Performance

Performance-Probleme können unterschiedliche Gründe haben.
Häufig sind es mehrere Ursachen, die in Summe zu einer schlechten Performance führen.

- Datenbank Deadlock-Fehler
- Datenbank Wartungsoperationen
- Datenbank MS SQL Trigger Bugfix
- Suche und Suchmethode
- Anzahl Vorgänge im Posteingang bzw. im Aufgabenordner
- Kontrolle UserExits
- Job-Skripte
- Arbeitsspeicher

Datenbank Deadlock-Fehler

Bei einem Schreibbefehl auf einen Datenbank-Transaktion erhält die Datenbank-Zeile einen Deadlock. Der Deadlock wird nach Abschluss der Transaktion wieder aufgehoben. Wird während des Deadlocks ein weiterer Schreibbefehl auf den Datenbank-Eintrag ausgeführt, wird ein Deadlock-Fehler ausgegeben. Der Fehler entsteht somit auf der Datenbank.

Die umgesetzten Lösungen können lediglich versuchen, Schreibbefehle auf die Datenbank zu minimieren und ein entsprechendes Fehler-Handling anzubieten. Die Gentable-Funktion "**gentable.insertIntoDB()**" schreibt zum Beispiel die Rechnungs-Positionen in die Tabelle "**invoice_posting_pos**". Die Funktion wird bei jedem Speichern einmalig ausgeführt. Alle Rechnungs-Positionen aller Rechnungen werden in diese Datenbank-Tabelle geschrieben. Die Tabelle wächst demnach immer weiter und bei Lösch- und Update- Befehlen wird es immer aufwendiger, den gewünschten Datensatz zu ermitteln. Entsprechend ist es wichtig, Wartungsoperationen bei einer SQL Datenbank einzuplanen!

Das Schreiben in die Datenbank kann über den Parameter "**GentableSuppressWritingDataIntoDb**" unterdrückt werden. Bei Rechnungen mit Bestelltbezug werden die korrekten Positionsdaten in der Tabelle allerdings zwingend für die 3-Way-Match-Prüfungen benötigt. Entsprechend ist das Unterdrücken der Schreibbefehle keine generelle Lösung.

Die folgende Beispiel-Fehlerausgabe stammt von einem MS SQL-Server. Als Lösung schlägt der SQL-Server vor, die Transaktion erneut auszuführen.

ReturnCode: -1
SqlState: 40001
NativeError: 1205
ErrorMsg: [Microsoft][ODBC SQL Server Driver][SQL Server]Transaction (Process ID 92) was deadlocked on lock | communication buffer resources with another process and has been chosen as the deadlock victim. Rerun the transaction.
.. this.Result:=false

Ab der Invoice-Version 2.0.200 wird dieser Vorschlag befolgt. Bei einem Schreibfehler in die Datenbank wird ein util.sleep() ausgeführt und der Schreibbefehl wird wiederholt.

