www.ordix.de

ORDIX news

02
2007
€ 2,20

Das IT-Magazin der ORDIX AG

Überblick über Oracle Grid Control S. 8

UCARP: Unix/Linux Hochverfügbarkeit
Aufbau eines Aktiv-Passiv-Clusters S. 14

DB2 UDB 9.1 „Viper“: Sicherheit mit LBAC
Vergabe von Lese- und Schreibrechten per Label Based Access Control S. 54

MySQL 5.1: Zeitgesteuerte Jobs
Vorstellung des Event Managers S. 46

PL/SQL-Performance unter Oracle 10g
Code Generator, PL/SQL Virtual Machine, Automatic Performance Diagnostic, etc. S. 5

20 Jahre DOAG!

Vorankündigung

20. Deutsche ORACLE-Anwenderkonferenz

Integriert

5. Deutsche ORACLE Business-Software Anwenderkonferenz

21. –22. November 2007 im CCN CongressCenter Nürnberg Ost

Ab Mitte März 2007
Ausstelleranmeldung

Ab Anfang Mai 2007
Call for Papers

Ab Ende Mai 2007
Teilnehmeranmeldung

Schwerpunktthemen

- ***Die neue Version ORACLE 11g***
- ***Erfahrungen mit Oracle Applications***

Zusätzlich

20.11.2007	***DOAG-Tag***
23.11.2007	***Schulungstag***

Informationen unter: www.doag.org/go/konferenz

Ei, ei, ei ...

Paderborn, April 2007

Ipod, demnächst Iphone – Apple macht mit allem Möglichen beste Geschäfte, nur nicht mit dem ursprünglich angestammten Business, PCs zu vermarkten. Da hat es Microsoft so richtig schwer. Dort läuft alles Neue von Xbox bis Zune nur mäßig. Geld wird immer noch mit „good (?) old winword“ und anderen Oldies verdient. Ja, Eigenentwicklungen sind eben schwer.

Oracle macht sich das Leben etwas leichter. Dort werden neue Produkte weiterhin einfach eingekauft. Auch hier kann man über den Erfolg geteilter Meinung sein, ebenso wie über die Schießübung in Richtung SAP. Anerkannte Fehlermeldungen zur „good (?) old Database“ bleiben dafür aber über Monate liegen. Nachdem man eigene Überlegungen angestellt und sich viel Arbeit mit Workarounds gemacht hat, bekommt man nach Wochen immerhin eine Bestätigung, dass das durchaus „so funktionieren könnte“.

Wenn jetzt jemand sagt, „da kann ich doch von Oracle zu Informix wechseln“, stellt sich schnell die Frage: Informix, was ist das? Ach ja, das ist das Produkt, das IBM vor Jahren gekauft hatte, das dann im Schrank liegen gelassen wurde und womit Kunden verschreckt wurden (die dann zu Oracle wechselten), das Datenbank System, das in „good (?) old Germany“ ein kurzes Marketing-Revival auf der privaten Internetseite eines engagierten Consultants erlebte.

Jetzt versucht IBM mit dem unglücklich gewählten Namen „Cheetah“ (es handelt sich hierbei nicht um Tarzans Affen), Informix erneut zu pushen. Googeln Sie nach Cheetah, finden Sie immerhin auf den ersten 3 Seiten einen (!) Eintrag zu Informix. Wesentlich mehr Ergebnisse beschäftigen sich aber mit dem 75sten Geburtstag des eben erwähnten Affen. So leid mir das für ein ehemals relativ gutes Produkt tut: Der Zug scheint abgefahren zu sein und auch ein Gepard holt ihn vermutlich nicht mehr ein.

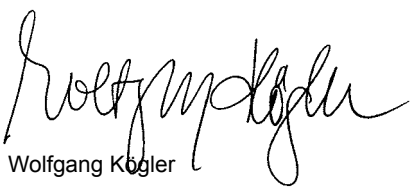
Da widme ich meine Gedanken lieber dem schönen, nachösterlichen Wetter, das gerade vorherrscht, während ich diese Zeilen schreibe und der Tatsache, dass die Mallorquiner demnächst Urlaub in Deutschland machen, weil hier das Wetter besser ist als auf der Insel. Dazu empfehle ich Ihnen unsere Auswahl an Artikeln auf den nächsten Seiten und aller Schelte zum Trotz dominiert hier eindeutig das Thema Oracle (u. a. PL/SQL Performance, Grid Control, Objekttypen).

Daneben finden Sie noch die interessanten Fortsetzungen unserer Artikel über Java Beans, Hibernate und XEN. Und auch Microsoft hat Einzug gefunden, allerdings nicht mit einem Artikel zu Winword, sondern einem weiteren Sujet, das ökonomisch auch nicht so erfolgreich ist wie Office, dem MS SQL Server.

Da ich die weißen Ipod Stöpsel nicht (ver-)trage, stülpe ich mir jetzt meinen Bluetooth-Kopfhörer über, obwohl ich damit wie ein gestrandeter Hubschrauberpilot aussehe. Ich drehe die Lautstärke des Ipod auf ein erträgliches Maß und überlege mir, ob ich für die nächste News Ausgabe einen Artikel über Daktari, Knut und Cheetah schreibe, während im Hintergrund die Reorganisation einer Informix Datenbank läuft.

Bis dahin wünsche ich Ihnen viel Spaß mit dieser neu gestalteten ORDIX News.

Schöne Grüße aus dem hochsommerlichen April-Deutschland



Wolfgang Kögler

PS: Über Informix 11 schreiben wir vermutlich in der kommenden Ausgabe. Bei Larry Ellison und Bill Gates werden wir uns anders erkenntlich zeigen ;-).

Inhalt

Training

Aktuelles

Java/J(2)EE

Projektmanagement

Betriebssysteme

Open Source

Standards

Datenbanken

Oracle 10g New Features:
Code-Generator, Initialisierungsparameter, PL/SQL VM

PL/SQL-Performance unter Oracle 10g

In Oracle 10g ist der Code-Generator vollständig erneuert worden. Der generierte Code ist nun deutlich effizienter und nach Oracles eigener Aussage sogar bis zu zweimal so schnell wie noch in Oracle 9i. Neben dem neuen Code-Generator wurden im Release 10g auch weitere Features eingeführt, die detailliert im Whitepaper „PL/SQL Just Got Faster“ [4] nachzulesen sind. Dazu zählen eine überarbeitete PL/SQL Virtual Machine (PVM) sowie die Automatic Performance Diagnostic und Tuning Features. Nachfolgend nennen wir einige Beispiele für Verbesserungen.

Der Artikel richtet sich an Oracle Datenbankadministratoren und -entwickler, die in der Version Oracle Database 10g PL/SQL-Skripte einsetzen.

Der neue Code-Generator

In vorherigen Oracle-Versionen wurde PL/SQL-Code eins zu eins übersetzt, ohne an ihm viele Änderungen vorzunehmen. Der Code-Generator der früheren Versionen war sehr mechanisch. Es galt das Motto: „What you write is what you get“. Wurde im Quellcode der Weg von A nach B über einen Umweg C gewählt, so wurde dieser Umweg während der Ausführung auch konsequent ausgeführt.

Nun wird ein neuer Code-Generator genutzt, der den Code zur Erzielung einer besseren Performance gegebenenfalls komplett umgestaltet. Und das, ohne jeglichen Eingriff von außen. Bei demselben Quellcode, der von A nach B über den Umweg C führt, weiß der Code-Generator in Oracle 10g nun, dass es auch einen direkten Weg von A nach B gibt, und führt auch nur diesen aus.

Ab Oracle 10g gilt nun: „What you write is NOT what you get“! Dies führt z. B. bei rekursiven Schleifen, wie in unserem Beispiel A zu sehen (siehe Abbildung 1), zu einer enormen Verbesserung der Performance. Viele altbekannte Regeln der PL/SQL-Programmierung sind nun hinfällig, weil sie durch den neuen Code-Generator in den zur Ausführung „günstigsten“ Code umgewandelt werden.

Neuer Initialisierungsparameter

Mit dem neuen Code-Generator wurde auch PLSQL_OPTIMIZE_LEVEL, ein neuer Initialisierungsparameter, eingeführt. Dieser beeinflusst die Code-Optimierung und kann mit den Werten 1 und 2 belegt werden, wobei 2 der Standardwert ist. Der Standardwert sollte normalerweise auch der bessere sein. Er liefert die optimale Code-Generierung, wohingegen der Wert 1 die Code-Generierung geringfügiger optimiert, dadurch aber eine kürzere Kompilierungszeit zur Folge hat.

Dauert die Kompilierung bei sehr großen Anwendungen zu lange, sollte der Wert auf 1 gesetzt werden. In sehr wenigen Fällen ist zwischen den beiden Parametern auch eine unterschiedliche Behandlung der Exceptions festzustellen. Eventuell werden Exceptions gar nicht ausgelöst oder zu einem früheren Zeitpunkt als erwartet. Wird der Parameter PLSQL_OPTIMIZE_LEVEL auf 0 gesetzt, so wird das Optimieren des Codes komplett ausgeschaltet. Dies wird natürlich nicht empfohlen.

PL/SQL Virtual Maschine

Die PVM ist, wie in allen anderen Programmiersprachen, z. B. Java, notwendig, um den vom Code-Generator erstellten Code auszuführen. Die PVM wurde zwar nicht wie der Code-Generator komplett erneuert, aber entscheidend verbessert. So werden Konkatenationen nicht mehr wie in früheren Releases nacheinander verarbeitet und temporär zwischengespeichert, sondern sie können in einer einmaligen Verkettung ausgeführt werden.

