Lotus Developer Domain
The Technical Resource for Lotus software
LDD Today

Level: Intermediate
Works with: Notes/Domino
Updated: 01-Apr-2003

by
Tara Hall
with Raphael Savir

Application performance is a measure of how efficient your application runs in certain environments and under specific loads. Can you measure application performance? Sure, all it takes is an isolated test environment that includes a network similar to your production environment, load testing software to simulate users and their tasks, and a lot of time. Unlike server performance testing where you can easily eliminate variables such as CPU, RAM, NICs, and so on, application performance testing involves meticulous testing of one field on one form in one view—at a time. Given the complexities of some custom Notes applications, this level of testing is not only tedious but seemingly unending. Who knows how long it could take for you to narrow down the one design element, formula, script, or property that could be preventing your application from performing at its optimum?

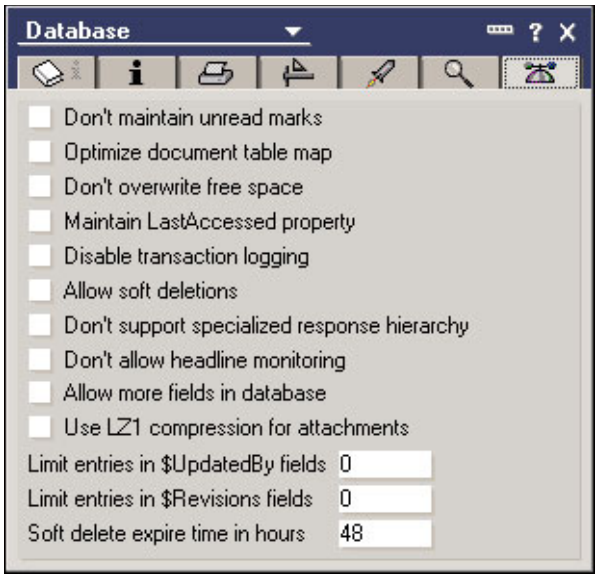
But there is an easier way, as we explain in this article. Based on several years of experience evaluating custom Notes applications to diagnose performance-related issues, we've compiled the most common properties that can affect application performance. In this first of a series of articles, we cover the database, view, and form properties known to affect application performance. We tell you when to use and when *not* to use certain properties for best performance results, and where applicable, we provide you with alternative solutions. This article assumes that you are an experienced Notes application developer.

Database properties

Database properties are often overlooked when an application is rolled into production. But the fact is that by enabling and disabling certain properties you may realize a performance savings with no loss in functionality, in development time, or in administrative resources. We'll look at the following common database properties that affect performance:

- Don't maintain unread marks
- Don't overwrite free space
- Maintain LastAccessed property
- Don't support specialized response hierarchy
- Web access: Require SSL connection

You can find all these properties on the Database Properties dialog box. The first four are on the Advanced tab and the last property is on the Database Basics tab.



Don't maintain unread marks
Admittedly, this property is a confusing one because it reads like a double negative, but by default, all read and unread documents are tracked in a database. This can be useful in a discussion forum where users want to see which topics and responses are new and unread. However, tracking read and unread documents has an impact on application performance. For example, suppose you have a knowledge database with 1,000,000 documents. The database is accessed by 10,000 users, many of whom replicate the database locally using a selective replication formula. When a user replicates, he experiences an initial delay as the local and server replica copies synchronize their Unread Marks tables. This process may take just as long as the actual data replication. This means long delays for your users when they replicate. Likewise, users who access the database on the server also experience a delay when they initially open the database because the program has to read through the Unread Marks table to determine which documents to display as read/unread for this user. The delay may only be a few seconds, but in a user's mind, it can count as one strike against your application.

To disable this option, select the Don't maintain unread marks option on the Advanced tab of the Database Properties box. In Release 5 and Notes/Domino 6, this option affects the entire database, not just a particular view.

