Domino/Notes-Performance-Tipps II

In einem der vorhergehenden Artikel haben wir die administrativen Einstellungen in Domino und Notes betrachtet, die bei herkömmlichen Installationen ohne Programmieranpassungen zum Teil deutliche Performance-Verbesserungen zur Folge haben. Neben den administrativen Einstellungen sind es vielfach die programm- oder programmierspezifischen Einstellungen oder auch die Methoden der Anwendungsentwicklung, die im Hinblick auf die Performance der gesamten Domino-Installation oder einzelner Teilbereiche zum Teil eklatante Auswirkungen haben können. Diese wollen wir in dem nachfolgenden Abschnitt näher betrachten und einige wesentliche Aspekte in der Notes-Entwicklung aufzeigen, die in der Praxis immer wieder vorkommen und deren Auswirkungen meist unterschätzt werden.

Ein wichtiges Element in der Anwendungsentwicklung von Domino sind die temporären Variablen.
Neben dem Einsatz der temporären Variablen für die Zwischenspeicherung von unterschiedlichen Ergebnissen während der Ausführung eines Programms sind diese eine einfache Möglichkeit, die Ergebnisse komplexer Datenzugriffe (und hierunter verstehen wir auch den „einfachen" @DbLookup auf externe Daten oder andere Notes-Datenbanken), die in den meisten Fällen äußerst performance-intensiv sind, zwischenzuspeichern, um weitere unnötige Zugriffe auf dieselben Daten zu verhindern.

Es gilt deshalb der Rat:

Verwenden Sie in Notes-Programmen für die Verarbeitung von externen Informationen Zwischenspeicher in Form von temporären Variablen, damit Mehrfachzugriffe auf dieselben Informationen vermieden werden.

Übrigens:

Temporäre Variablen eignen sich hervorragend für die Sortierung von Ansichten. Die meisten Anwendungen verfügen über die Möglichkeit, die Sortierung innerhalb der Ansichten durch einen Klick auf die Spalten zu sortieren. Neben den Standardfunktionen gibt es immer wieder individuell realisierte Sortiermechanismen, die beispielsweise durch eine Schaltfläche die Strukturierung der Informationen zur Folge haben. Auch für solche Sortierfunktionen eignen sich die temporären Variablen. Hier können bei großen Datenmengen Verarbeitungszeiten zum Teil extrem verkürzt werden.

Notes bietet in der Anwendungsentwicklung die Möglichkeit der Verwendung von berechneten Feldern. Es versteht sich von selbst, dass die Anzahl der berechneten Felder in einer Maske sich auf die Ausführungsgeschwindigkeit der Anwendung auswirkt. Je nach Art der Verwendung eines Dokumentes, gemeint ist hier, ob ein Dokument zum Lesen geöffnet wird oder ob auf dieses im Verwaltungsmode zugegriffen wird, können unterschiedliche Anforderungen an die Berechnung von Feldern gestellt werden.

Vielfach ist die Verwendung von berechneten Feldern in Masken, die zur Anzeige geöffnet werden, unnötig. Dennoch werden diese Felder berechnet – und dies zum Teil mit recht komplexen Formeln. Macht das Berechnen der einzelnen Felder in jedem Fall Sinn, wenn das Dokument nur zum Lesen geöffnet ist? Nein? Dann empfiehlt es sich, die Berechnung des Feldes von der Art der Öffnung des Dokumentes abhängig zu machen. Notes bietet dazu die Formel @IsDocBeingEdited. In dem nachfolgenden Beispiel wird die Berechnung eines Feldes davon abhängig gemacht, ob das Dokument zum Lesen oder zum Editieren geöffnet ist:

```
@If(@IsDocBeingEdited; @DbColumn("Test"; ""; ViewName; 1); kList)
```

Ist das Dokument zum Lesen geöffnet, dann entfällt die Ausführung der @DbColumn-Anweisung – diese wird nur im Editier-Modus ausgeführt.

Durch diese einfache Lösung können nicht selten spürbare Performance-Verbesserungen erzielt werden!