Die meisten Änderungen wurden an den Befehlen der PVM vorgenommen, z. B. an der Speicheradressierung. Außerdem konn-

DOAG News
Deutsche ORACLE-Anwendergruppe e. V.
Ebenfalls veröffentlicht in der DOAG News 4/2006
Die Zeitschrift der DOAG-Mitglieder

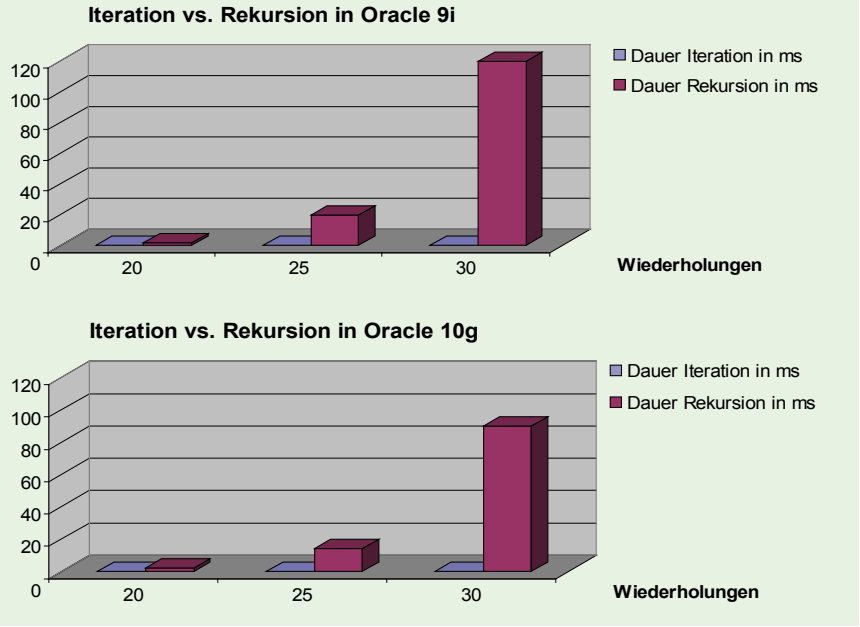


Abb. 1 a und b: Das Ergebnis des Performance-Vergleichs zwischen Rekursion und Iteration - jeweils für die Versionen 9i und 10g.

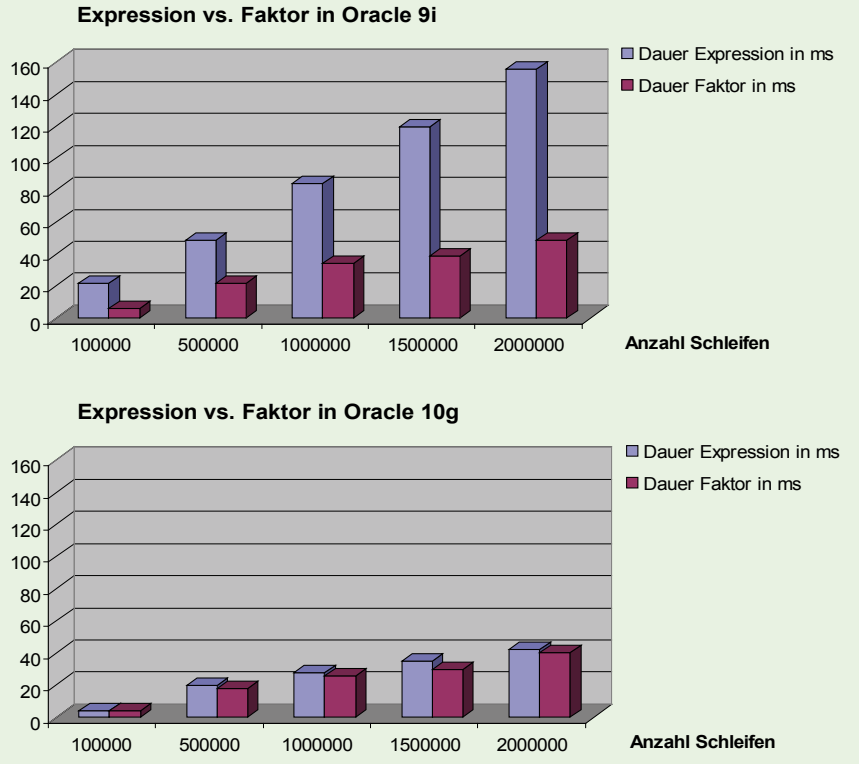


Abb. 2 a und b: Performance-Vergleich zwischen Expression und Faktor.

te durch eine bessere Programmierung der PVM selbst die Performance während der Ausführung entscheidend gesteigert werden. Da sich die PVM wie eine „Black-Box" verhält, sind diese Neuerungen vom Entwickler nicht beeinflussbar. Sie sind für ihn aber trotzdem sehr hilfreich.

Performance-Steigerung in Oracle 10g

Folgende Beispiele sollen die Performance-Steigerung in Oracle 10g verdeutlichen:

A) Rekursion versus Iteration

Jeder PL/SQL-Entwickler weiß, dass eine Rekursion immer langsamer ist als eine gleiche, iterative Lösung.

Diese Aussage trifft zwar auch in Oracle 10g zu, aber in unseren Tests konnten rekursive Lösungen gegenüber Oracle 9i deutlich aufholen. Den Beispiel-Code, der dem Vergleich zugrunde liegt, finden Sie im Internet unter [1].

Wie Abbildungen 1 a und b zeigen, sind rekursive Lösungen in Oracle 10g deutlich leistungsstärker als in früheren Versionen. Im Vergleich zu Oracle 9i konnten wir eine bis zu circa 25 Prozent gesteigerte Performance feststellen. Nichts desto trotz konnte die Rekursion die Performance der Iteration nicht einholen.

B) Expression versus Faktor

Ein Grundsatz, der für jeden PL/SQL-Entwickler gilt, ist, dass Rechen-Ausdrücke nicht innerhalb einer Schleife zu verwenden sind. Dies führte in Oracle 9i zu deutlichen Performance-Einbußen, da die Rechenoperation in jedem Schleifendurchlauf erneut auszuführen war.

Dies gilt nicht mehr in Oracle 10g. Hier sind sogar fast gleichwertige Zeiten festzustellen. Den Code zu einem einfachen Beispiel finden Sie unter [2].

Auch die Abbildungen 2 a und b zeigen den deutlichen Performance-Unterschied zwischen Expression und Faktor. In Oracle 10g ergibt sich im Fall von Expressions sogar eine Performance-Verbesserung von bis zu 75 Prozent gegenüber der Version Oracle 9i.

C) Konstanten in PL/SQL

Als letzte Einflussgröße seien Konstanten erwähnt. Diese hatten bisher keinen Einfluss auf die Performance einer PL/SQL-Anwendung. Dies wird nun mit dem Beispiel unter [3] für Oracle 10g widerlegt. Das Ergebnis

aus den Abbildungen 3 a und b belegt, dass die vorzugsweise Nutzung von Konstanten gegenüber einer Nutzung von Variablen geringfügig die Performance verbessert.

Es gibt unzählige weitere Beispiele, die eine Performance-Steigerung in Oracle 10g verdeutlichen. Unsere Tests zeigen es. Probieren Sie es einmal selbst mit Ihren vorhandenen Skripten oder unseren Beispielen aus.

PL/SQL Performance Measurement Harness

Mit dem Harness Software Kit stellt Oracle dem Entwickler ein Tool zur Verfügung, mit dem sich die Performance-Unterschiede eines PL/SQL-Programms messen lassen. Das Tool steht zum freien Download zur Verfügung und kann mit den circa 30 vorhandenen Test-Programmen oder mit eigenen PL/SQL-Anwendungen genutzt werden.

Durch eine Vielzahl vorgefertigter Reports lassen sich so z. B. HTML-Dokumente oder sogar Microsoft Excel-Sheets erstellen. Das Tool dient nicht nur zur Performance-Messung der PL/SQL-Anwendungen auf Datenbanken der Versionen 9i und 10g, sondern es kann bereits ab der Version 8.0 benutzt werden. Eine ausführliche Anleitung sowie das komplette Software-Kit finden Sie unter [4] zum Download.

Fazit

In Oracle 10g wird dem Entwickler die Arbeit vereinfacht. Er braucht sich weniger Gedanken um die Performance-Optimierung, um schnelle Kompilierungszeiten oder gar um das Sammeln von Statistiken zu machen.

Dies erledigt der Code-Generator nun großteils selbst. Und das, wie wir gesehen haben, sogar sehr schnell und zuverlässig. Auch wenn wir die von Oracle angepriesenen 50 Prozent in unseren Testfällen nicht durchweg bestätigen konnten, ist dennoch eine deutliche Performance-Verbesserung festzustellen. Wer trotzdem Code von Hand optimieren möchte und gar keine Optimierung seitens des Code-Generators wünscht, ist aber auch weiterhin mit Oracle PL/SQL sehr gut bedient.

Man sollte sich die beiden folgenden Aspekte bewusst machen: In der Version 10g gilt „Just write it the way that feels most natural“ und „What you write is NOT what you get“.

Dustin Schmitt (info@ordix.de).

