."

However, once in use, an application usually doesn't have keywords edited daily. Some delay before new values are available to users is an acceptable tradeoff for better performance. Use NoCache only when really necessary.

There are three caching options:

- "Cache" (the default) refers only to the view the first time you do a particular lookup during an application session, and it remembers that lookup result for use thereafter until you exit the application.
- "NoCache" bypasses the cache and always goes to the view. If there is a cached value for the same lookup, it is not updated.
- "ReCache", the forgotten option, always goes to the view, but it also *updates the cache* with the lookup value. By using ReCache, you can deliberately update the cache at specific times – say, when a document is saved to which the lookup refers. At other times you can use the cached value, because you know that the value is at least up to date with the information entered by this user.

4.1.2 Choosing the correct view for a lookup

Sometimes a view that is otherwise efficient is still not the best choice for @Db functions. For instance,
@Unique(@DbColumn(“”:“NoCache”; “”:“”; “InvoicesByCompany”; 1)) has multiple problems:

- It uses NoCache where it probably doesn't need to. You don't add a company every day, and when you do, you can use the "ReCache" option in the Postsave of the Invoice form to make the new name immediately available.

- The current database is specified with the expression “”:“”. Instead, use “” because “”:“” not only has more confusing punctuation, but also it's a bit slower to evaluate.

- Don't look up a list that contains duplicate values and then use @Unique to remove the duplicates. Instead, do your lookup to a view column whose values are already unique, because they come from a categorized column.

This last point is especially important, because a column lookup that works fine with 100 documents of test data can degrade very quickly, once you start to use the application in practice and get thousands of documents. Especially when the application is used on a server, it takes time to send the complete contents of a view column over the network to the user's workstation. Reading already-unique values from a view is much faster.

> **NOTE**: The “Generate unique keys in index” option might look like an alternative to categorized columns for obtaining a list of unique values, but it has drawbacks that make it unsuitable for such use.

It's also a bad idea to look up to a view that takes a long time to index, and specifically to a view that uses @Today or @Now in its selection formula or in a column formula. If you just need to look up documents for a particular date, then instead of using @DbColumn to a view that contains only those documents, use @DbLookup to a view of all documents sorted by date and provide the date as a lookup key.

4.1. 3 Avoiding repeated lookups

Using @Db functions unnecessarily can occur in several ways. Here are a few common ones:

Repeated in a formula

```
@If(@IsError(@DbLookup(“”: “NoCache”; “”; “SomeView”; CustID; 3);
        “”;
        @DbLookup(“”: “NoCache”; “”; “SomeView”; CustID; 3))
```

Not only does this formula use NoCache where it probably doesn’t need to, but it looks up twice where it only needs to do it once. Here are two alternatives:

```
_tmp := @DbLookup(“”; “”; “SomeView”; CustID; 3);
@If(@IsError(_tmp); “”; _tmp)
```

or

```
@DbLookup(“”; “”; “SomeView”; CustID; 3; [FailSilent])
```

Unnecessary keyword lookup in read mode

When a document is opened for viewing, for certain types of keyword fields the Notes client doesn't need to know the list of choices. Obvious exceptions are checkbox and radio button fields, in which all the choices are displayed even in read mode, and anything that uses keyword synonyms (“Display text|value”) since the document only stores “value,” but the form must know to display “Display text.”

In other cases, though, write the keyword formula to defer the lookup until you actually need the list of choices:

```
_t := @If(@IsDocBeingEdited; @DbColumn(""; ""; "Customers"; 1);
@Return(@Unavailable));
```

@If(@IsError(_t); ""; _t)

By returning @Unavailable when the document is in read mode, the formula tells the form to ask again if it needs the list of choices later. That will occur when the user changes to edit mode and the cursor enters that field.

So not only do you avoid doing lookups when the user is merely viewing a document, but you also spread out the delay over editing the document; eight half-second delays are a lot less annoying than one four-second delay. And if the user doesn't put the cursor in that field, they won't need to wait for the lookup at all.

Several lookups where one lookup would do

Suppose you have a customer ID stored in your “invoice” document, and you want to use that ID to look up and display the customer name, their address, and their purchasing contact name. So you have several Computed for Display fields on the form, each one containing a formula that uses @DbLookup(“”; “”; “CompanyByID”; CustID; *x*) where *x* is a column number or field name.

It's more efficient to have a single column that contains all the values you need, which you can then pick apart to get the individual field values. So the column formula might be:

CustName : StreetAddress : (City + “ ” + State + “ ” + Zip) : PurchasingContact

On your form, add one hidden Computed for Display field named CustDetails, as follows:

@DbLookup(“”; “”; “CompanyByID”; CustID; 4)

(assuming the combined column is column 4). Then, you would use this formula where you wanted to display the name:

CustDetails[1]

and so on.