Allerdings:

Bei der Verwendung dieses Verfahrens sollten Sie unbedingt darauf achten, dass beim Umschalten aus dem Anzeigemodus in den Verwaltungsmodus das Berechnen der Felder durchgeführt wird, die nach dem Beispiel von zuvor bedingt wurden. Dies kann zum Beispiel durch das Einfügen eines PostModeChange erfolgen, über den ein erneutes Laden des Dokumentes initiiert wird. Zusätzlich kann das Schlüsselwort „Felder bei Schlüsselwortänderung aktualisieren" auf Feldebene aktiviert werden.

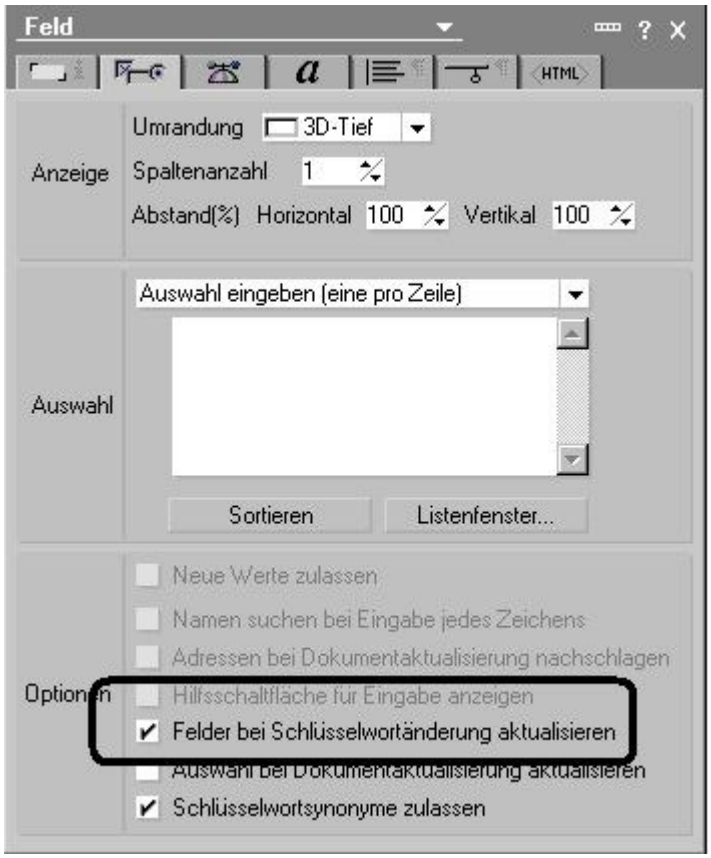

Abb.001: Feldaktualisierung

Ein weiterer Parameter, der im Bereich der Anwendungsentwicklung einen entscheidenden Einfluss auf die Performance hat, ist das Schlüsselwort „Felder automatisch aktualisieren", das sich auf Maskenebene befindet.