Don't overwrite free space
In Notes Release 3 and earlier releases, Notes held deleted data—unencrypted—until empty space, or white space, was removed during a database compact. In Release 4, this functionality was changed subtly so that the deleted data was overwritten with random characters to make it irretrievable. (This is called overwriting free space.) In Release 5 and in Notes/Domino 6, you can choose to enable/disable this functionality. Overwriting free space has an adverse affect on database performance.

To help you understand this feature, let's take, for example, documents deleted from your desktop PC. When you delete a document on your Windows operating system, it goes directly to the recycle bin. Then you can empty the recycle bin, and it appears that the document is gone for good. But let's say that after emptying the recycle bin, you realize that you need that document after all. Is the document lost forever? Not necessarily—it may no longer reside in your recycle bin, but it still exists on the computer. With the help of appropriate software (for instance Norton Utilities) you can retrieve the deleted document.

So as a security measure, when you delete a Notes document, Notes overwrites the deleted data to prevent anyone from retrieving it. When you press F9 or choose View - Refresh, the document is removed. Imagine that your Notes document went from:

The quick brown fox jumped over the lazy dog

to:

XX XXXXXXXXXXXXX XXXXXXXXXXX XX X XXXXXXXXXX

Note: This example does not accurately illustrate how Notes overwrites deleted data.

At this point, it is irrelevant whether or not someone can retrieve the document because the data itself is destroyed. Note that when you perform a soft delete of a document, Notes does not overwrite the data. Only a hard delete activates the overwrite.

In most cases, there's no need to keep overwritten data. There are, however, a few instances when you may want Notes to continue overwriting deleted data:

- Physical access to your servers and your databases may be compromised allowing an unauthorized user to access them.
- The databases are not encrypted or the ACL leaves the database open to attack.
- Your organization has a security policy that requires this feature.

If none of these cases applies to your organization, servers, or databases, then consider disabling this property by selecting the Don't overwrite free space option.

Maintain LastAccessed property
The Maintain LastAccessed property was introduced in Release 4; it tracks the date when a document was last accessed (that is, the last time a document was read or modified). By default, databases only track when a document was last modified, but by selecting the Maintain LastAccessed property option, the database can also track when a document was last read. Naturally, for optimal application performance, you want to keep this option deselected.

However, this option may have value for anyone archiving documents. For instance, returning to our example knowledge database containing 1,000,000 documents, imagine that 1,000 new documents are added to the database each day. With so many documents being added, we find it necessary to archive old ones. We can use the Maintain LastAccessed property option to find out when documents were last accessed to determine which documents to archive. We may set up our archiving feature to archive any document that was not opened or read in the last 18 months based on the LastAccessed property.

You may want to use this option to help archive documents in any database in which documents expire, become outdated, or have a short life cycle, for instance, a discussion database or possibly a workflow database. You may not, however, want to use this option in a database in which documents are accessed infrequently or there is no last accessed date to track, for example, in a help desk application.

One more thing to keep in mind: the LastAccessed property is not applicable to Web applications. This property ignores Web browser reads.

Don't support specialized response hierarchy
The Don't support specialized response hierarchy option allows your application to take advantage of the specialized response @formulas: @AllChildren and @AllDescendents. These functions allow you to build views based on specific criteria for parent documents and all of their response documents. Let's take for example a discussion database with 10,000 main topics and 100,000 response documents. Suppose you create a handful of views that show just certain categories, like Application Performance. If only 100 main topics are categorized this way, you may anticipate that the view would show those 100 main topics, plus all of their responses (and response to response documents and so on). Traditionally, Notes has relied upon a Selection formula such as:

```
SELECT (Form = "Main Topic" & Categories = "Application Development") | @IsResponseDoc
```

Unfortunately, this formula gives you all 100,000 response documents. You presumably wouldn't see most of them because your view will have a response hierarchy. But they are all there, slowing down your view indexing and taking up disk space. If you allow a specialized response hierarchy (enabled in Release 4, and optional in Release 5 and Notes/Domino 6), then you can use a slightly different formula:

```
SELECT (Form = "Main Topic" & Categories = "Application Development") | @AllDescendants
```