Repeating a lookup on refresh

Suppose you need to look up the user's manager's name into a computed field when composing a form, as follows:

@DbLookup(“”; “VOLE1”: “EmpData.nsf”; “EmpByName”; @Name([CN]; @Username); “Manager”)

Computed fields are recalculated every time the form is refreshed. Many forms are refreshed often (because you enabled the option to refresh fields on change of a keyword field), so this can be a significant slowdown. Making the field “Computed when Composed” would be better.

If you don't need to store the field in the document (and remember, don't store fields you don't need to store!), then you could make it Computed for Display instead, but in that case, do the following to avoid repeating the lookup on refresh:

```
@If(@IsDocBeingLoaded;
        @DbLookup(“”; “VOLE1”: “EmpData.nsf”; “EmpByName”; @Name([CN];
@Username); “Manager”);
        @ThisValue)
```

Assignment of sequential numbers using @DbColumn

This is a frequent mistake. When designers must create a unique ID for each document, they often do so by adding “1” to the number of the latest existing document. So they write a formula such as this:

```
tmp := @DbColumn(“”:“NoCache”; “”; “RequestsByNumber”; 1);
nextNumber := @If(tmp = “”; 1; @ToNumber(@Subset(tmp; -1)) + 1);
@Right(“000000” + @Text(nextNumber); 7)
```

This is a really bad idea. As the number of documents grows, the @DbColumn takes longer and longer to execute. Plus, it doesn't actually guarantee a unique ID when there are multiple users of the application, particularly if there are multiple replicas.

If the number is assigned when the document is saved, it's not available until then, which is inconvenient. Whereas, if it's assigned when the document is composed, that allows a lot of time for someone else to create and save a document with the same number.

You might reconsider your requirements. People often ask for sequential numbering when, in fact, the application requires only a *unique* identifier that need not be numeric. Look into the @Unique function, which generates a reasonably short value that's almost certain to be unique (uniqueness can be guaranteed with a little extra work, for example, by assigning each user a unique "suffix", often their initials).

If you decide that sequential numbers are actually required, see the developerWorks article, Generating sequential numbers in replicated applications, for one way to do it reasonably efficiently. Look for more on this subject in an upcoming developerWorks article.

5 Form design

In this section we address some common problems worth noting.

5.1 Don't use Computed fields if Computed for Display works

Since storing fields slows up the application generally, it makes sense to avoid storing values if you can easily calculate them when needed. There's a tradeoff here; a Computed field isn't calculated when the document is opened in read mode, so if it's a slow formula, it's better to store it, to improve read mode performance (on the other hand, that also means it can be out of date).

But definitely never have a Computed field that simply redisplays the value of another field – in which case you're then storing two copies of the same information.

5.2 Hundreds of fields

The most common reason for large numbers of fields on a single form is that there's a table with multiple rows and columns of information, and a field in each cell, up to the

maximum number of rows you support. This is a tough situation, because this is by far the easiest way to provide this functionality.

There are, however, alternate ways to manage tables of values. The most obvious is to actually put a table in a rich text field and let users fill in whatever they want (use @GetProfileField in the rich text field's default formula to read a “starter” table from a profile document). The drawback is that the user gets no help filling in the cells, such as you could provide with keyword lists, translations, and validations if there were individual fields. But sometimes this is an acceptable alternative.

There are also some published tools and techniques for editing tables one row at a time in a dialog box and displaying the results in a table. For example, the Domino Design Library Examples in the Lotus Sandbox contain a set of design elements that can be used to edit and display data in tables, without having a field for each cell. The system is described in detail in the document titled “Table Editor” in the Documentation database. It takes some work to implement, but it does wonders for performance.

Sometimes we see forms containing many fields that are left blank in most documents. For instance, perhaps 5% of your documents need a “regulatory approval” section containing 50 fields. In the other 95%, you're wasting space and causing poor performance by storing all these blank fields.

It would be better in such a case to have two different forms – a main form with the fields that are always needed, and a separate “Regulatory Approval” form, which might be a response to the original document, and only created when required. This is a case in which it's better to have some extra documents, to avoid having many extra fields.

Don't forget about multivalued fields. Rather than having five fields that let the user enter up to five separate values, use one field that allows multiple values. Then there is no limit to the number of entries (unless you choose to impose one), and the resulting values are *much* easier to use in views and formulas.

> **NOTE:** If an application is already slow due to having too many fields, editing the design elements alone does little to speed it up; you must also write agents to go through existing documents and delete the extra items already in them. There are Business Partner products available that make this simple. If, however, your change is a major reorganization, such as moving some fields onto a special response document, these agents may be fairly complex. It saves time to think ahead and do it right the first time.