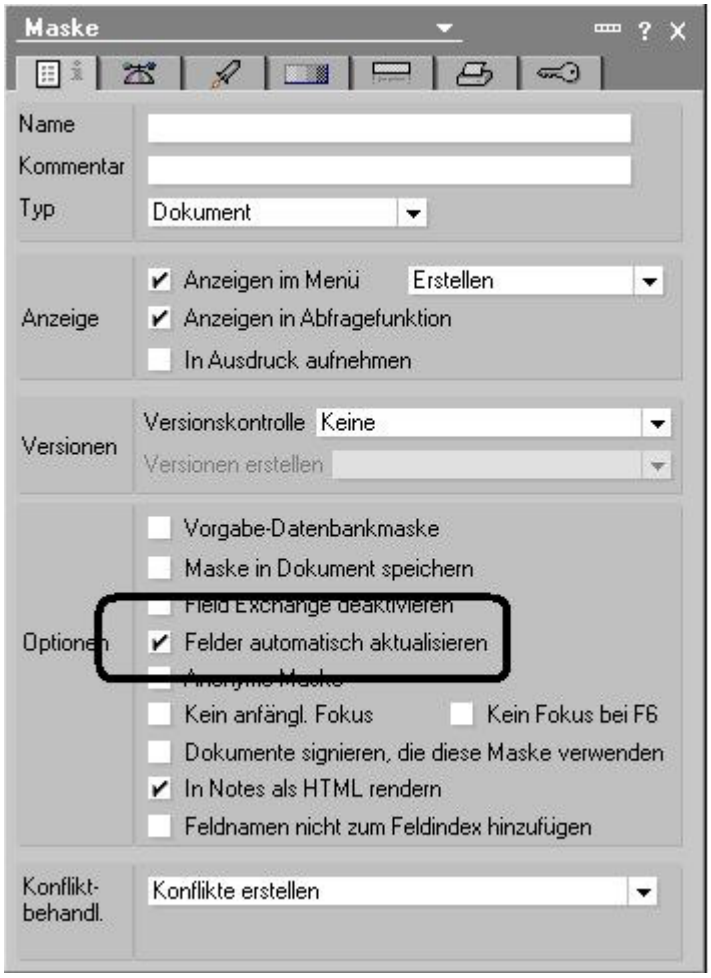

Abb.002: Felder automatisch aktualisieren

Mit dieser Option wird festgelegt, dass alle Felder, die sich in dieser Maske befinden, bei jeder Neupositionierung des Cursors – d.h. auch bei jedem Verlassen eines Feldes – automatisch berechnet werden. Diese Option ist sicher sinnvoll, wenn Eingabeübersetzungen oder Eingabe-Validierungen durchgeführt werden. Meist wird die Bedeutung dieser Option allerdings unterschätzt. Befinden sich in einer Maske beispielsweise viele zu berechnende Felder, die allerdings aktionsabhängig selektiv berechnet werden müssen, dann ist die Option „Felder automatisch aktualisieren" auf Maskenebene weniger sinnvoll als der Einsatz des zuvor beschriebenen Verfahrens „Feld bei Schlüsselwortänderung aktualisieren".

Ein weiterer Parameter innerhalb der Anwendungsentwicklung von Notes/Domino kann sich massiv auf das Laufzeitverhalten der Anwendung auswirken: Der Parameter „cache" oder „nocache" kann bei der Verwendung der @Db-Formeln angegeben werden. Dabei wird festgelegt, ob die mit Hilfe dieser Formel eingelesenen Informationen jeweils direkt aus der Datenbank eingelesen werden oder nach einem einmaligen Einlesen in einem Cache zwischengespeichert werden. In der Praxis wird dieser Parameter oft aus „Ressourcengründen" so eingestellt, dass die Informationen direkt aus der Datenbank eingelesen werden – dies geht in vielen Fällen stark zu Lasten der Ausführungsgeschwindigkeit, denn der Zugriff auf die Informationen kann unter Umständen mehr Ressourcen binden als das Zwischenspeichern. Als grobe Regelt gilt hier: Verwenden Sie die Option „nocache" immer dann, wenn sich Daten im Zuge der Verarbeitung in einer Maske mehrfach ändern; verwenden Sie „cache" dann, wenn Daten keinen häufigen Änderungen in dem jeweiligen Verarbeitungsschritt unterliegen.

Das nachfolgende Beispiel soll dies verdeutlichen helfen:

Die Verwendung des Parameters „cache" kann zum Beispiel bei Umsatzzahlen, die sich nicht mehr verändern („Umsatz Vormonat"), zur Anwendung

kommen. Dieser Wert ist in der Regel nicht mehr veränderbar und ein einmaliges Einlesen der Informationen genügt für die weitere Bearbeitung. Dabei können diese Informationen im Cache zwischengespeichert werden. Für solche Werte sollte der Parameter „cache" verwendet werden.

In der Praxis findet der Parameter „cache" oft in Verbindung mit weiteren Entwicklungseinstellungen Anwendung:

Nehmen wir eine Diskussionsdatenbank, in der Anwender neue Kategorien anlegen können. Das ist zunächst wenig spektakulär, allerdings müssen bei der Eingabe mehrerer Kategorien hintereinander diese Neueingaben jeweils aktuell für die Folgeverarbeitung zur Verfügung stehen. In solch einem Fall empfiehlt sich die Verwendung von „nocache" in Kombination mit dem Parameter „ODBC-Zugriff". Das Zusammenspiel dieser beiden Einstellungen ermöglicht performante Zugriffe auf externe Daten. Die Angabe des Parameters „ODBC-Zugriff" erfolgt auf Ansichtsebene.