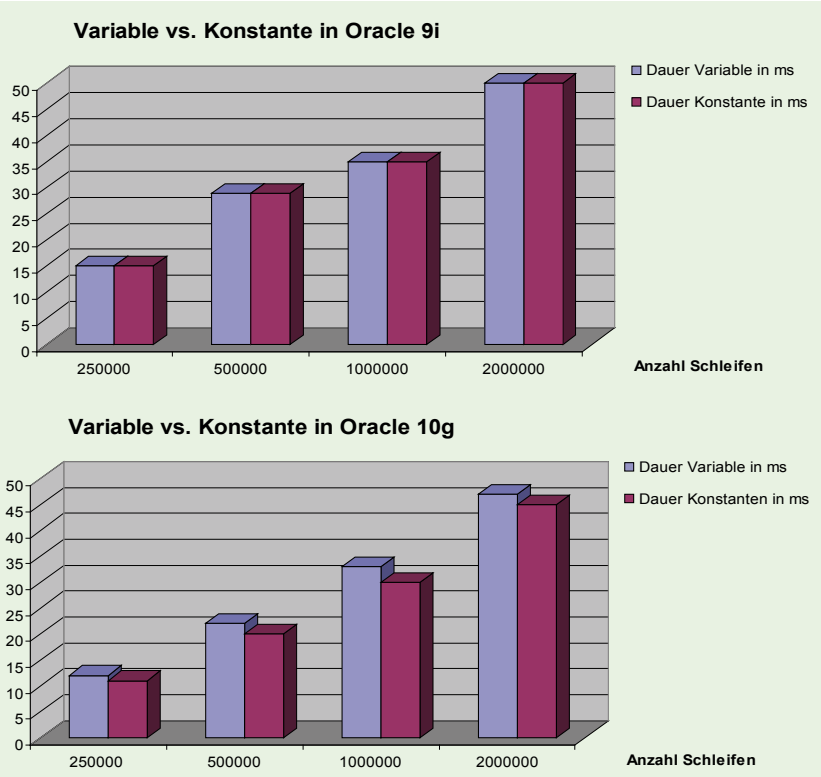


Abb. 3 a und b: Das Ergebnis des Performance-Vergleichs zwischen der Nutzung von Konstanten und Variablen in PL/SQL.

Links

- [1] Quellcode Beispiel A: http://www.ordix.de/ORDIXNews/2_2007/Datenbanken/pl_sql_performance_a.html
- [2] Quellcode Beispiel B: http://www.ordix.de/ORDIXNews/2_2007/Datenbanken/pl_sql_performance_b.html
- [3] Quellcode Beispiel C: http://www.ordix.de/ORDIXNews/2_2007/Datenbanken/pl_sql_performance_c.html
- [4] Alle Neuerungen zu PL/SQL in Oracle 10g sowie das eingangs erwähnte Whitepaper „PL/SQL Just Got Faster”: http://www.oracle.com/technology/tech/pl_sql/htdocs/New_In_10gR1.htm
- [5] Allgemeine Infos zu PL/SQL finden Sie unter: http://www.oracle.com/technology/tech/pl_sql/index.html

Glossar

PL/SQL	Prozedurale Erweiterung der Abfragesprache SQL (Structured Query Language).
Code-Generator	Erstellt aus einer Anwendung, wie z. B. PL/SQL, ausführbaren Bytecode.
Exceptions	Fehlerbehandlungsteil in PL/SQL.
PL/SQL Virtual Maschine (PVM)	Hier wird der durch den Code-Generator erstellte Bytecode ausgeführt.
Black-Box	Ein Objekt, bei dem nur das äußere Verhalten von Interesse ist. Seine innere Funktionsweise ist unbekannt.
Kompilieren	Vorgang des Code-Generators bei der Erstellung des Byte-Codes.

Oracle Enterprise Manager Grid Control (Teil I)

Überblick über Oracle Grid Control

Dieser Artikel wendet sich an Administratoren von Oracle Datenbanken in heterogenen Serverlandschaften.

Mit Einführung der Datenbankversion 10g hat Oracle einen Nachfolger des Oracle Management Server (OMS) auf den Markt gebracht. Er unterscheidet sich in Architektur und Funktionalität deutlich von seinem Vorgänger. Während der alte OMS ein java-basiertes Administrationswerkzeug für die hauseigenen Datenbanksysteme war, kommt der Enterprise Manager (EM) Grid Control als Webanwendung mit deutlich erweitertem Funktionsumfang daher. Oracle bietet mit der neuen Software nicht nur die Überwachung und Administration der eigenen Datenbankserver ab Version 8.1.7. Mit Hilfe von Plug-Ins werden auch heterogene Serverlandschaften inklusive einiger Konkurrenzprodukte wie IBM DB2 oder Microsoft SQL Server überwacht. Damit tritt Oracle Grid Control in Konkurrenz zu etablierten Tools wie Nagios und Tivoli.

Plattformen, Komponenten, Platzbedarf

Oracle stellt den Enterprise Manager (EM) Grid Control für die Plattformen Windows, Linux, Solaris, HP UX und AIX5L bereit. Das Software-Paket ist circa 1,7 GB groß und kann kostenlos von der Oracle Webseite heruntergeladen werden. Die Client Software für die Überwachung der Server im Netzwerk, bei Oracle „Agent“ genannt, ist in diesem Paket noch nicht enthalten und muss gesondert heruntergeladen werden.

Auch die Client Software steht für die oben genannten Betriebssysteme in der 32- und 64-Bit-Version zur Verfügung. Die Größe der Agenten ist sehr unterschiedlich und reicht von circa 250 MB bis zu 1 GB. Den Platzbedarf des EM Grid Control gibt Oracle mit circa 2,5 GB (ohne Repository) an und setzt mindestens 1 GB Arbeitsspeicher voraus.

Bestandteile

Der EM Grid Control besteht aus mehreren Komponenten (siehe Abbildung 1). Den Kern der Anwendung bildet der als J2EE-Anwendung realisierte Oracle Management Service. Er erhält seine Informationen von Agenten, die auf jedem zu überwachenden Host installiert werden müssen. Die gesammelten Daten werden vom Management Service in einem Datenbank Repository (Management Repository) verwaltet und dem Anwender über ein Web Interface zur Verfügung gestellt. Für das Repository wird eine Oracle Datenbank in der Enterprise Edition benötigt.

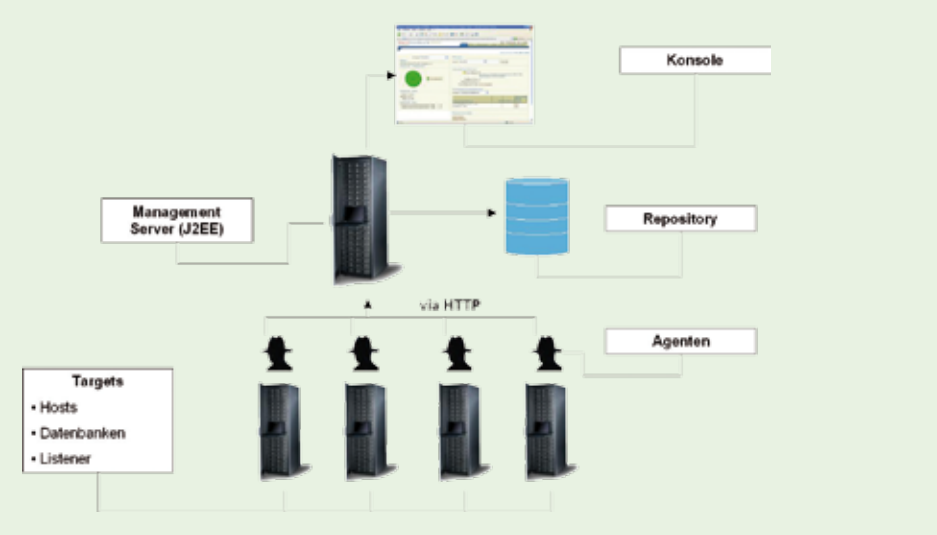


Abb. 1: Architektur des Oracle Enterprise Managers Grid Control.

Alles umsonst?

Für den Betrieb des EM Grid Control ist eine Oracle Datenbank 10g Enterprise Edition erforderlich. In dieser wird das Repository abgelegt. Wird hierzu eine vorhandene Datenbank verwendet, fallen keine weiteren Lizenzkosten an. Wenn aber eine neue Datenbank aufgesetzt werden muss, muss diese zu den üblichen Konditionen lizensiert werden. Auch der Einsatz des EM Grid Control selbst erfordert unter Umständen den Erwerb einer Lizenz.

Die Grundfunktionen zum Monitoring und zur Administration stellt Oracle zwar kostenlos zur Verfügung, die ebenfalls integrierten Tuning Tools sind aber lizenzpflichtig. Kann man jedoch auf Tuning-Tipps von Oracle verzichten und verfügt bereits über eine Datenbank in der Enterprise Edition, so erhält man mit dem EM Grid Control eine komplett kostenlose Monitoring- und Administrationslösung von Oracle.

Installation

Nach dem Download des Paketes sollte zunächst der Management Server samt Repository installiert werden. Voraussetzung hierfür ist, wie bereits erwähnt, das Vorhandensein einer Datenbank der Enterprise Edition (mindestens Version 10.1.0.4). In dem zum Download gehörenden Tutorial ist die Installation gut beschrieben und läuft meist problemlos ab.

Bereits bei der Installation ist den Agenten die IP-Adresse des laufenden Management Servers mitzugeben. Sie können deshalb erst nach dem Aufsetzen der zentralen Anwendung auf den zu überwachenden Hosts installiert werden. Ein Agent ist dabei in der Lage, mehrere Dienste gleichzeitig zu überwachen, so dass keine Mehrfachinstallationen auf ein und demselben Rechner erforderlich sind, falls mehrere Dienste auf demselben Knoten laufen.

Man sollte dabei auch nicht vergessen, einen Agenten auf dem Datenbankserver mit dem Management Repository zu installieren. Nach erfolgreicher Installation eines Agenten muss dieser noch im EM Grid Control konfiguriert werden.

Erste Schritte

Ist die Installation abgeschlossen, gilt es, ein grundlegendes Setup durchzuführen. Für die Nutzerverwaltung unterstützt Oracle die gezielte Vergabe von Zugriffsrechten auf einzelne Ziele und auf die Verwendung von Modulen des EM Grid Control. Ähnlich wie in den hauseigenen Datenbanken wird ein Rollenkonzept verwendet, um den Administrationsaufwand zu reduzieren.