5.3 Excessive graphics

Some forms go hog-wild with graphics, using large bitmaps for the background and many decorative doodads. Large images take time to load, take up memory in your design element cache, and take time to draw when you view the form. A little care when creating the form can yield a professional appearance without paying a big price in performance. Here are some tips:

- Never paste an image onto the form; instead, either use an image resource design element or import the image. If you plan to use the same image on multiple forms, an image resource makes sense because it lets your client cache the image separately from the form design. Even if you don't *plan* to use the same image in multiple forms, it's nice to make it an image resource, because you never know whether someone

later will want to create another form with the same image.

- Never scale an image down to the size you want *after* placing it on the form. Use a graphics editor (for example, GIMP) to scale the original large image to the size you require – even if this means you need multiple image resources of the same image in different sizes.

  While you're at it, if the image is a JPEG, try different compression settings to see whether you can reduce the size of the file. JPEG compression is “lossy,” so the image will not be as true to the original if you do this, but if you compress it as much as you can without a visible loss of quality, your form will load faster. There are tools for purchase that can help you find this balancing point.

- Use the correct file format for your images. If the image uses a limited palette – like most logos – GIF format will usually produce the smallest file. If it's a full-color photo or painting, JPEG is usually best. Never use BMP files as they're generally not compressed at all.

- Table cells and graphic cell backgrounds take time to draw. Hidden cell borders render faster than visible borders, particularly borders with 3-D effects. Tables with merged cells render faster than tables nested inside other tables.

5.4 Stored forms

Don't use stored forms. Just don't.

5.5 Automatically refresh fields

The form option “Automatically refresh fields” should be used rarely. It makes the form refresh quite often during editing, causing delays as computed fields and input translation formulas are recalculated. It's generally better to use the field-level option “Refresh on keyword change”, or the field events Onchange or Onblur, to cause a refresh only as needed.

5.6 Too many shared design elements

Forms can pull information from several other design elements, like image resources, shared fields, shared actions, subforms, outlines, stylesheets, and script libraries. It's quite possible that opening one document will read information from a dozen design elements besides the form, which takes time. The advantage of shared design elements is that they make the application easier to maintain. The disadvantage is that accessing multiple notes at load time takes longer.

Lotus Notes maintains a cache of design information, so the design information doesn’t need to read from the original design elements every time; however, initial load time can be a concern. Caching also means that using shared design elements can aid performance, if the same subform or image is used in many different forms.

Shared actions don't hurt performance because there's only one design note containing shared actions, so you can include several for the price of one. Shared view columns don't affect performance.

Due to the maintainability advantage, it’s recommended that you “un-share” design elements only after you've tried other measures and performance still isn't acceptable.

6 Views

Inefficient and unnecessary views cause delays due to the:

- time it takes to update the index when the view is opened.
- time it takes to retrieve information when the view is used in an @Db function.
- the Update task on the server periodically checking each view to see whether it needs to be updated with recently modified documents. More views (or more complex views) therefore tie up the server and slow down all applications.

Another common cause of views opening slowly are large numbers of documents in the database. When you open a view, Lotus Notes checks whether there are any documents modified more recently than the last update time of the view index. The more documents you have, the longer it takes to do this test, even if the answer turns out to be “no.”

6.1 @Now or @Today in views

Much has been written about how to provide date/time-based views without using @Today or @Now. One example is the IBM Support Web site Techdoc, Time/Date views in Notes: What are the options?, which provides alternate ways of creating date-based views.

Let’s now discuss a few additional points. First, the often-repeated advice to use @TextToTime(“Today”) is incomplete. By itself, this only works for the first day. You must do extra work to make this function correctly.

Why? Ordinarily, when you open a view, Lotus Notes looks at the “view index” – the stored list of documents and row values in the view – and examines only the documents created or modified since the index was last updated, to see whether they should be added to the view, or removed, or their column values recalculated. If there were no documents modified since the view was last used, the process is very quick.

If, however, you use @Today, the old view index is no longer useful. For instance, suppose the selection formula is:

SELECT Status = “Processing” & DueDate <= @Today

Documents can be added to this view, *even if they haven't changed*, because the value of @Today has changed since the view was last used. So every time you use this view, Lotus Notes discards the old view index and looks at *every document in the database,* to determine whether it belongs in that view and to recalculate column values.

If you use @TextToTime(“Today”) instead of @Today, you can “outsmart” the view indexer. Congratulations. Lotus Notes will reuse the old view index and examine only modified documents. This is faster, but unfortunately it yields incorrect results, because when @Today changes, we must look at all the documents again.

Suppose you have a column that shows a red exclamation point if a “request” document is still open after three hours (testing it against @Now). That situation can change, even if the view was last used five seconds ago. With @Today, though, it would be nice to just have the view index updated less frequently.