Datenbank Wartungsoperationen

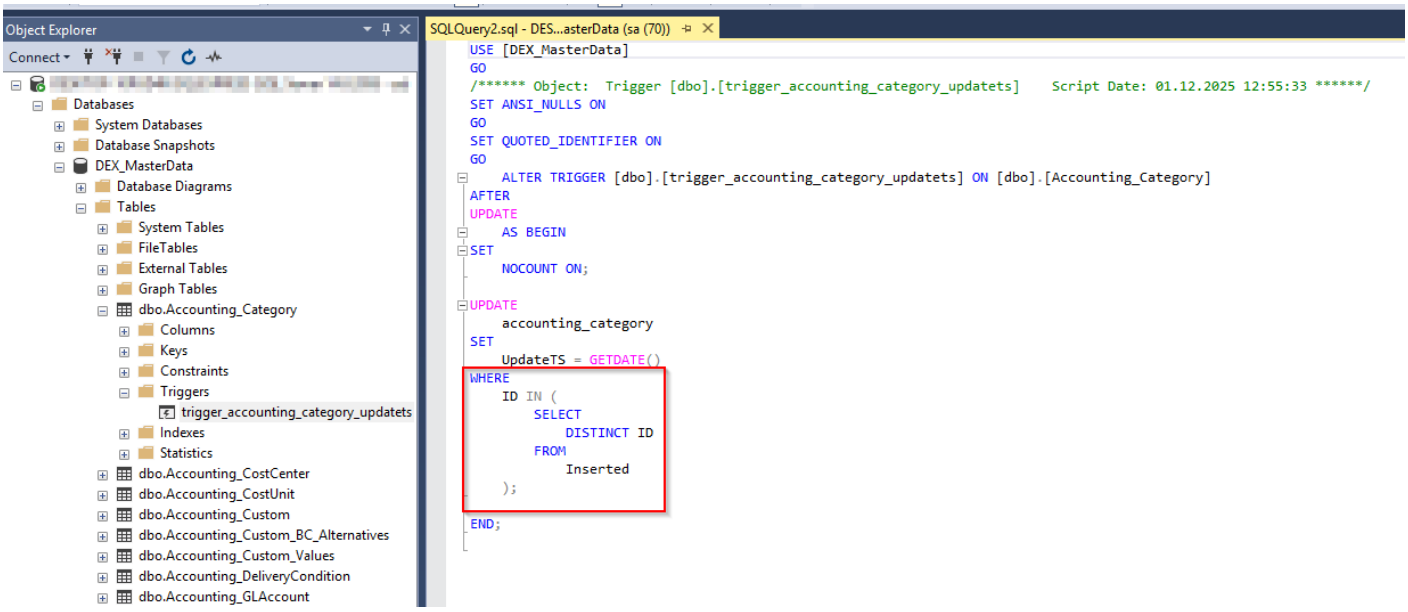
Documents benötigt performante Datenbankzugriffe! Damit eine MS SQL-Datenbank performant bleibt, müssen Wartungsoperationen eingeplant werden. Eine fehlende Wartung der Datenbank führt dazu, dass die Datenbank-Performance im Laufe der Zeit immer langsamer wird!
Es sollte zunächst immer eine vollständige Sicherung der Datenbanken erfolgen! Im Anschluss sollte eine Verkleinerung der Datenbanken durchgeführt werden und die Statistiken sollten aktualisiert werden und die Indizes sollten neu aufgebaut werden.
Wenn das Datenbank Wiederherstellungsmodell nicht auf simpel steht, wird die Datenbank nicht verkleinert und die Performance leidet darunter!

Datenbank MS SQL Trigger Bugfix

Alle Tabellen der **DEX_MasterData** und **DEX_Workflow** haben einen Trigger für die Tabellen-Spalte **UpdateTS**. Durch diesen Trigger wird bei jederem Update-Befehl der Spaltenwert aktualisiert. Hierdurch kann man bei Supportfällen schnell herausfinden, wann Datensätze zuletzt geändert wurden.

In einigen Invoice-Versionen fehlte die WHERE-Bedingung im Update-Befehl und dadurch wurde das Update immer auf alle Zeilen ausgeführt. Dies kann zum Beispiel beim Schreiben der Rechnungspositionen in die **Invoice_Posting_Pos** Tabelle zu Deadlock-Fehlern führen.

Als Bugfix gibt es 2 Möglichkeiten
1. Die WHERE-Bedingung muss manuell zu allen Triggern hinzugefügt werden
2. Der Trigger kann gelöscht werden



Suche und Suchmethode

Suchmethode

Die Suchmethode kann in den globalen Einstellungen konfiguriert werden der Standard bei der Auslieferung war lange Zeit die Suchmethode 0. Diese Suchmethode ist komfortabel aber sehr performancelastig. Wenn die Datenmenge ansteigt und mehrere User gleichzeitig die Volltextsuche verwenden, kann die Performance schnell merklich sinken.
Die Suchmethode sollte auf 1 oder 2 geändert werden.