Außer der User-Verwaltung kann der Administrator unter den Menüpunkten „Setup“ und „Voreinstellungen“ bevorzugte Zugangsdaten

sowohl zu den Zielsystemen als auch zu einem Mailserver für den Versand von Benachrichtigungen hinterlegen.

Konfiguriert man darüber hinaus auch einen Metalink Account, kann der EM Grid Control automatisch nach Patches für Oracle Datenbanken auf den Zielsysteme suchen. Allerdings eröffnet man Oracle damit gleichzeitig auch die Möglichkeit einer detaillierten Bestandsaufnahme der eigenen Serverlandschaft.

Warum die Konfiguration nicht komplett unter „Setup“ zusammengefasst wurde, sondern auf zwei Menüpunkte verteilt ist, ist nicht ganz nachvollziehbar. Unter Voreinstellungen kann zum einen ein Teil der Oberfläche des EM Grid Control selbst konfiguriert werden, indem man festlegt, welche Art von Zielen zu unterscheiden sind.

Zum anderen kann ein detaillierter Benachrichtigungsplan erstellt werden. Dieser ermöglicht es, für jeden Tag des Jahres genau festzulegen, welcher Administrator zu bestimmten Zeiten eventuell versandte Benachrichtigungen erhalten soll.

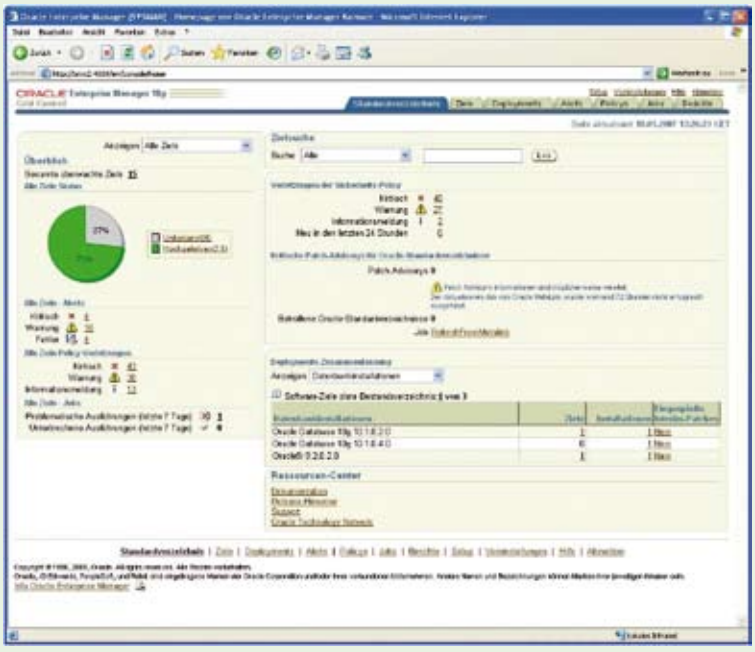


Abb. 2: Die Startseite des EM Grid Control bietet einen schnellen Überblick über den aktuellen Zustand der überwachten Systeme.

Ob dies sinnvoll ist, sei dahingestellt. In der Praxis dürfte es wesentlich komfortabler sein, entsprechende Mailverteiler einzurichten, als einen kompletten Schichtplan in den EM Grid Control zu übertragen und diesen mit allen gegebenenfalls auftretenden, kurzfristigen Änderungen, wie z. B. krankheitsbedingten Ausfällen, auf dem aktuellen Stand zu halten.

Kostenkontrolle

Ein weiterer wichtiger Punkt im Setup ist die Konfiguration des Zugriffs auf die Management Packs des EM Grid Control. Eine Verwendung kostenpflichtiger Module kann hier gezielt unterbunden werden. Es empfiehlt sich, diese Einstellungen gleich zu Beginn vorzunehmen, da die Links zu den lizenzpflichtigen Komponenten sich nicht von den anderen Verweisen abheben. Somit ist die Gefahr einer ungewollten Benutzung relativ groß. Die Höhe der Lizenzkosten ist recht unterschiedlich und richtet sich nach der Höhe der Lizenzkosten für den Server, auf dem die Instanzen laufen.

Plug-Ins

Da Oracle den EM Grid Control auch in heterogenen Systemumgebungen gerne als alleiniges Monitoring- und Management-Tool beim Kunden sehen möchte, gibt es die Möglichkeit, über Plug-Ins auch Services anderer Hersteller einzubeziehen. Oracle selbst stellt derzeit unter anderem Erweiterungen für DB2 und den Microsoft SQL-Server bereit. Das Unternehmen baut anscheinend aber auch darauf, dass Drittanbieter weitere Plug-Ins für den EM Grid Control bereitstellen oder dass Kunden selbst aktiv werden und die benötigten Komponenten eigenständig entwickeln.

Customizing

Nach dem grundlegenden Setup muss das System zunächst an die individuellen Bedürfnisse der Einsatzumgebung angepasst werden. Oracle unterscheidet bei den Meldungen des EM Grid Control Alerts und Policy-Verletzungen. Alerts sind dabei Über- bzw. Unterschreitungen von festgelegten Grenzwerten.

Policy-Verletzungen sind allgemeine Sicherheitsprobleme, wie beispielsweise Standardnutzer mit Standardpasswörtern oder offene Ports. Hier gilt es also, zu entscheiden, welche Meldungen für den konkreten Fall wichtig sind und die restlichen auszublenden. Diese

Einstellungen können für jedes Ziel einzeln vorgenommen werden. Für eine Datenbank der Version 10g hält Oracle einen Katalog von circa 180 Policys und von etwa ebenso vielen Metriken vor.

Über die Verwendung der vorgegebenen Metriken hinaus lassen sich aber auch eigene Metriken definieren. Hier ist also sehr genau zu prüfen, welche Werte im konkreten Fall wirklich von Bedeutung sind. Ansonsten besteht die große Gefahr, in einem Wust von Informationen zu ersticken.

Übersicht

Nachdem das Informationsdickicht gelichtet ist, kann die eigentliche Arbeit beginnen. Nach der Anmeldung präsentiert der EM Grid Control zunächst eine recht übersichtliche Startseite (siehe Abbildung 2).

In einem Tortendiagramm wird sofort ersichtlich, wie viele der Zielsysteme korrekt hochgefahren sind und wie viele sich in einem unbekannten Zustand befinden. In einigen Tabellen bekommt der Nutzer einen Überblick über die Gesamtzahl der Alerts und Policy-Verletzungen aller Ziele (jeweils unterteilt in die Unterpunkte „kritisch“, „Warnung“ und „Informationsmeldung“). Er bekommt ebenfalls einen Überblick über installierte Oracle Produkte und Korrekturstände. Über das Hauptmenü gelangt man von der Startseite aus in die Bereiche „Ziele“, „Deployments“, „Alerts“, „Policys“, „Jobs“ und „Berichte“.

Ziele

Über den Karteireiter „Ziele“ gelangt man am schnellsten zu einer Übersicht der zu überwachenden Systeme. Ein Untermenü schränkt auf Wunsch die Liste der Ziele gemäß den Vorgaben aus den Voreinstellungen auf eine ausgewählte Kategorie ein.

Wählt man hier beispielsweise eine Datenbankinstallation aus, so gelangt man zu einer Übersichtsseite des betreffendes Systems. In Diagrammen und Tabellen erhält man einen umfassenden Überblick über die wichtigsten Daten, wie z. B. den aktuellen Status, Uptime, CPU-Auslastung des Hosts, aktive Sessions oder die aktuellen SQL-Antwortzeiten sowie eine Übersicht über die kritischen Alerts und Warnungen.

An einigen Stellen können Aktionen, wie das Herunterfahren der Datenbank, direkt angestoßen werden. Ein weiteres Untermenü führt zu Seiten mit Performance-Informationen, Administrations- oder Wartungsmöglichkeiten.

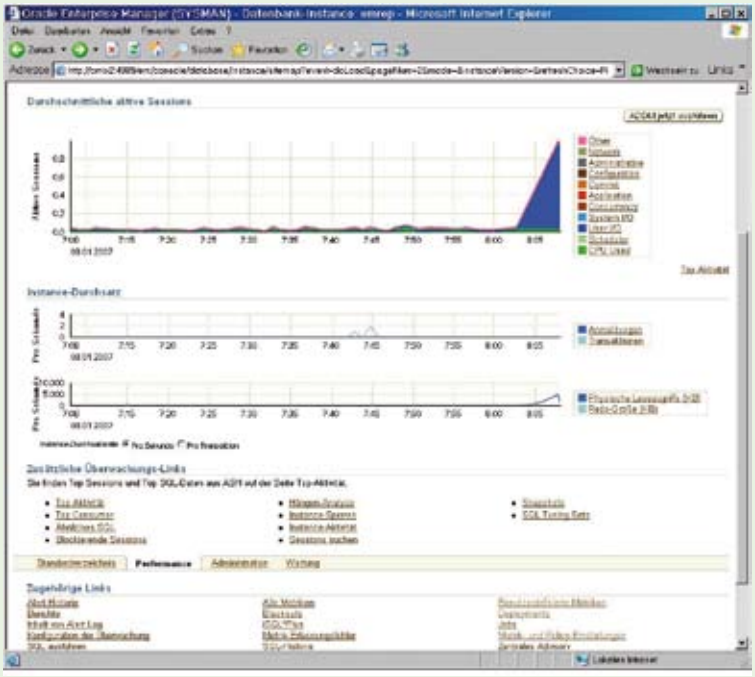


Abb. 3: Die Performance-Übersicht vermittelt einen guten Eindruck des zeitlichen Verlaufs der Systemleistungen.