You can actually do this by using the view indexing options in the view Property box. On the Advanced options tab, you can specify that the view be updated “Auto, at most every x hours,” where x is a number you specify. The advantage is that the view opens very quickly. The disadvantage is that the view doesn't immediately show changes even to documents that *have* been modified. The user must manually refresh the view to see the latest data.

Another popular alternative is to create a scheduled agent that executes nightly, updating the view selection formula (using the NotesView.SelectionFormula method) to contain that day's selection formula. For instance, such an agent might contain the statement:

view.SelectionFormula = {SELECT Status=“Processing” & DueDate=[} & Today & {]}

There are, however, some disadvantages to this:

- The view design change must replicate everywhere before all replicas show correct documents.
- Server administrators may be suspicious of agents that change the design of a production application.
- The first user to open the view the next morning still must wait for the view to index. You can get around this by setting the view indexing option to “Automatic” or by having your agent refresh the view.
- If the database gets its design from a template, your view will be overwritten from the template. To avoid this, you can either arrange for the agent to run after the nightly design refresh or make the change to the template.

Another solution is to compromise on the user interface. For instance, instead of a view of “open requests that are overdue”, you could have “open requests by due date,” so that the overdue requests sort at the top of the view. They're almost as easy to find there, and the view opens much faster.

In some cases, it's appropriate to use a folder to display a group of documents based on date criteria. A nightly agent can populate the folder with documents appropriate for that date, and access settings on the folder can prevent users changing its contents manually. This isn't a good choice if the folder contents should change during the day as documents are edited (you can also manage this with custom coding, though it becomes cumbersome).

6.2 Unnecessary views

Many applications are slow because they contain many views, and any you can remove will help. The effect is on server performance generally, rather than on the specific application.

> **NOTE:** The designer of the database doesn't necessarily have access to see every view. Users' "Server private" views and other views with reader lists that do not include the developer are invisible, but they still affect performance. A server administrator can see these views using "Full access administration" mode.

The default view refresh settings (Auto after first use, Discard index after 45 days) mean that indexes of views that are unused for 45 days are discarded and are no longer automatically refreshed by the server. At that point their effect on performance is

minimal. However, having the views in the outline means that it's likely someone will use them occasionally by accident while searching for the right view.

You can therefore not only improve performance but also provide a less frustrating user experience by limiting views to only those that are necessary, are designed for the users' specific tasks, and are named so that users recognize the one they need without hunting for it.

Frequently, views are created for some special, one-time use, and there's no process for recording who asked for them, who's using them, and when they can be safely deleted. Often they are “Server private” views, visible only to the person who created them – but they still affect performance. Limiting access to create such views helps preserve performance (if you want to see what private views are there, a server administrator can list them using “Full access administration” mode).

We recommend using the Comment field of a view to describe the task for which it was created, who uses it, and a “sunset” date after which it can be deleted, if known. That way, if you have a question about whether a view is needed, you at least know who to ask. If you want to delete a view to see whether anyone protests, cut and paste it into another database that contains no documents, just to have a place to keep it in case you want to retrieve it.

Often, an application contains views that are only different with respect to the way they're sorted. Such views should generally be combined into a single view with re-sortable columns. Although the cost of adding a re-sort column is significant, it's still less than having a second, separate view.

This is especially true if you use the new column option in Lotus Notes 8.0, “Defer index creation until first use.” This option delays creating the index for a re-sort until a user requests it. This does cause a long delay for that first user, but if no one ever requests it, everyone enjoys better performance.

6.3 Private views

When you go looking for unneeded views, bear in mind that you as a developer can't necessarily see all the views in the application. If users have private views stored on the server, or if there are shared views with access lists that don't include you, you will be unable to see those views in Designer – but they still affect performance. A server administrator using “Full access administration” mode can bypass the access controls to get you a list of all the views (and delete any that you want to remove).

6.4 Unnecessary re-sorts

Since the server must do extra work to make alternate view sorting available immediately on request, you should enable re-sorting only when it's actually useful. Ascending and descending count as two separate re-sorts, so don't enable them both unless there's a real need for both. In Lotus Notes 8.0, if you're not sure a re-sort will be used, enable the “Defer index creation until first use” option on that column.

Note that you also have the option to make clicking a column header navigate the user to a different view that's already sorted by that column, so you can provide the convenience of a re-sort without the extra cost (if the other view already exists).

6.5 Unnecessary columns

It's tempting to create a column for every field, but don't do it. Limit the information in views to what the user actually needs to see there; the screen will be less cluttered and the whole application will be faster and use less storage.

6.6 Overly complex formulas

If you have a view column formula or selection formula that uses looping functions (@For, @While, @Transform) or is longer than, say, 200 characters excluding comments, try to simplify it. If you can't simplify it, consider moving the formula into a computed field on the form, so that the view can refer only to the field name. This is especially helpful for formulas that are used in multiple views.