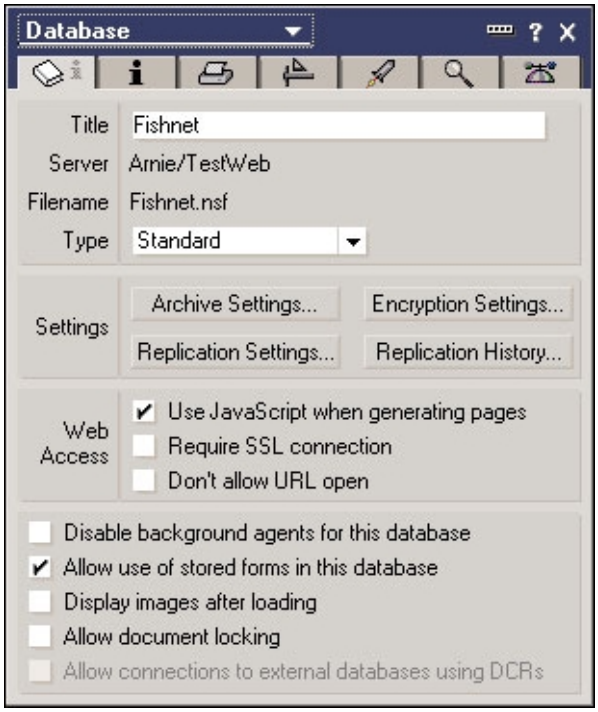
This formula gives you the exact set of documents you seek, so your view indexing and disk space requirements are minimized.

So far, we've only told you why you should enable this feature (that is, leave the option unchecked). But if your application has no use for formulas that use @AllChildren or @AllDescendants, then there is no reason for the program to maintain this information, so you can save processing cycles by selecting the Don't support specialized

response hierarchy option to disable the specialized response hierarchy.

Web Access: Require SSL connection
The Web Access: Require SSL connection option enables an SSL (secure socket layer) connection for each Web transaction, so all data transmitted between the client and server is encrypted. After you enable SSL, you and your users will experience a performance degradation of approximately 10 percent. However, the architecture of each individual application can affect that percentage. When SSL is enabled, each packet of information is encrypted, so an application that requires many transmissions back and forth between client and server requires many encryptions.

Here's an example: suppose you have a form which uses SSL to encrypt all form data. The form contains an @DbLookup formula. SSL encrypts each transaction between the client and server, so in addition to encrypting the form data, SSL also encrypts the lookup transaction.



Sometimes enabling SSL for your applications is unavoidable. For instance, it may be your organization's policy to run SSL on specific applications. Other times, enabling SSL is warranted under certain circumstances, such as having a client in one country and a server located in another country. If you can't be certain whether or not you can trust a network, then enabling SSL makes sense. If, however, you have a trusted network—one which you don't think can be compromised—then SSL isn't needed for your applications and can spare you and your users the performance degradation.

Create private views/folders
Create private views/folders is an ACL option. Users who are granted this privilege can create private views and folders and store them on the server that hosts the database. Creating many views, especially large ones, can become a performance issue due to the extra indexing required. Also note that private views and folders stored on the server are difficult for administrators or developers to delete.

The compact and updall tasks
While compacting your applications and running the updall task to refresh view indexes are good database practices, they will generally not improve your application performance. There are exceptions, such as a database with a very large percentage of white space, but running compact against a database with 5 or 10 percent white space will not realize any noticeable performance gains.