Performance

Auf der Performance-Seite (siehe Abbildung 3) geben Graphen schnell einen Überblick über die zeitliche Entwicklung wichtiger Parameter wie CPU-Auslastung, Anzahl aktiver Sessions oder den Datendurchsatz. Auch hier gelangt man durch Auswahl der Werte oder Legenden wieder zu weiterführenden Seiten. Am unteren Rand der Seite findet sich meist eine Linksammlung zu ähnlichen oder weiterführenden Themen. So kann man zwar sehr schnell an detaillierte Informationen gelangen, es besteht aber auch die Gefahr, sich in der Anwendung zu verrennen, da der Funktionsumfang insgesamt sehr groß ist und die Links teilweise in völlig andere Bereiche des Systems führen.

Administration

Die Administrationsseite (siehe Abbildung 4) bietet eine umfangreiche Link-Sammlung zu

den Themen Datenbank- und Schemaverwaltung. Hier ist es möglich, Kontrolldateien, Tablespaces und Rollback-Segmente zu verwalten, Benutzer, Gruppen oder Tabellen zu bearbeiten oder einen Überblick über die Speicherparameter zu erhalten.

Die Links führen zunächst jeweils zu einer Beschreibung des aktuellen Zustandes. Im Weiteren lassen sich dort vorhandene Einträge bearbeiten, löschen oder neue Einträge erzeugen. Wo immer es möglich ist, werden dem Anwender dabei Listen mit Optionen präsentiert, aus denen er die gewünschten auswählen kann.

Wartung

Sehr ähnlich gestaltet sich der Menüpunkt “Wartung” (siehe Abbildung 5). Auch hier erwartet den Nutzer eine Liste von Tools zu Themen wie Backup/Recovery, Data Guard oder Patchverwaltung. Wie bei den Administrationstools, bilden die vorhandenen Werkzeuge auch hier meist die ansonsten vom Administrator manuell auszuführenden Aktionen Schritt für Schritt nach und unterstützen mit Optionslisten. Die angebotene Hilfe hält sich dabei sehr in Grenzen, so dass die Tools zwar einem erfahrenen Administrator das Leben einfacher machen, einen solchen aber keinesfalls ersetzen können.

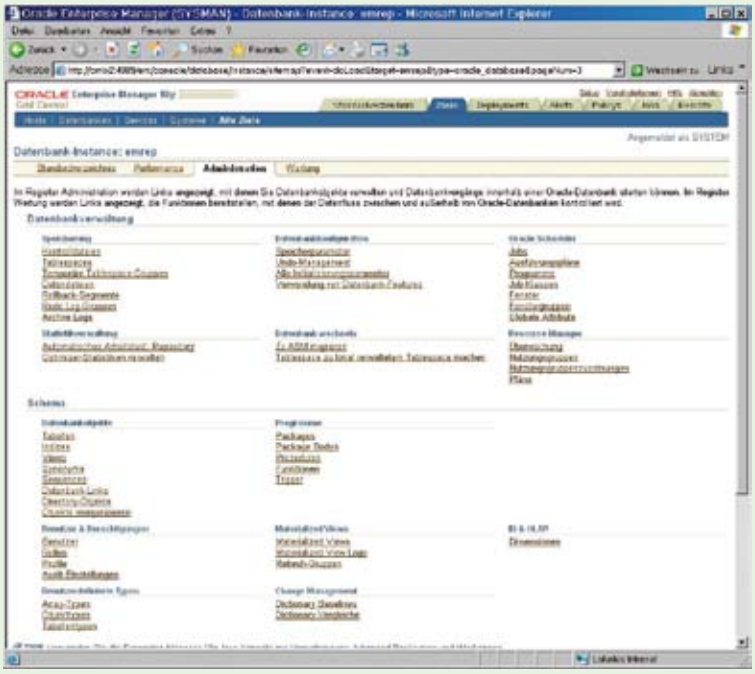


Abb. 4: Über die Administrationsseite gelangt man zu einer Vielzahl von Tools für die Verwaltung der überwachten Ziele.

Deployment, Alerts und Policys

Unter dem Punkt „Deployment“ präsentiert Oracle Informationen über die überwachten Hosts, deren Betriebssysteme sowie vorhandene Software-Installationen. Laufen auf den Hosts Oracle Darenbanken, so kann man Metalinks auf passende Patches hin durchsuchen und diese zentral gesteuert einspielen. Des Weiteren können von hier Datenbanken in einen anderen Speicherort geklont werden.

Die Karteireiter „Alerts“ und „Policys“ geben einen Überblick über die vom EM Grid Control festgestellten Grenzwertüberschreitungen und Policy-Verletzungen. Die Ansicht kann dabei gezielt auf einzelne Ziele oder Zielarten eingeschränkt werden. Bei den Alerts ist zudem eine Einschränkung auf einen bestimmten Zeitraum des Auftretens möglich.

Job-Verwaltung

Ein recht mächtiges Werkzeug des EM Grid Control stellt der integrierte Job Scheduler dar. Über ihn können Jobs aller Art zum Beispiel in Form von Betriebssystembefehlen, SQL-, oder RMAN-Skripten erstellt und verwaltet werden. Gespeicherte Jobs können automatisiert zu festgelegten Zeiten auf einzelne Ziele oder Gruppen gleichartiger Ziele ausgeführt werden. Benachrichtigungen über den Erfolg oder Misserfolg können versandt werden. Dieses zentrale Jobmanagement stellt eine gravierende Vereinfachung gegenüber dem manuellen Klonen von Jobs auf verschiedene Rechner dar. Es minimiert die Gefahr von Inkonsistenzen der auszuführenden Skripte drastisch.

Berichterstellung

Oracle hat eine umfangreiche Sammlung von Berichten erstellt, mit deren Hilfe sich detaillierte Informationen über die Zielsysteme auslesen lassen. Die Spanne reicht hier vom Anzeigen von Nutzern und Rollen mit bestimmten Berechtigungen über eine Verfügbarkeitshistorie eines Listeners bis zu Details der Speichernutzung in der SGA. Wem hier noch Berichte fehlen, für den besteht die Möglichkeit, mit Hilfe eines Werkzeugs eige-

ne Berichte zu generieren und zum Beispiel aus den Ergebnissen einer SQL-Abfrage Diagramme unterschiedlicher Art zu generieren. Der Phantasie sind hier kaum Grenzen gesetzt.

Fazit

Oracle stellt mit dem EM Grid Control ein mächtiges Monitoring- und Administrationswerkzeug zur Verfügung, das derzeit aber noch mit einigen Kinderkrankheiten kämpft. So ist zum Beispiel die Nutzerführung über die Web-Oberfläche teilweise recht inkonsistent. Als Beispiel sei hier das Aktualisieren der Browser-Ansicht genannt. Einige Seiten laden regelmäßig automatisch neu, andere müssen manuell über den Refresh Button des Browsers aktualisiert werden. Und einige zeigen nur nach einem Klick auf einen Refresh Button der Anwendung den aktuellen Stand.

Ähnlicher Wildwuchs herrscht auch bei der Anzahl der nötigen Bestätigungen, um eine geplante Aktion auszuführen. Diese Probleme dürften aber auf die frühe Version der Software zurückzuführen sein. Sie werden hoffentlich im Zuge der nächsten Releases verschwinden.

Der Funktionsumfang, besonders bezüglich des Monitorings und der Administration, ist bereits jetzt sehr umfangreich. Er ermöglicht ein zentrales Management vieler Server und kann sicher eine große Arbeitserleichterung darstellen. Allerdings kann auch ein Werkzeug wie Grid Control keine grundlegenden Kenntnisse der Materie ersetzen. Besonders die Tunings-Tipps sind ohne vertieftes Wissen mit größter Vorsicht zu genießen.

Speziell wegen der teils kostenpflichtigen Module ist auch bei der Konfiguration größte Vorsicht geboten, da ansonsten ein falscher Klick sehr teuer werden kann.

Wenn man sich dessen bewusst ist und im Idealfall sogar die Voraussetzungen für einen kostenlosen Einsatz gegeben sind, kann ein Blick auf dieses Softwarepaket durchaus lohnend sein.

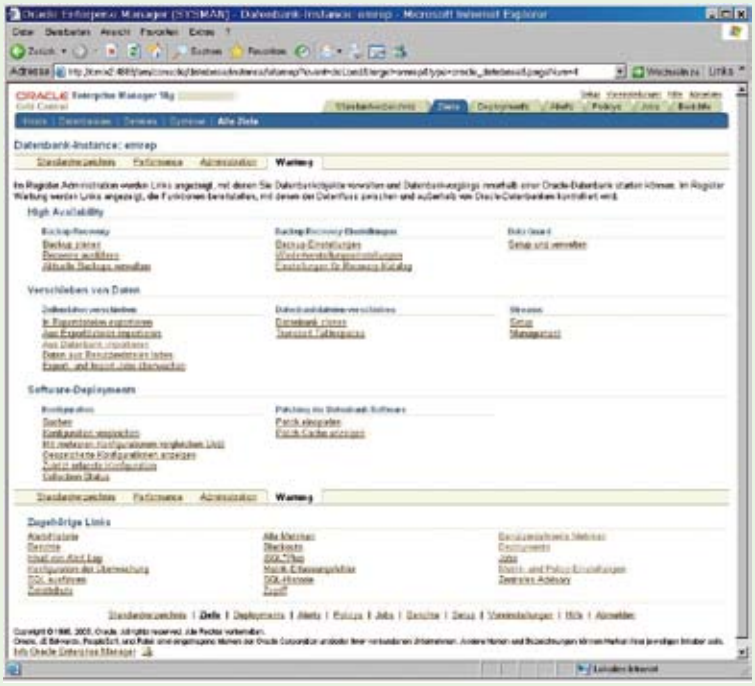


Abb. 5: Oracle stellt im EM Grid Control eine umfangreiche Werkzeugsammlung zur Wartung von Datenbanken bereit.