Even if you don't choose the computed-field route, most long formulas can be simplified with a little thought. Consider using @Select or @Replace instead of long @If statements, and review the logic to see whether tests can be simplified by doing them in a different order.

Be aware of operators and @Functions that operate on all the members of a list. There's no need to write a loop for many simple manipulations on string lists; for instance, to get the first three characters of each element, use @Left(*listfieldname*; 3).

We also have “combinatoric” operators like *=, which can be used to compare every combination of elements from two lists and which can help you write more concise formulas.

If you've programmed in other languages, you might be accustomed to logical operators that evaluate only those expressions necessary to determine the value of the conjunction. For example, you might expect this:

Form = “Report” & (Sections = “Financials” | Total > 10000)

to first check whether Form is Report, and only if that is true, test the rest of the expression. In macro language (and in LotusScript), logical operators don't work that way. Both parts of the expression are always evaluated. So, if the second part is expensive to evaluate, you might choose to do your own “lazy logic” formula as follows:

@If(Form = “Report”; Sections = “Financials” | Total > 10000; @False)

The function @If takes longer to execute than the & operator, but if you can use it to avoid executing some expensive functions unnecessarily, you come out ahead.

6.7 Overuse of multiple categorization

Categories are excellent. They can let you list a document under multiple headings in the same view, which is quite useful. But don't go overboard, because listing a document twice in a view takes almost twice as long as listing it once. If each document is in fifty categories, multiply by the number of documents, and how many rows must the poor server calculate and store?

Even if you don't use *multiple* categories, categorized views are slower than the corresponding view with simple sorting. The time is based on the number of rows, not the number of documents, and each document and each category heading is a row.

6.8 Overzealous indexing

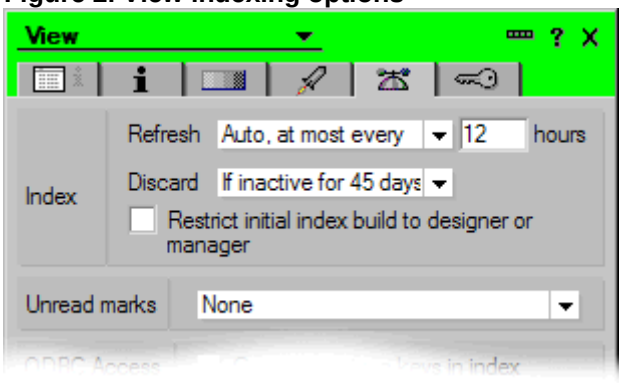
The View Properties dialog box contains a set of options that controls view indexing. These options are seldom used, but choosing an appropriate indexing option can have a big performance payoff.

For instance, suppose a database contains special keyword documents that you frequently look up to fill keyword lists on a form. The keyword documents change very rarely, but other documents in the application are modified all the time.

We already know from our discussion of @DbLookup that it's best to use cache for such a lookup, but you still must go to the view the first time when you have no cached value. When you do that, Lotus Notes notices there are documents modified since the view was last used and wastes time looking at those documents and finding that they aren't in the view.

The view that you use in your @DbLookup of keyword values doesn't need to be re-indexed every time it's used. For such a view, it makes sense to select the indexing option "Auto, at most every *x* hours," with an appropriate value for *x* (see figure 2).

Figure 2. View indexing options



Such views are still updated by the server when nobody's using them, just not as often. Occasionally, an unlucky user will "take the hit" of refreshing the index. However, the average lookup time will be much less, and it's unlikely that any single user will incur this cost for *every* lookup in a complex form, so the form will still open faster than if you didn't use this option.

If a view is used one day every quarter for a quarterly review, there's no point in keeping the index for 45 days. Set it to discard after two days, and you're giving the server less work.

There are other situations in which you can improve performance by choosing an appropriate indexing option. It's worth determining what setting is right for each of your views.

> **NOTE**: It is possible to refresh an index in the current replica programmatically, using the NotesView.Refresh method. Suppose there's an index that's normally rarely updated, but when you save a particular form that contributes data to the view, you

must have the view updated so you can use the new data in lookups right away. In the Postsave code on that form, use the Refresh method on the view. At the same time, you could use @Db functions with ReCache, to update the cache of specific lookups to that view.

6.9 Reader fields

There's no substitute for Reader fields if you need them, but they can make view performance quite slow. When you open a view that contains documents with Reader fields, Lotus Notes scans down the rows, looking for those to which you have access. It stops looking when there are enough to fill your screen. If you have access to only one document, it must look at every row in the view to determine this – and that could take a while.