Richtig suchen

Die Anwender sollten in der Suche geschult werden und es muss darauf hingewiesen werden, dass es für die gesamte Systemperformance besser ist, wenn die Suchbegriffe in die passenden Suchmaskenfelder eingetragen werden und die Volltextsuche möglichst vermieden wird. Wenn ein großteil der User generell mit der Suchmethode 0 im Volltext über alle Workflow- und Archivbelege sucht, kann diese Arbeitsweise schnell zu einem Performance-Thema werden!
Wenn ich als Anwender einen Vorgang im Workflow suche, dann muss ich zum Beispiel nicht zusätzlich das Archiv durchsuchen. Wenn ich nach einer Rechnungsnummer suche, kann ich die Suche auf den Mappentypen Invoice und das entsprechende Index-Feld einschränken.

Anzahl Vorgänge im Posteingang bzw. im Aufgabenordner

Die Anzahl der Vorgänge im persönlichen Posteingang kann sich auf die Performance auswirken. Wenn ein Anwender tausende von Vorgängen im Posteingang hat, kann bereits das Öffnen des ersten Vorgangs sehr lange dauern und auch bei der Weiterleitung eines Vorgangs dauert es lange.

Die Ablage im Posteingang der Anwender kann über die Workflow-Aktions-Konfiguration deaktiviert werden. Häufig wird zum Beispiel die Validierung von allen Rechnungen durch eine Gruppe ausgeführt. In dem Fall sollte die Ablage der Belege im persönlichen Posteingang unterdrückt werden, damit es nicht zu dem oben beschriebenen Problem kommt.

Ab der Invoice Version **2.0.500** gibt es das Job-Skript **DEXPRO_JOB_EmptyUserInbox**. Der Job entfernt alle Vorgänge aus den Posteingängen der definierten Anwender und verwendet den Parameter **RemoveFilesFromUsersInbox**, über den die User konfiguriert werden können.

Kontrolle UserExits

Beim Speichern und bei Abschluss einer Aktion werden Skripte ausgeführt. Die Skripte selber sind verschlüsselt und können im Projekt nicht angepasst werden. Die Prüfungen in den verschlüsselten Bereichen können zum Teil über Parameter an- und ausgeschaltet werden.
In den Skripten werden UserExit-Funktionen ausgeführt und in diesen UserExits kann beliebiger Code ausgeführt werden. Bei Performance-Problemen sollte dieser Code auf lange Ausführungszeiten überprüft werden.

Speichern

Im Invoice-Modul wird beim Speichern das Skript **Invoice_DF_OnSave** ausgeführt und im Skript werden UserExit-Funktionen ausgeführt. Diese befinden sich im Skript **Invoice__UserExit_DocFileLib**. Es gibt eine Funktion **ue_OnSaveStart()**, die direkt zu Beginn ausgeführt wird. Wichtig ist zum Beispiel, dass pro Skriptausführung nur ein sync()-Befehl auf den Vorgang erfolgen sollte. In den UserExits sollte generell auf sync()-Befehle verzichtet werden! Die Rechnungsdaten werden später im Skript gespeichert. Werden im UserExit andere projektspezifische Funktionen aufgerufen, dann muss man sich auch diese Funktionen ansehen.

> In den UserExits sollten keine sync()-Befehle ausgeführt werden.

Es folgen diverse Prüfungen auf Kopf-Ebene, die über diverse Parameter gesteuert werden können. Im Anschluss werden die Rechnungspositionen in das Gentable-Objekt geladen und es werden weitere Prüfungen ausgeführt. Nach den Standard-Prüfungen wird extra eine UserExit-Funktion für das Gentable bereitgestellt. Manchmal wird in den Projekten bereits im ue_OnSaveStart() das Gentable ausgelesen und zurückgeschrieben, wodurch ein unnötiges sync() ausgeführt wird. Hier kann man den Code optimieren, indem man das korrekte UserExit verwendet. Über die UserExit Funktion **Gentable.prototype.ue_Invoice_OnSave** können projektspezifische Ausführungen am Gentable-Objekt vorgenommen werden. Das Objekt muss nicht in das Gentable-Feld zurückgeschrieben werden. Das passiert automatisch im Anschluss.
Nach dem Gentable-UserExit werden die Positionsdaten in die Datenbank-Tabelle **Invoice_Posting_Pos** geschrieben. Über den Parameter **GentableSuppressWritingDataIntoDb** kann zum Beispiel im Invoice-Modul das Schreiben in die Datenbank deaktiviert werden. Wichtig ist aber, dass die Daten zu bestimmten Zeitpunkten in die Datenbank geschrieben werden. Bei Rechnungen mit Bestellbezug werden in der Regel zwingend aktualisierte Daten in den Tabellen benötigt.