Glossar

Grid	Netzwerk aus gemeinsam genutzten Ressourcen
OMS	Oracle Management Server. Software von Oracle zur zentralen Administration mehrerer Oracle Datenbanken.
Repository	Datenbank zur Aufnahme von Metadaten über die Zielsysteme
Agent	Software, die auf den überwachten Systemen Daten sammelt.
EM Grid Control	Oracle Enterprise Manager Grid Control. Nachfolger des Oracle Management Server mit erweitertem Funktionsumfang. Ermöglicht über Plug-Ins auch Monitoring und Administration von Produkten anderer Hersteller.
SGA	System Global Area. SGA ist der von Oracle allokierte Hauptspeicher auf dem Datenbankserver.
Management Pack	(kostenpflichtiges) Modul des EM Grid Control

Ausblick

Nachdem dieser Artikel zunächst einen Überblick über die Funktionalität des EM Grid Controls gegeben hat, werden die folgenden Artikel dieser Serie sich jeweils mit einem Teilbereich der Software auseinandersetzen und dabei detaillierter auf den jeweiligen Funktionsumfang eingehen. Den Anfang wird ein Beitrag zum Thema „Installation und Konfiguration des Oracle Enterprise Mangers Grid Control“ machen.

Roman Peters (info@ordix.de).

Funktionen der Implementierung UCARP

UCARP: Hochverfügbarkeit für Linux und Unix

Dieser Artikel richtet sich an Systemadministratoren, die ein System hochverfügbar auslegen möchten.

Die redundante Auslegung von Systemen ist häufig die einzige Möglichkeit, geschäftskritische Anwendungen vor einem Ausfall zu bewahren. In den ORDIX News wurde schon häufiger über Heartbeat, das „Linux High Availability Projekt", berichtet. Dieser Artikel soll eine Alternative zu diesem Projekt aufzeigen und deren Funktionen erläutern.

Grundlegendes

UCARP ist eine portable Userland-Implementierung des Common Adress Redundancy Protocols (CARP), das vom OpenBSD Projekt entwickelt wurde. Dieses Protokoll zeichnet sich besonders durch den sehr geringen Overhead und die verschlüsselte Kommunikation aus. Diese Verschlüsselung wird mittels SHA-1 vorgenommen. Darüber hinaus wird für die Verbindung der Knoten keine dedizierte Verbindung benötigt, d. h. die Statusmeldungen der Systeme werden über das Netzwerk übertragen.

Die Hochverfügbarkeit wird gewährleistet, indem den Netzwerkschnittstellen zusätzlich zur physikalischen noch eine virtuelle IP-Adresse zugeteilt wird. Diese virtuelle IP-Adresse ist immer dem Master-System zugeordnet. Fällt es aus, wird das Backup-System zum Master. So ist der Cluster immer über eine IP-Adresse erreichbar. Dies ist notwendig, damit beim Ausfall eines Knotens für den Client keine Veränderungen auftreten.

CARP versus VRRP

Wenn man sich über CARP oder allgemein über die redundante Auslegung von Systemen informiert, wird man schnell auf ein weiteres Protokoll stoßen, das VRRP (Virtual Router Redundancy Protocol). Dieses Protokoll wird in vielen kommerziellen Routern zur Einrichtung eines Aktiv-Passiv-Clusters verwendet.

Doch wo liegen die Unterschiede zwischen diesen beiden Protokollen und weshalb sollte CARP eingesetzt werden?

- VRRP basiert auf Patenten der Firma CISCO. Um patentrechtlichen Problemen aus dem Weg zu gehen, sollte auf den Einsatz von VRRP verzichtet werden.
- CARP arbeitet im Gegensatz zu VRRP protokollunabhängig. Das bedeutet, es ist mit IPv4 und IPv6 nutzbar.
- Wie oben beschrieben, wird die gesamte Kommunikation von CARP verschlüsselt. Dies ist beim VRRP nicht der Fall.

Der Vergleich der beiden Protokolle zeigt, dass CARP dem VRRP nicht nur ebenbürtig, sondern sogar überlegen ist. Dies ist zum Teil dadurch zu begründen, dass die Entwickler von CARP ihr Protokoll anhand von VRRP entwickelt haben. So konnten Fehler des VRRP bei der Entwicklung von CARP vermieden werden.

Unterschiede zu Heartbeat?

In der Ausgabe 1/2006, Seite 19, haben wir über die zweite Version von Heartbeat berichtet. Seit dieser Version ist es möglich, mittels Heartbeat bis zu 16 Knoten in einem Verbund aufzunehmen. Vorher konnte ein Cluster nur aus zwei Knoten bestehen. Mit UCARP lassen sich ebenfalls mehrere Knoten zu einem

Cluster zusammenschließen. Getestet wurde von uns ein System mit einem Master- und zwei Backup-Servern. Es sollte aber problemlos möglich sein, weitere Slaves hinzuzufügen. Der Aufbau des von uns getesteten Systems wird im Laufe des Artikels noch genauer erläutert.

Der größte Vorteil von Heartbeat im Vergleich zu UCARP ist sein hoher Bekanntheitsgrad im Linux Umfeld. Auf Grund dessen sind wesentlich mehr Dokumentationen zu Heartbeat verfügbar. Dieses Manko soll dieser Artikel zu einem gewissen Grad ausgleichen.

Es muss aber von Fall zu Fall entschieden werden, welche Lösung bevorzugt wird. UCARP zeichnet sich durch seinen schlanken Aufbau und seine Stabilität aus, bietet aber nicht so viele Funktionen wie Heartbeat.

Beispiel

In vielen Fällen ist es sinnvoll, einen Server redundant auszulegen. Die Einrichtung eines solchen Clusters lässt sich am Besten anhand eines Beispiels beschreiben.

Der Proxy-Server ist in einem Unternehmen häufig der zentrale Punkt im Aufbau des Netzwerks und ein Ausfall dieses Servers führt schnell zu gravierenden Problemen. Somit ist ein Proxy-System, das doppelt ausgelegt ist, ideal für unser Beispiel. Um alle Möglichkeiten von UCARP zu erläutern, besteht unser Testaufbau aus einem Master- und zwei Slave-Systemen. Details sind im Netzplan dargestellt (siehe Abbildung 1).

Um das System, wie es auf dem Netzplan abgebildet ist, einzurichten, sind verschiedene Schritte notwendig, die im Folgenden genau erläutert werden. Für den Aufbau ist jedes Unix-Derivat denkbar. Drei installierte Systeme mit einem Squid-Server werden vorausgesetzt.

1. Installation

Als erstes muss das Programm z. B. von der offiziellen UCARP-Webseite [1] heruntergeladen werden. Im Anschluss daran kann UCARP auf allen Knoten kompiliert und installiert werden (`./configure && make && make install`).

Nach der Installation finden Sie UCARP unter `/usr/local/sbin/ucarp`. Ebenfalls wurde eine Manpage angelegt. Für weitergehende Informationen ist in dem aktuellen Verzeichnis die offizielle Dokumentation von UCARP in der Datei README abgelegt.

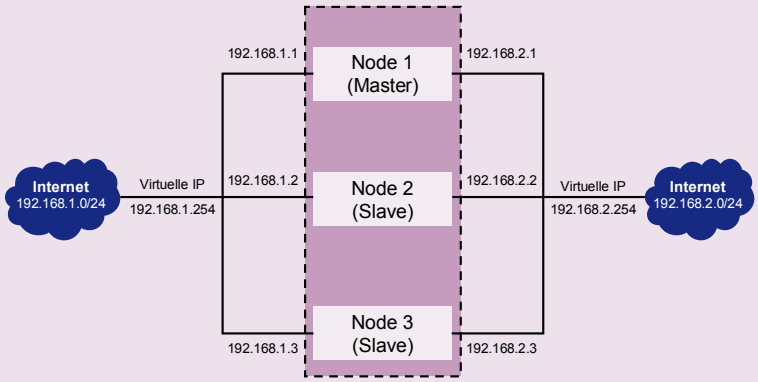


Abb. 1: Netzplan.

`--interface=<if> (-i <if>)`	An welches Interface wird UCARP gebunden.
`--srcip=<ip> (-s <ip>)`	Die „reale“ IP-Adresse des Hosts.
`--vhid=<id> (-v <id>)`	Die ID der virtuellen IP-Adresse (1-255), um mehrere virtuelle IP-Adressen in einem Subnetz zu ermöglichen.
`--pass=<pass> (-p <pass>)`	Das Passwort, mit dem die Verbindung verschlüsselt wird.
`--preempt (-P)`	Wenn dieses Flag gesetzt ist, wird dieser Rechner so schnell wie möglich Master.
`--addr=<ip> (-a <ip>)`	Die virtuelle IP-Adresse.
`--upscript=<file> (-u <file>)`	Dieses Skript wird ausgeführt, wenn der Knoten zum Master wird.
`--downscript=<file> (-d <file>)`	Dieses Skript wird ausgeführt, wenn der Knoten zum Slave wird.
`--advbase=<sec> (-b <sec>)`	Die Häufigkeit der Anfragen in Sekunden (default 1 Sekunde).
`--deadratio=<ratio> (-r <ratio>)`	Dauer, bis ein Host für tot gehalten wird.
`--daemonize (-B)`	Mit diesem Flag wird der Prozess im Hintergrund gestartet.
`--facility=<facility> (-f)`	Der Syslog-Daemon wird angegeben.

Abb. 2: Die wichtigsten Optionen von UCARP.

2. Konfiguration

UCARP verwendet normalerweise keine Konfigurationsdateien und legt auch keine Start- oder Stoppskripte an. Falls nicht auf Skripte

zur Steuerung des Prozesses verzichtet werden kann, müssen diese von Hand erstellt werden.

Das Verhalten von UCARP wird nur durch die Optionen beeinflusst, die beim Start mit angegeben werden. Es ist notwendig, mindestens die wichtigsten Optionen (siehe Abbildung 2) zu kennen.