There are several things you can do about this:

- Use short Reader field values. It's faster to check membership in a single role than to compare them against a long list of access names (and there are also maintainability advantages to using roles).

- Avoid using views in such an application. If users have access to only one or two documents, you can provide them access via other ways, for example, by sending them an automatic email with links to those documents.

- Use an embedded single-category view that displays only the category containing “their” documents.

- Use a categorized view that is set to display empty categories (that is, categories that contain no documents the user can view). Of course, this can also make it hard for the user to find their documents unless you navigate the user to them, so you might use this in conjunction with @SetViewInfo to display only “their” category.

  NOTE: There's a security impact to using such a categorized view; that is, you're showing the user one field – the category – in documents to which they otherwise have no access. Make sure this is OK.

- Encourage people to use local replicas. Because the local replica contains only the documents to which they do have access, it's no extra work to eliminate those they can't view.

Don't use Reader fields solely as a navigational aid; for instance, as a means to make it easy for users to find “their” documents because they're the only ones in the view that they can see. If the information in the document isn't actually private, there are better ways to help the user find the correct ones, as described above and in the next section.

6.10 Private on first use

@UserName and @UserRoles don't yield the desired results when used in the selection or column formulas of a shared view. This is the most common reason developers create “Private on first use” views to show only “My Documents”. These views are either stored on the server, in which case they affect the performance of the application generally, or they're stored in the user's local “desktop” file.

Desktop views don't affect the server's performance directly, but when someone opens one, just like other views, it's re-indexed to show the latest changes. That means the user's workstation must request from the server all documents modified since the last use of the view. This can take time while the user is waiting and, if many users are doing it, can bog down the server with numerous requests to “send all the data.”

Note that view indexing uses only the *summary* data, so large rich text fields and file attachments are not an issue here.

In addition to performance issues, private views present a maintenance challenge because there's no easy way for the developer to update the design of users' private copies. Even shared columns don't work in this context because, to update a shared column in a view, the person doing the update must have access to the view.

Often, you can avoid Private on first use by using the “single category” capability of Notes views. If you are showing “My Documents,” for instance, you can use a view that categorizes documents by owner, then either embed the view in a form or page, with a “single category” formula, or use @SetViewInfo in the view Postopen event to limit display to the current user. Because there's only one shared view, the indexing cost is minimized overall, and the individual user doesn't need to wait like they would for a desktop private view because the index is always relatively up to date.

7 Code

Once you start writing LotusScript or Java™ code, you open up a whole new world of opportunities to create slow performance. Here we discuss some common pitfalls.

7.1 GetNthDocument

Using NotesDocumentCollection.GetNthDocument to iterate through the collection is very slow; instead, use GetFirstDocument and GetNextDocument. There are certain types of collections for which GetNthDocument is just as efficient, but it's easier just not to use it.

7.2 Too much action code on a form or view

If you have many actions on your form, view, or folder, and you write the code for each action in the design element (or even if you use a shared action), you're storing a lot of code that must be loaded into memory each time the design element is used.

Most of the time, only one or two of your many actions will be used, so you're wasting time loading it all. If the action appears in multiple places, you're caching that same code multiple times in your design cache, using up memory that you might require for other purposes.

Consider moving some action code into agents. This lets you have just one copy of the code that is loaded into memory only when someone asks to run it. The action button can be written in macro language to call the agent by use of @Command([RunAgent]), so very little code must be loaded along with the design element.

This is especially important if you let users create private views or folders because that action code will be duplicated many times in their folders, taking up space, and can't be updated unless the user manually deletes their private view.

7.3 Too many script libraries

The time required to load multiple script libraries in the same script is greater than linear. That is, loading ten script libraries takes much longer than twice the amount of time to load five libraries, particularly if libraries “Use” other libraries.

This might change in the future, but even so, there's a break-even point; it takes longer to access two design elements than to access one design element containing the same total amount of code. Combining script libraries that are often used together saves loading time, even though in some cases you may be including code you won't call in the specific agent.

7.4 ComputeWithForm

The ComputeWithForm method of NotesDocument is an easy way to update computed fields in a document without having to duplicate code. Unfortunately, it's also rather slow compared with “manually” calculating and assigning the new field value. If your agent is slow and you're using ComputeWithForm, you can probably speed it up considerably by taking that call out and putting in a few lines of code to assign specific fields.

7.5 Auto-update views

By default, when you use a NotesView object, it implements the normal index refresh attributes of the view. For example, suppose you're updating a collection of “Vegetable” documents, and as part of the processing you must look up the pests for that vegetable in the “Pests” view of the same database. But when you save a Vegetable document, now a document has been modified.

As you process the next document and do a lookup to the “Pests” view, Lotus Notes notices that the view index is out of date and refreshes it. You know that the change you made doesn't affect the Pests view, but Lotus Notes doesn't know this until it tests the changed document.