Beim Zurückschreiben der Rechnungspositionen in das Gentable-Feld wird ein sync()-Befehl auf den Vorgang ausgeführt und das sollte auch der einzige im Skript ausgeführte sync()-Befehl sein! Die Kopfdaten zu einer Rechnung werden in die Tabelle **Invoice_Posting_Head** geschrieben. Dies kann über den Parameter **HeadDataSuppressWritingDataIntoDb** deaktiviert werden.

Zum Abschluss wird nach allen Prüfungen und Anpassungen die UserExit-Funktion **ue_OnSave_End()** ausgeführt. Hier muss berücksichtigt werden, dass der sync()-Befehl im Skript bereits erfolgt ist.

Im Procurement-Modul und weiteren Modulen sind die UserExit-Funktionen analog aufgebaut.

Abschluss einer Aktion

Für den Abschluss einer Aktion kann über die Aktions-Konfiguration der ausgehende Kontrollfluss als Button angezeigt werden. In dem Fall erfolgt die Weiterleitung direkt. Der ausgehende Kontrollfluss kann auch ausgeblendet werden und die Weiterleitung kann über eine im Standard enthaltene benutzerdefinierte Aktion im Skript erfolgen (Hintergrund: nur so sind aus technischen Beschränkungen andere Beschriftungen auf dem Button möglich). An den benutzerdefinierten Aktion sind die Skripte **DEXPRO_Action_FinishAction**, **DEXPRO_Action_FinishAction_Distribute** oder **DEXPRO_Action_FinishAction_Distribute_AP** hinterlegt. Die Skripte selber haben keine UserExits und leiten nur den Vorgang weiter.

Es kann aber auch sein, dass eine projektspezifische benutzerdefinierte Aktion für die Weiterleitung verwendet wird. In dem Fall kann im Skript beliebiger Code enthalten sein und der sollte bei Performance-Problemen kontrolliert werden.

Eine benutzerdefinierte Aktion kann im Bearbeitungs-Modus angezeigt werden. Wenn die benutzerdefinierte Aktion ausgeführt wird, wird zuerst der Speicher-Vorgang gestartet. Um die Laufzeit der Weiterleitung getrennt vom Speichern zu prüfen, sollte zuerst gespeichert werden und dann darf erst die benutzerdefinierte Aktion ausgeführt werden.

Bei einer Weiterleitung über eine benutzerdefinierte Aktion wird die Weiterleitung über ein DocFile.forwardFile() ausgeführt. Die komplette Weiterleitung wird jetzt im Kontext des Skripts ausgeführt, welches bei der benutzerdefinierten Aktion hinterlegt ist. Die Ausführungszeit des Skripts umfasst die komplette Weiterleitung von dem Start der Weiterleitung bis der Vorgang einen neuen Workflow-Schritt erreicht hat! Zunächst wird das Prüfskript **DEXPRO_WF_CheckActionEnd** auf dem ausgehenden Kontrollfluss ausgeführt. Auch hier gibt es wieder UserExits. Da der Workflow für alle Module derselbe ist, befinden sich die UserExits im globalen Skript **DEXPRO__UserExit_WorkflowLib**. Analog zum Speichern gibt es das UserExit **ue_OnActionEnd_Start()**, welches zu Beginn des Skriptes vor den Standard-Prüfungen ausgeführt wird. Dann finden Prüfungen im Gentable statt und die Gentable-UserExit-Funktion **Gentable.prototype.ue_OnActionEnd** befindet sich im Skript **DEXPRO__UserExit_GentableAdd**. NAch den Prüfungen und nach dem sync()-Befehl gibt es noch die UserExit-Funktion **ue_OnActionEnd_End()**. Diese UserExits müssen auf Performance geprüft werden.