Um einen einfachen Cluster mit zwei Rechnern aufzubauen, wären nur folgende Befehle notwendig:

- Knoten 1:
ucarp-v1-ptest-a192.168.1.254-s192.168.1.1

- Knoten 2:
ucarp-v1-ptest-a192.168.1.254-s192.168.1.2

Bei dieser Lösung haben beide Knoten dieselbe Priorität. Bei einem kompletten Ausfall des einen Systems, würde das andere seine Aufgabe übernehmen. Es wird aber kein Dienst oder ähnliches überwacht. Das bedeutet, wenn auf dem einen System der zu überwachende Dienst nicht mehr läuft, wird dies nicht erkannt und somit auch kein Wechsel der Knoten vorgenommen. In der Praxis ist so eine Konfiguration natürlich sinnlos.

Um genau solche Fehler auszuschließen und einen größeren Komfort zu erlangen, sind einige Skripte notwendig. Beispielhaft werden hier die Skripte für den ersten Knoten abgebildet. Auf den anderen Knoten laufen, mit geringen Abweichungen, dieselben Skripte.

Start- und Stoppskript

Um UCARP auf komfortable Weise starten und stoppen zu können, wird für unsere Lösung ein Skript benötigt, das genau diese

```
01 # Aufruf fuer eth0
02   /usr/local/sbin/ucarp -v 1 -p test -s 192.168.1.1 -a 192.168.1.254 -i eth0 \
03   -P -B -u /etc/ucarp/vip1-up.sh -d /etc/ucarp/vip1-down.sh
04   sleep 1
05   PID1=`ps -ef | grep /usr/local/bin/ucarp | grep -v grep | awk '{print $2}'` \
06   echo $PID1 >/var/run/ucarp1.pid
07
08 # Aufruf fuer eth1
09   /usr/local/sbin/ucarp -v 2 -p test -s 192.168.2.1 -a 192.168.2.254 -i eth1 \
10   -P -B -u /etc/ucarp/vip2-up.sh -d /etc/ucarp/vip2-down.sh
11   sleep 1
12   ps -ef | grep /usr/local/bin/ucarp | grep -v grep | grep -v $PID1 \
13   awk '{print $2}' >/var/run/ucarp2.pid
14
15 # Start des Ueberwachungsskripts
16   /usr/local/sbin/ucarp-check &
```

Abb. 3: Start- und Stoppskript für UCARP.

```
1  #!/bin/sh
2  # Skript zur Ueberwachung des Serverdienstes
3
4  SCHLEIFE=1
5  DIENST=squid
6
7  while [ $SCHLEIFE == 1 ]
8  do
9    ps -ef grep $DIENST | grep -v grep >/dev/null 2>&1 || { /etc/init.d/ucarp stop >/dev/null 2>&1; \
10   logger -i -t ucarp-check -- Der Serverdienst ist down. UCARP wurde beendet; 12   SCHLEIFE=0; }
11   sleep 1
12 done
```

Abb. 4: Skript zur Überwachung des Serverdienstes.

Funktionen übernimmt. Die wichtigsten Zeilen aus dem Startskript sind in Abbildung 3 zu sehen. Abbildung 2 erläutert die Funktionen der verschiedenen Optionen, mit denen UCARP aufgerufen wird.

Skript zur Überwachung des Serverdienstes

Falls der Squid nicht mehr läuft, wird im Normalfall kein Wechsel der Server durchgeführt. Das Internet ist nicht mehr erreichbar. Damit dies nicht passiert, wird ein Skript benötigt, welches die Überwachung übernimmt und UCARP bei einem Zwischenfall beendet. Das in Abbildung 4 aufgeführte Skript wird unter **`/usr/local/sbin/ucarp-check`** abgelegt und von dem Startskript (Abbildung 3, Zeile 16) aufgerufen. In unserem Skript wird nur überprüft, ob der Dienst noch läuft, also ob sein Prozess noch vorhanden ist. Man könnte an dieser Stelle auch kompliziertere Methoden implementieren, um die Funktionsfähigkeit des Serverdienstes zu überprüfen.

Die Skripte vip-up.sh und vip-down.sh

Diese Skripte werden ausgeführt, wenn der Knoten Master bzw. Slave wird. Für die beiden UCARP-Prozesse existieren jeweils zwei Skripte, in denen die virtuelle IP-Adresse einem Interface zugeteilt oder entsprechend wieder entzogen wird. In diesen Skripten lassen sich aber durchaus noch weitere Funktionen einbauen. Das Skript, welches aufgerufen wird, wenn der Knoten zum Master wird, benötigt in jedem Fall folgenden Befehl: **`/sbin/ip addr add <IP-Adresse> dev <Interface>`**.

Das Script ucarpctl

Für jemanden, der keine Erfahrung im Erstellen von Skripten hat, existiert eine gute Alternative. Das Programm ucarpctl-0.2 von Robert Woodcock [2] unterstützt die aktuelle Version 1.2 von UCARP. Das Programm ist in C geschrieben und eigentlich nur ein Skript zur Steuerung des UCARP-Prozesses. Es greift auf die Konfigurationsdatei (**`/etc/ucarp.conf`**) zu, in der alle Einstellungen zu UCARP vorgenommen werden. Die Datei für den ersten Knoten ist in Abbildung 5 zu sehen.

Die Möglichkeit, UCARP mittels ucarpctl zu starten, ist zwar auf den ersten Blick sehr komfortabel, es lassen sich aber nicht so leicht Änderungen am Startskript vornehmen. Das bedeutet, mit dieser Lösung wird man ohne entsprechende C-Kenntnisse im professionellen Einsatz schnell an die Grenzen stoßen.

```
common
  pass test
  preempt
  binarypath /usr/local/sbin/ucarp

virtip 192.168.1.254
  srcip 192.168.1.1
  interface eth0
  vhid 1
  upscript /etc/ucarp/vip-up1.sh
  downscript /etc/ucarp/vip-down1.sh

virtip 192.168.2.254
  srcip 192.168.2.1
  interface eth1
  vhid 2
  upscript /etc/ucarp/vip-up2.sh
  downscript /etc/ucarp/vip-down2.sh
```

Abb. 5: Konfiguration von ucarpctl.

Links

- [1] www.ucarp.org
- [2] http://freshmeat.net/projects/ucarpctl/

Glossar

VRRP	Virtual Router Redundancy Protocol. Mit Hilfe des VRRP lassen sich IP-Systeme in einem Netzwerk redundant auslegen, indem einer Gruppe von Knoten eine virtuelle IP-Adresse zugeteilt wird.
CARP	Common Address Redundancy Protocol. Das CARP wurde vom OpenBSD-Team entwickelt und stellt eine patentfreie Alternative zum VRRP dar.
UCARP	Ist eine portable Userland-Implementierung des CARP.

Fazit

UCARP bietet alle grundlegenden Funktionen, die man benötigt, um einen Aktiv-Passiv-Cluster aufzubauen. Die Konfiguration ist im Gegensatz zu Heartbeat trivial, bietet allerdings nicht so viele Möglichkeiten. So wurden im Großen und Ganzen alle Funktionen, die UCARP bietet, durch das Beispiel in diesem Artikel abgedeckt. Wem dieser Funktionsumfang ausreicht und wen die etwas spärliche Dokumentation nicht stört, der findet in UCARP eine gute Alternative zu den gängigen Lösungen.

Marius Dorlöchter (info@ordix.de).

Oracle Zugriffskontrolle: Fine Grained Access Control

Der Artikel richtet sich an Datenbankadministratoren die sich mit dem Schutz bzw. der Kontrolle des Zugriffs auf firmensensitive Daten beschäftigen.

Der Artikel beschäftigt sich mit der Funktionalität der fein granulierten Zugriffskontrolle (FGAC) unter Oracle. Sie hilft dem Datenbankadministrator bei der Auswahl firmensensitiver Daten für einzelne Benutzer im Hinblick auf die Erhöhung der Sicherheit bei der Bearbeitung. Zunächst geben wir einen kurzen Überblick über die bisher zur Verfügung stehenden Möglichkeiten von Sicherheitsmechanismen. Danach stellen wir die fein granulierte Zugriffskontrolle und ihre neuen Einsatzmöglichkeiten ab Oracle 10g Release 2 vor.

DOAG Deutsche ORACLE-Anwendergruppe e. V. News
Ebenfalls veröffentlicht in der DOAG News 2/2007
Die Zeitschrift der DOAG-Mitglieder

Der Standardmechanismus der Zugriffskontrolle auf Tabellendaten wird über die Vergabe von Objektprivilegien (SELECT, INSERT, UPDATE, DELETE) an die einzelnen Benutzer oder über die Zuweisung von Benutzerrollen gesteuert. Damit ist nur eine statische Zugriffskontrolle auf die einzelne Tabelle oder die einzelne View gewährleistet.

Sobald der Benutzer das entsprechende Privileg besitzt, kann er auf sämtliche Datensätze dieser Tabelle zugreifen. Häufig ist es jedoch notwendig, dass der Benutzer nur auf die für ihn relevanten Daten zugreifen darf, z. B. beim Internet Banking.

Oftmals wird dies über die Bereitstellung von Views gelöst. Die Benutzer erhalten die Zugriffsrechte auf die Daten der Basistabellen nur über die Views. Die Nachteile dieser Lösung sind:

- fehlende Dynamik
- eine große Anzahl und Vielfalt von Views
- Administration über DDL-Kommandos

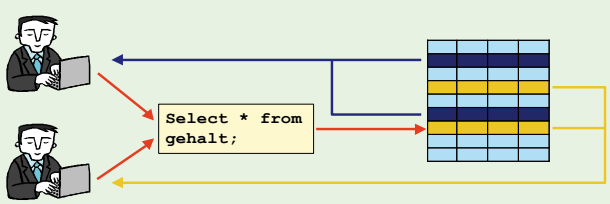


Abb. 1: Informationsgewinnung mittels Fine Grained Access Control (FGAC).