It makes sense in this case to use the AutoUpdate property of NotesView to tell Lotus Notes not to bother updating the view index, unless you explicitly request it by using the Refresh method. This makes a substantial speed difference.

You can also use this method even if the changes you're making do affect the contents of the NotesView, provided you know the changes won't matter to whatever you're doing. For instance, you know that your update will remove the document from the view, but it doesn't matter because you're on to the next document.

7.6 Failure to use efficient collection-based methods

The NotesDocumentCollection class has a few methods whose names end in “All,” which give you a way to do things to all documents in the collection. Familiarize yourself with these methods, because they are a lot faster than iterating through the collection and operating on each document individually. (Unless, of course, you need to do multiple things to each document; then it's probably faster to iterate so that you save each document only once.)

7.7 Repeating expensive operations

Certain methods and properties in the built-in classes are fairly slow. Your code will run faster if you avoid using these functions repeatedly when you don't need to. For

instance, suppose that as you process a collection of documents, for each document you must use one of its fields as a lookup value to pull information from another view:

```
Dim view As NotesView
Set docCur = coll.GetFirstDocument
Do Until docCur Is Nothing
        Set view = db.GetView("CustomersByID") ' oops! Don't do this in the loop!
        Set docCust = view.GetDocumentByKey(docCur.CustID(0), True)
...
        Set docCur = coll.GetNextDocument(docCur)
Loop
```

In this code sample, if **coll** contained 1000 documents, we called the expensive GetView method 1000 times. The code will be much faster if we just swap the positions of the **Do Until** and **Set view** lines, so that GetView is called only once.

The agent profiler is a good way of finding things like this. It is described in the developerWorks Lotus articles, Troubleshooting application performance: Part 1: Troubleshooting techniques and code tips, and Troubleshooting application performance: Part 2: New tools in Lotus Notes/Domino 7.

7.8 Saving documents that have not changed

Recall that one of the factors affecting performance is "churn", that is, how often documents are modified. When you write agents to process documents, try to avoid saving changes to the document unnecessarily. Before you assign an item, check whether the item already has that value. If you don't end up changing anything, don't call the Save method. Often, you can use the search methods to filter out documents from your collection that you don't need to process.

The agent might take a little longer to run if you're constantly checking items to determine whether they need to be changed. Or, it might not, because it takes much longer to save a document than to compare information in memory. But in any case, other parts of your application will work more efficiently, from replication to view indexing to full-text indexing.

Avoiding unnecessary saves also reduces the chance of causing replication conflicts. Replication uses *item* modification times; it doesn't send an entire document to the other replica, only those items that have changed. So even if you must save a document anyway, it saves you time on replication if you have modified only those items that needed to have a new value. Your users with local replicas will be grateful.

7.9 Ways of searching for documents

One thing most agents must do is locate a set of documents to process. There are different ways of doing this, each of which is appropriate for different situations.

The developerWorks Lotus article, Lotus Notes/Domino 7 application performance: Part 1: Database properties and document collections, discusses different ways of searching for and processing collections of documents. In summary:

- If you have a view that contains the documents you want, sorted in a useful way, it's usually fastest to read the documents from the view, for example, by using the

GetAllDocumentsByKey method.

- For databases containing large numbers of documents, the FTSearch methods are faster than NotesDatabase.Search, provided your database is full-text indexed. Note that you can also do a full-text search by entering it in the Document Selection event of an agent.

In both these cases, you're saving time by taking advantage of indexing work that's already been done by the server in advance. So there's less work to do at runtime, compared with NotesDatabase.Search, which must test every individual document.

The full-text search doesn't let you filter documents to quite the level of detail that NotesDatabase.Search does, but it usually saves enough time that you can afford to iterate through the search results and skip over the few that don't apply. Learn the complete full-text query syntax, which is documented in the Notes Client help (not the Designer help), under the heading “Refining a search query using operators.” You may find you can do much more with it than you thought.

> **NOTE**: Depending on your requirements, you can sometimes combine the power of formula search and the performance of full-text search by doing a full-text search in a view, which has formula-based selection criteria.

7.10 Deleting unused documents from cache

In earlier versions of Lotus Notes, you could get better memory usage by deleting NotesDocument objects from memory when you were done with them, using the Delete statement. However, with versions 6.0 and later, this is no longer worth doing. (If you know what this is referring to, don't bother doing it anymore. If you don't know, don't worry about it.)

You might still use Delete for reasons other than performance, for example, because you think the document will have been modified by another process since you last opened it and you want to ensure you're reading the latest data.

7.11 More efficient loops, assignments, etc.

Some of the literature offers comparative timing for “for” versus “while” loops, global versus stack variables, and so on. However, unless your application is especially compute-intensive, you're unlikely to see much performance improvement from these things.