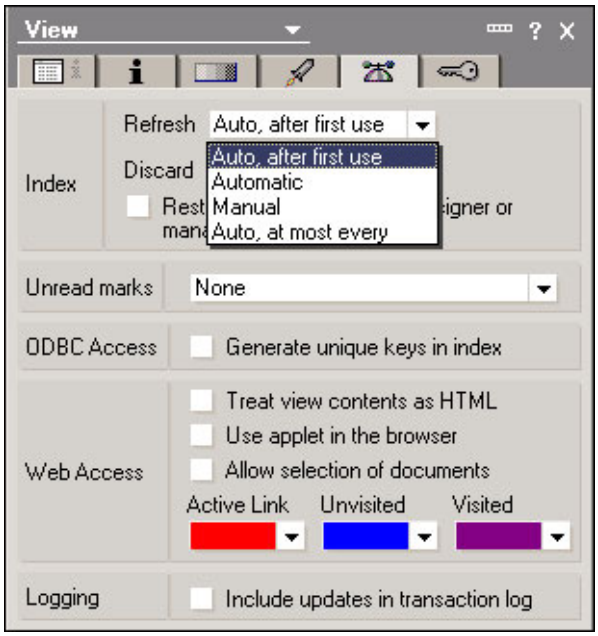
View properties
There are several key areas that affect view performance:

- Time/date sensitive formulas (selection or column formulas)
- Column sort on the fly
- Reader name fields
- ODBC access

Time/date sensitive views
A time/date sensitive view is one that contains a column with a time/date formula, such as @Modified or @Now, or with a selection formula that has a time/date formula. These views can offer great functionality, but they can also be expensive performance-wise. Every 15 minutes, the Domino server runs the update task to refresh all view indexes in a database. This task is non-configurable (that is, you cannot configure the task to run on a different schedule). Suppose at 9:00 AM the task runs to refresh all views in a database. At 9:02, a document is modified in the database. At 9:15, the update task runs again and notices that a document has been modified in this database. It may have been mailed in, replicated, created, updated, deleted, and so on, but regardless, a document modification took place since the last time the update task ran, so the database must be checked for out-of-date views.

At this point, all views that may contain the modified document(s) have been flagged as out-of-date. And all time/date sensitive views are flagged as being out-of-date. So, in addition to updating the views that you could reasonably suppose need updating, the indexer also needs to update all the time/date sensitive views. But it gets worse. These views cannot be refreshed; they must be rebuilt, which takes much longer. To get an idea of how much longer, if you run some sensitive diagnostics, you will find that typical views take maybe 100 milliseconds to refresh, but typical time/date sensitive views take 10-50 *seconds* (not milliseconds) to refresh because they are really being rebuilt. On a busy server with many databases to check and many views to update every 15 minutes, your environment cannot afford to spend this kind of time on a single view.

Also note that setting the View Properties box Refresh field to Manual (as opposed to Automatic or Automatic, at most every) opens the view quickly for users without affecting the 15 minute Update task.



If the view refresh is not set to Manual, then every time a user opens the view, it forces a rebuild. Of course, using this setting means that when users open the view, it is not necessarily up-to-date, so you have to weigh the performance advantages with the potential disadvantage of out-of-date data.

Time/date sensitive views are a very useful (but extremely expensive) option, so use them with care.

Checking the log file
If you're not sure how long your views are taking to refresh or rebuild, use the log file. Set Log_update=2 in your Server Configuration document to record when the Indexer refreshes/rebuilds views. It lists each view in each database that it refreshes/rebuilds, enabling you to track how much time is being spent on a particular view or

database. With a little practice, you can become quite fast at spotting any problems on your servers.

Time/date alternatives
There are alternatives that you can implement for time/date sensitive views:

- Use @Text in the Selection formula, as described in **Time/Date Views in Notes: What Are the Options?**. This alternative has limitations, but can be very useful and is a good performer.
- Run an agent to mark documents that should appear in the ex-time/date sensitive view. For example, if you have a view that shows documents with Status = "Open" and whose DueDate is before today, you can have an agent run nightly to mark such documents with OverDueFlag = "Yes." Then your view selection formula need only check for that flag. When documents are closed, that flag can be erased. Although this is a great solution for a single-server database, it is not a good solution if your users typically use local replicas because they have to replicate the agent's data changes every day. Similarly, you may not want to use this if your database replicates to dozens of servers worldwide.
- Create a view that categorizes documents by date. This solution is so simple that it is often overlooked. Using the example above, make a view of all Status = "Open" documents categorized on their DueDate. Users can easily examine overdue documents, or documents which are near due, simply by scrolling to the correct date category.