Wenn die Weiterleitung nicht durch eine Fehlermeldung gestoppt wird, wird die Folge-Aktion ermittelt; die Workflow-Regeln werden ausgewertet und ggf. werden Aktionen nach Auswertung der Workflow-Regeln übersprungen und dann wird die Folge-Aktion ermittelt und dann müssen die Workflow-Regeln auch wieder zur nächsten Aktion ausgewertet werden.

Wenn der neue Workflow-Schritt erreicht ist, wird der Vorgang den neuen zuständigen Benutzern ggf. im Posteingang hinzugefügt. Wenn einer der neu zuständigen Benutzer über neue Vorgänge im Posteingang direkt informiert wird, dann wird auch noch die Mail an den neuen Benutzer versendet und all dies passiert immer noch im Kontext des DocFile.forwardFile()! (Die Mailversendung kann über die Mandanteneigenschaft **EMailAsync** auf eine asynchrone Versendung umgestellt werden. Der Nachteil ist, dass Fehler bei einer manuellen Mailversendung nicht mehr direkt an den Anwender zurückgegeben werden.)

Am Ende wird dem Anwender ggf. direkt der nächste Vorgang angezeigt. Wenn der ausgehende Kontrollfluss als Button angezeigt wird, dann kümmert sich Documents um die Anzeige. Wenn der Button als benutzerdefinierte Aktion angezeigt wird, muss via Skripting der nächste Vorgang ermittelt werden und hier kann via UserExit **NavigationReturnObject.prototype.ue_Next** eingegriffen werden.

Das DocFile.forwardFile() umfasst demnach viele andere Ausführungen und Einzelschritte. Im Support kommt bei Performance-Problemen häufig die Meldung an, dass das Skript **DEXPRO_Action_FinishAction** eine lange Laufzeiten hat und dieses Skript optimiert werden soll. Das Skript selber führt aber nicht viel mehr als das DocFile.forwardFile() aus und demnach gibt es hier nicht viel zu optimieren! Das eigentliche Problem liegt irgendwo bei den Einzelschritten im DocFile.forwardFile()!

> Die Weiterleitung zum Beispiel über das Skript **DEXPRO_Action_FinishAction** umfasst alle Workflow-Skripte; die Zuordnung des Vorgangs in die Postangangs-Körbe der neuen User und umfasst selbst die Mailversendung über neue Vorgänge im Posteingang!

Eine einfache Variante, um bei der Weiterleitung diverse Einzelschritte auszuschließen kann ereicht werden, ist eine Verzögerung. Hierzu gibt es 2 Varianten.

Parameter Workflow_DelayAfterAction

Über den Parameter **Workflow_DelayAfterAction** kann direkt nach Abschluss der Aktion eine Verzögerung angesteuert werden. Bei Abschluss der Aktion wird lediglich das Prüf-Skript **DEXPRO_WF_CheckActionEnd** ausgeführt. Der Vorgang läuft im Anschluss direkt in eine Workflow-Verzögerung von einer Minute. Die interne Documents-Job-Engine löst die Verzögerung auf. Der Nachteil ist, dass man auf die Documents-Job-Engine angewiesen ist. Sollte die Documents-Job-Engine mit der Ausführung diverser Jobs ausgelastet sein, kann es ggf. lange dauern, bis die Verzögerung aufgelöst wird. In dem Fall sollte man sich wiederum die Asuführungszeitpunkte und Laufzeiten der Jobs anschauen und hätte hier eine mögliche Ursache für die Performance-Probleme. Ein weiterer Nachteil ist, dass die Workflow-Aktion am Vorgang noch den alten Wert hat und der Filter am zugehörigen Ordner ggf. erweitert werden muss, um die Vorgänge in Verzögerung auszublenden.
Durch die Verzögerung direkt nach der Aktion entfallen diverse potentiel performancelastige Aktionen. Lediglich die Anzeige des Folge-Vorgangs wird wieder ausgeführt. Sollte die Ausführungsdauer immer noch langsam sein, wird die Ursache sehr wahrscheinlich an den UserExits bei Abschluss der Aktion liegen.