Most scripts spend far more time opening documents and views than manipulating variable values. Avoiding an unnecessary array reference might save you a millionth of a second; whereas, an unnecessary view open may be more on the order of whole seconds. If you're going to put time into improving performance, begin with the big-payoff items.

It can be useful to know about performance characteristics of different LotusScript expressions and statements, but it’s more valuable to build good habits when writing code originally; it rarely pays to go back later and fix inefficient assignments.

The only tips in this area we consider worthwhile are:

- Don't use GetNthDocument (as discussed above)
- Declare variables explicitly, to avoid the default Variant type. This not only provides better performance, but also helps you find errors sooner – at compile time. Use the option in the programming pane properties that has the Option Declare statement inserted automatically in your LotusScript code.

7.12 Using the LC LSX

If you're integrating with outside relational databases or data files, the LC LSX API is typically faster than the built-in ODBC classes. The IBM Redbooks publication, Implementing IBM Lotus Enterprise Integrator 6, contains much information about how to program with this API and how to maximize its performance.

8 Testing

As mentioned at the start of this paper, many nice little applications that work great with a small data set and a few users rolls over and plays dead when you give it many documents and many users. It makes sense to test your design with large numbers of documents, and test what happens when 50 people use it at once (in case you don't happen to have 50 friends with time on their hands, there are automated testing tools available that can simulate this situation).

Also, it’s not difficult to write agents to take a small set of sample data and multiply it into many thousands of documents by assigning random values to selected fields. This can even be done with formula agents, if you use the agent option to create new documents (in the lower right-hand portion of the agent editing screen).

A word of warning, though: If you're testing an application that's already in production, do your testing in a copy of the database (*not* a replica) on a non-production server, and preferably one that doesn't replicate with the production servers. That way, you run no risk of contaminating the production data, nor will you crash or bog down the server people are using to get their work done.

9 Use Profile documents

Profile documents are a good way to efficiently store and retrieve information that doesn't change often. Because the entire document is cached the first time it's used, it's quite efficient to store all your customizable keyword lists there. There are no issues of view indexing, and no need to worry about controlling caching. They replicate just like regular documents (except that users with replication selection formulas can't accidentally break your application by excluding them, so they are better than “regular” keyword documents in that way also). Just use them. They're fun and easy.

10 Conclusion

This white paper, over-long though it may be, is not comprehensive. The Resources section below provides additional useful information and techniques. It’s also advisable to keep up with the various Lotus-oriented blogs and wikis, which frequently have performance-related tips.

11 Resources

- The IBM Redbooks publication, Performance Considerations for Domino Applications (March 2000) is the canonical resource for performance issues. It's somewhat dated but is still quite relevant.

- The developerWorks Lotus article, Lotus Notes/Domino 7 application performance: Part 1: Database properties and document collections, discusses different ways of searching for and processing collections of documents.

- The developerWorks Lotus article, Lotus Notes/Domino 7 application performance: Part 2: Optimizing database views, examines indexing times for different types of views and discusses how to reduce the performance impact of Reader fields.

- The developerWorks Lotus articles, Troubleshooting application performance: Part 1: Troubleshooting techniques and code tips and Part 2: New tools in Lotus Notes/Domino 7, discuss how to determine what part of an application is causing a slowdown.

- The developerWorks Lotus article, Application Performance Tuning, Part 1, goes into greater detail on database properties affecting performance, and how to cause view indexing times to be logged.

- The developerWorks Lotus article, Application Performance Tuning, Part 2, is primarily about increasing performance by avoiding unnecessary computation.

- The IBM Support Web site Techdoc, Time/Date views in Notes: What are the options?, provides alternate ways of creating date-based views.

- The IBM Business Partner piece, Performance Engineering Notes/Domino Applications, covers some of the same material covered here, with some comparative measurements of exact effects of more documents and more fields. Some tips specifically for Web applications are also included.

- This Lotus Sandbox download contains an agent called Replicate Customers, which demonstrates an algorithm for doing one-way synchronization without deleting and re-creating every document.

- The developerWorks Lotus article, Generating sequential numbers in replicated applications, describes one technique for getting a sequential number across multiple replicas.

12 About the author

Andre Guirard is a member of the development team for the Domino Designer. A long-time Notes developer, Andre now focuses on developer enablement. He has written many technical articles for developerWorks, The View, and other publications; is an IBM Redbooks contributor; and owns the developerWorks blog, Best Practice Makes Perfect. He often speaks at Lotusphere and other IBM conferences. In his free time, he is the Labor Pool for his wife's gardening projects, and writes science fiction and fantasy stories.

13 Acknowledgements

The author extends his thanks to John Curtis and the business partners who helped technically review this white paper.