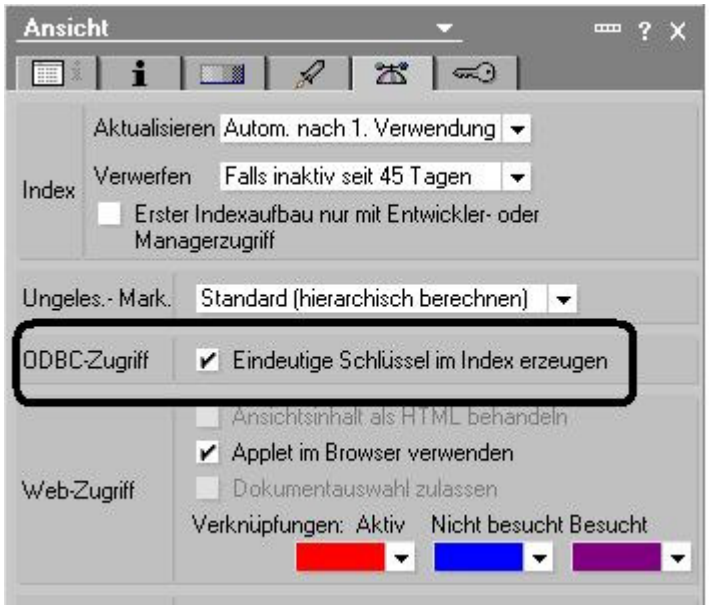

Abb.003: ODBC-Zugriff

Die Option „ODBC-Zugriff" auf Ansichtsebene dient der Erstellung eines eindeutigen Indizes auf diese Ansicht, der unter anderem für den Zugriff mittels ODBC verwendet wird. Dieser Zugriffspfad ist eindeutig und die Basis für schnelle und gezielte Datenbankzugriffe.

Stellen wir uns ein Beispiel vor:

Wir haben eine Diskussionsdatenbank mit 10.000 Hauptdokumenten und 100.000 Antwortdokumenten. Eine Ansicht ermöglicht die Selektion aller Hauptdokumente, um auf diese gezielt zugreifen zu können. Verwenden wir für diese Ansicht „Eindeutige Schlüssel im Index erzeugen", wird neben einem gezielten Index auch die Eindeutigkeit der Einträge sichergestellt – d.h.: Jedes Hauptdokument kommt auch nur einmal in der Ansicht vor. Wird unter Verwendung von zum Beispiel @DbLookup auf diese Ansicht zugegriffen, so sind die Zugriffszeiten entsprechend schneller, da gezielt weniger Dokumente verarbeitet werden müssen. Nehmen wir einmal an, in der Beispieldatenbank existieren 10.000 Hauptdokumente, davon einige Doppeleinträge. Durch den eindeutigen Index werden diese Doppelvorkommen reduziert und jedes Hauptdokument existiert in dieser Ansicht eindeutig. Zum Beispiel existieren 150 eindeutige Hauptdokumente. Dies hat zur Folge, dass anstelle von 10.000 Dokumenten 150 Dokumente verarbeitet werden müssen!

Bei dieser Methode des Datenzugriffs muss zudem sichergestellt sein, dass im einen Fall, in dem mehrere Kategorien einem Dokument zugeordnet sind, auch tatsächlich alle einzelnen Kategorien verarbeitet werden!

Neben diesen Methoden der performance-optimierten

Anwendungsentwicklung, die sich weitestgehend auf die Verwendung der Formula-Sprache konzentrierten, gibt es selbstverständlich auch eine Vielzahl von Tipps in den übrigen Entwicklungssprachen und Werkzeugen von Notes/Domino, die wir in einem der nächsten Artikel näher betrachten werden.

Den Autor Jörg Zeig erreichen Sie unter ZeBIS Beratung/Informationssysteme - Im Bühlsgarten 3 - 57223 Kreuztal - Tel. (+49) 2732/892491 - e-Mail: HJZeig@ZeBIS.de