Technische Workflow-Aktionen Delay, Delay1, Delay2, ...

Bei der Variante mit einer technischen Aktion werden einige zusätzliche Aktionen ausgeführt, die sich aber nicht großartig auf die Performance auswirken sollten. Die technischen Workflow-Aktionen **Delay**, **Delay1**, **Delay2**, ... **Delay5** können zwischen bestehenden Aktionen hinzugefügt werden. Es handelt sich nicht um eine Workflow-Verzögerung! Der Vorgang liegt in einer Workflow-Aktion und wird durch die Gruppe **TechAccessProfile** gesperrt. Dieser Gruppe sollten nur technische User zugeordnet werden und die Mailversendung über neue Posteingänge sollte bei diesen Usern deaktiviert sein! Nach Abschluss der Aktion und nach Abschluss der Prüfungen wird die Folge-Aktion ermittelt und die Workflow-Aktion am Vorgang wird auf "Delay" (bzw. "Delay1", ...) umgestellt. Dadurch müssen die Filter an den bestehenden Ordnern nicht erweitert werden. Workflow-Regeln müssen für technische Aktionen nicht berechnet werden.
Für jedes Modul gibt es entsprechende Skripte, um die Verzögerung aufzulösen. Im Invoice-Modul muss zum Beispiel das Job-Skript **Invoice_JOB_ForwardDelay** als Job eingeplant werden.

Job-Skripte

Ein wichtiges Performance-Thema ist die Ausführung von Job-Skripten. Hier verhält es sich ähnlich wie bei den Suchmethoden. Die bequemste Suchmethode ist extrem performancelastig.

Zu Beginn eines Projekts wird das Ausführungsintervall bei Jobs gerne auf stündlich oder auf eine noch häufigere Ausführung eingstellt. Das ist zu Beginn eines Projekts auch häufig kein Problem, da sich wenige Daten im System befinden. Die Anzahl der Vorgänge wächst nach dem produktiven Start langsam an. Zu Beginn hat der Kunde ggf. nur mit einem Mandanten gestartet und nach einige Zeit werden weitere Mandanten aktiv geschaltet. Die Anzahl an Stammdaten steigt ebenso wie die Anzahl an Vorgängen.

Jobs, die bei Projektbeginn nur wenige Sekunden oder wenige Minuten gebraucht haben, laufen dann ggf. über eine Stunde und blockieren mit der Ausführung ggf. auch wiederum andere Jobs. Vor allem können aufwändige Jobs die gesamte System-Performance herunterziehen.

Bei Performance-Problemen sollte man sich zu jedem einzelnen Job die Ausführungsdauer anschauen und entsprechend muss eine sinnvolle Ausführungshäufigkeit abgeleitet werden. Aufwändige Jobs, die nicht zwingend tagsüber laufen müssen, sollten generell nur am späten Nachmittag oder in der Nacht ausgeführt werden.

Arbeitsspeicher

In der Standardauslieferung verwendet Documents maximal **4GB** Arbeitsspeicher. In der **documents.ini** kann der Eintrag

- **$MM_LogMemory 1**

hinzugefügt werden, um alle 10 Sekunden Informationen zum Speicherverbrauch in die Logs zu schreiben. Sollte der Speicherverbrauch permanent sehr hoch sein, kann der maximal verwendete Speicher über den Eintrag

- **$MM_MaxPrivateBytes 32000**

auf zum Beispiel **32GB** erhöht werden.