Column sorting
Column sorting is an option that allows users to sort columns in a view ascending, descending, or both. Each sorting arrow on a column in a view increases the view index and the time it takes to refresh the view. From a user interface and maintenance perspective, it's better to have a single view with a handful of sorting arrows than multiple views. From a performance perspective, however, the key is to avoid adding many sorting arrows to a view without reducing the overall number of views. To give some specific metrics, you will find that adding two sorting arrows to a view increases the index size/time approximately three-fold. Four sorting arrows increases the view index size/time five-fold, and so on.

Tip: Check your hidden views that are only used for lookups. Sorting arrows are not used in these views, so they should be eliminated if any exist.

Reader name fields
Reader name fields are not view properties, but they can affect view performance. Reader name fields can provide document security, but can have adverse affect on views that display documents that contain reader name fields. For example, suppose you have a Human Resource database for your 10,000 employees and suppose there is a flat view (not categorized) that displays all 10,000 documents. Every employee can see only his/her own document due to Reader Names restrictions. When an employee opens the database, the Domino server sorts through the documents to determine which ones should be shown to the user. Normally, without Reader Names, the first 30 or so documents are displayed to the user, filling the screen. You can do a quick test to confirm this in any database by opening a view and then pressing the down arrow. You scroll quickly for a dozen or so rows; then there is a pause, usually less than a second in duration, while the server grabs another chunk of data to be displayed. The scrolling recommences for 30 or so rows, then another pause occurs, and so on.

In the HR application, however, the server is unable to fill the user's screen with data. That is not a problem, but the server has no way of knowing that it can't fill the screen, so it continues to sort through all 10,000 documents, looking for more documents that are visible to the user. Finally, exhausting all documents, the server gives up and shows just the one document, but the delay may be considerable. In an isolated environment, the delay might be seconds, but in a real-world production environment, with many users trying to perform a similar task, you can easily have most users timing out because of the delay.

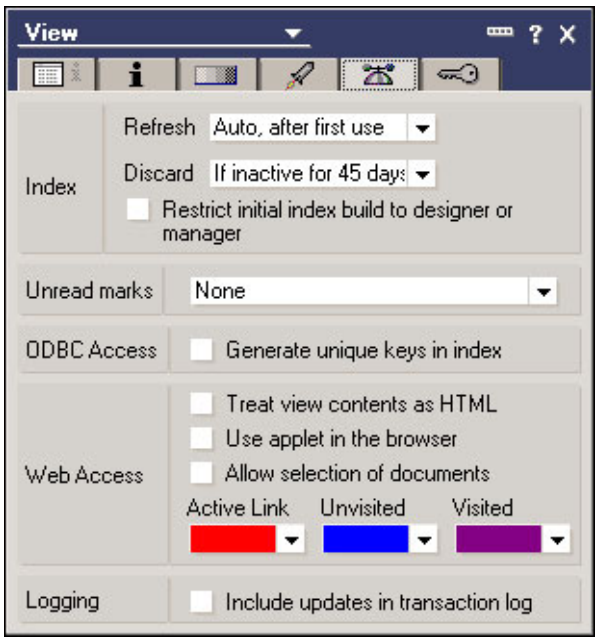
There are a few alternatives that can help you avoid this problem:

- Categorize the view, and initially collapse the categories in the view by selecting the "Collapse all when database is first opened" option. Although it is true that this setting carries a slight performance hit, in this situation it produces a great saving.
- Avoid selecting the "Don't show categories having zero documents" option to show only categories containing documents. Selecting this option has the same result as using a flat view: The server has to check all 10,000 documents before displaying anything to the user.
- Use an embedded view that displays a single category. Use an @UserName formula to display only the documents for that user. This is a fantastic performer, and it works brilliantly for Web browsers, too!

Note that the performance issues associated with reader name fields in a view are the same regardless of whether your users have Web browsers or Notes clients.

ODBC Access

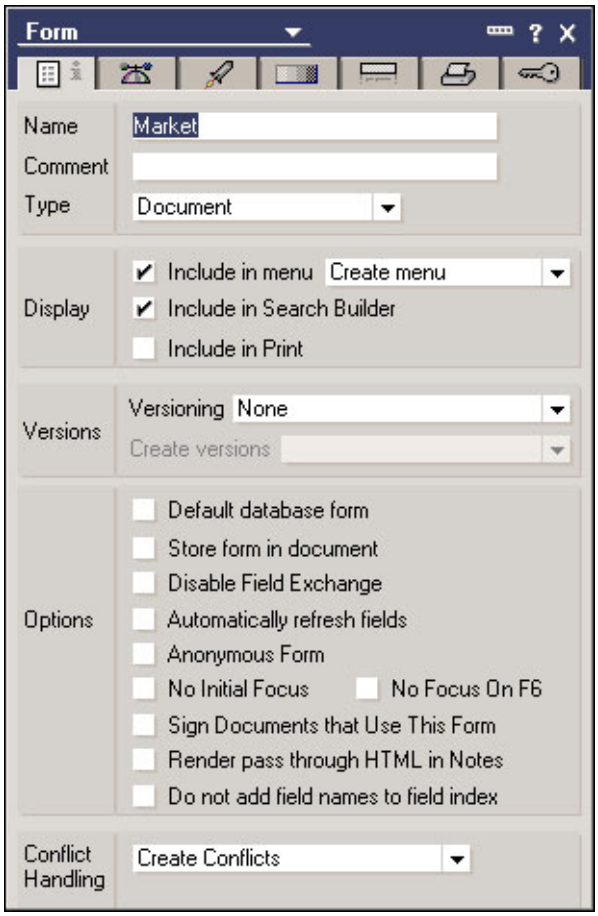
The ODBC Access property "Generate unique keys in index" provides a unique key for database interactivity. You can use this property for faster lookups. Here's how: Suppose you have a discussion database with 10,000 main topics and 100,000 response documents. The database contains a hidden view that lists only the 10,000 main topics for the purpose of looking up all the existing categories. If you select the ODBC access property for the hidden view, you can reduce the number of documents to one for each category. This property eliminates all duplicate documents. Using @DbLookup on the hidden view produces faster lookups because it dramatically reduces the view size. For example, if there are only 50 unique categories in the database, this hidden lookup view would contain only 50 documents instead of 10,000.



Note that if your documents have multiple categories selected, you need to use additional code in the view column formula to ensure that all categories are displayed and can be looked up.

Form properties

There is really only one form property that affects application performance: Automatically refresh fields. When you enable auto-refresh on a form, each time you move from one field to the next, auto-refresh updates all previous fields on the form.



For instance, when you move into the second field on a form, auto-refresh updates the first field on the form, when you move to the third field, it updates the first and second fields, and so on. This feature eventually takes its toll on application performance, particularly if your form uses workflow and lookups. For simple forms containing validation formulas, auto-fresh has far less of a performance impact. For more complex forms, consider using exit events instead of auto-refresh.

Conclusion

In this article, we examined some of the most common properties to affect application performance; however, custom applications can have a unique combination of factors that can impact performance. While you may experience improved performance enabling or disabling the properties described in this article, you may still find it useful to test your applications to see if further improvements can be made—just keep in mind that application performance testing can be difficult and time consuming. In the next article in this series, we look at agents and code that can influence application performance.