Fine Grained Access Control

Hier greift nun Fine Grained Access Control (FGAC): Über FGAC wird dem Benutzer sozusagen eine „Virtual Private Database“ (VPD) zur Verfügung gestellt. Damit kann er nur noch die Datensätze sehen und manipulieren, auf die er entsprechend der definierten Sicherheitsrichtlinien („Policys“) zugreifen darf.

Der Zugriff auf die relevanten Daten wird in der Datenbank selbst gesteuert. Diese Funktion steht ab der Version 8i zur Verfügung. Dies bedeutete bisher, dass nur die einzelne Zeilenebene betrachtet wird. Seit dem Release Oracle 10g R2 ist nun die Möglichkeit der spaltenweisen Betrachtung hinzugekommen.

Vorgehensweise

Für eine spaltenweise Betrachtung ist es notwendig, eine so genannte Policy als Regel-

werk an eine Basistabelle zu binden. Diese Policy beinhaltet wiederum eine Prädikatsfunktion, mit deren Hilfe gedanklich eine Erweiterung der WHERE-Klausel des entsprechenden DML- bzw. SELECT-Statements vorgenommen wird. Diese Prädikatsfunktion kann neuerdings über PL/SQL individuell an die einzelnen Anforderungen der spaltenweisen Betrachtung angepasst werden.

Policy

Die Policy wird mit Hilfe des Datenbank-Packages DBMS_RLS erstellt. Zum Hinzufügen nutzt man die Prozedur ADD_POLICY. Die einzelnen Parameter beinhalten die Informationen in Abbildung 2. Hinzu kommen noch die Einzelprozeduren der Abbildung 3. Darüber hinaus existieren auch Gruppenprozeduren, die dann auf bestimmte Policy-Gruppen wirken.

Dynamische Prädikatsfunktion

Die Policy-Funktion bezeichnet man auch als dynamische Prädikatsfunktion, weil sie erst in der Parse-Phase des eingegebenen SELECT oder DML-Statement implementiert wird und dann in der Ausführungsphase die relevanten Informationen liefert. Sie stellt eine WHERE-Klausel bereit, mit der die entsprechende Zugriffsregel definiert wird. Das Beispiel in Abbildung 4 gibt einen Überblick über die Erstellung der Prädikatsfunktion. Abbildung 5 zeigt die Erstellung einer Policy mit Prädikatsfunktion.

Spaltenbasierte Beschränkung bzw. Spaltenmaskierung

Die mit Release 2 eingeführte Modifikation der spaltenbasierten Beschränkung beruht auf der Möglichkeit, die einzelnen, sicherheitsrelevanten Spalten zu bezeichnen. Damit werden nur die Datensätze der markierten sicherheitsrelevanten Spalten angezeigt, für die eine Zugriffsberechtigung besteht. Somit sind nur diejenigen Spaltenwerte zu sehen, auf die der aktuelle Benutzer zugreifen darf. Die übrigen Werte der sicherheitsrelevanten Spalten werden mit NULL-Werten angezeigt. Abbildung 6 zeigt einen entsprechenden Aufruf und die zugehörige Ausgabe.

Kontextsensitiver Policy Typ

Ebenso kann man nun eine Policy im sessionbezogenen Kontext erstellen. Die Prädikats-

OBJECT_SCHEMA	**Benutzer**
OBJECT_NAME	Tabellen- oder Viewname
POLICY_NAME	Name der Policy
FUNCTION_SCHEMA	Schema der verwendeten Policy-Funktion
POLICY_FUNCTION	Name der zu verwendenden Policy-Funktion
STATEMENT_TYPES	Geltungsbereich (SELECT, INSERT, UPDATE, DELETE)
UPDATE_CHECK	Kennung für eine Aktualisierungsprüfung (TRUE/FALSE)
ENABLE	Kennung für den aktiven Zustand (TRUE/FALSE)
STATIC_POLICY	Zuordnung zu allen Benutzern (TRUE/FALSE) außer SYS
POLICY_TYPE	Statische Policy oder definierter Typ
LONG_PREDICATE	Zeichenlänge der Prädikatsfunktion FALSE <= 4 KB TRUE > 4 KB bis 32 KB
SEC_RELEVANT_COLS	Definition spaltenbasierter VPD
SEC_RELEVANT_COLS_OPT	Maskierung der Werte von spaltenbasierten VPD

Abb. 2: Parameter der Prozedur ADD_POLICY aus dem DBMS_RLS Package.

DROP_POLICY	Löschen einer vorhandenen Policy
REFRESH_POLICY	Alle gecachten Statements, die mit der Policy zusammenhängen, werden erneut geparst. Dadurch hat die letzte Änderung der Policy direkt Auswirkungen, sobald die Funktion aktiviert wird.
UPDATE_CHECK	Erzwingt nach einer Zeilenänderung eine erneute Überprüfung der Funktion.
ENABLE_POLICY	Aktivieren und Deaktivieren einer Policy. Default ist ENABLE beim Erstellen der Policy.

Abb. 3: Weitere Einzelprozeduren aus dem DBMS_RLS Package.

```
CREATE OR REPLACE FUNCTION meier.mitarbeiter_sec
   (schema IN varchar2, tab IN varchar2)
RETURN VARCHAR2 AS
BEGIN
   RETURN 'MITARBEITERNAME='''||
      sys_context('userenv', 'session_user') ||'''';
END mitarbeiter_sec;
/
SELECT mitarbeitername,gehalt,abteilungsnr FROM mitarbeiter
   WHERE MITARBEITERNAME = (sys_context('userenv', 'session_user') ;
```

Abb. 4: Erstellung der Prädikatsfunktion mit dem daraus resultierenden SELECT.

```
EXECUTE DBMS_RLS.ADD_POLICY (object_schema => ' MEIER ',
   object_name => 'MITARBEITER', policy_name => 'MA_POLICY',
   function_schema =>'MEIER', policy_function => 'MITARBEITER_SEC',
   statement_types => 'SELECT');

select mitarbeiternr,mitarbeitername,gehalt from MEIER.mitarbeiter;
```

Abb. 5: Erstellung einer Policy mit Prädikatsfunktion.

```
EXECUTE DBMS_RLS.ADD_POLICY (object_schema => ' MEIER ',
   object_name => 'MITARBEITER', policy_name => 'MA_POLICY',
   function_schema =>'MEIER', policy_function => 'MITARBEITER_SEC',
   statement_types => 'SELECT',  sec_relevant_cols => 'GEHALT',
   sec_relevant_cols_opt=>DBMS_RLS.ALL_ROWS);

select mitarbeiternr,mitarbeitername,gehalt from MEIER.mitarbeiter;

Ausgabe:
MITARBEITERNR   MITARBEITERNAME    GEHALT
1               Dr. Klose
11              Schroeder
155             MEIER              20000
```

Abb. 6: Erstellung einer Policy mit Prädikatsfunktion und zusätzlicher, spaltenbasierter Beschränkung.

```
CREATE CONTEXT mitarbeiter_ctx USING ora00.mitctx;
CREATE OR REPLACE PACKAGE ora00.mitctx AS
   PROCEDURE set_mitarbeiternr_prc;
END;
/
CREATE OR REPLACE PACKAGE BODY ora00.mitctx IS
   PROCEDURE Set_mitarbeiternr_prc IS mit_name VARCHAR2(200);
BEGIN
   SELECT mitarbeitername INTO mit_name FROM ora00.mitarbei-
ter
      WHERE mitarbeitername = SYS_CONTEXT('USERENV','SESSION_USER');
   DBMS_SESSION.SET_CONTEXT ('MITARBEITER_CTX', 'MA_NO', mit_name);
END SET_MITARBEITERNR_PRC;
END;
/
BEGIN
   mitctx.set_mitarbeiternr_prc;
END;
/
CREATE OR REPLACE PACKAGE ora00.mitctx AS
   PROCEDURE set_mitarbeiternr_prc;
END;
/
```

Abb. 7: Erstellen eines Kontextes.

```
CREATE OR REPLACE PACKAGE ora00.mit_security AS
FUNCTION mitarbeiter_sec (schema IN VARCHAR2, tab IN VARCHAR2)
RETURN VARCHAR2;
END;
/
CREATE OR REPLACE PACKAGE BODY ora00.mit_security AS
FUNCTION mitarbeiter_sec (schema IN VARCHAR2, tab IN VAR-
CHAR2)
RETURN VARCHAR2 IS v_mit VARCHAR2(200);
BEGIN
   v_mit := 'mitarbeitername = SYS_CONTEXT(''MITARBEITER_CTX'',''MA_NO'')';
   RETURN v_mit;
END mitarbeiter_sec;
END mit_security;
/
```

Abb. 8: Erstellen der Prädikatsfunktion, die nun den neu erstellten Kontext nutzt.

funktion wird dabei bezogen auf existierende Kontextwerte zusammengestellt [z. B. SYS_CONTEXT ('USERENV', 'SESSION_USER')].

Dabei wird die Prädikatsfunktion nach dem Parsen des SQL-Statements entsprechend des gesetzten Kontextes benutzt. Dies ist dann sinnvoll, wenn mit mehr als einem Benutzer oder verschiedenen Benutzergruppen gearbeitet wird.

Dazu wird zuerst der entsprechende Kontext definiert. Der zugrunde liegende Kontext, in dem der Benutzer arbeitet, sollte durch einen Logon-Trigger definiert sein. Der Kontext selbst stellt dabei ein über die gesamte Session gültiges Attribut/Wertepaar dar. Dies gilt ab Oracle 10g Release 2 mit der Funktionalität SYS_CONTEXT auch bei einer parallelen Abfrage.

Bei der kontextsensitiven Erstellung einer Policy braucht man die Prädikatsfunktion nicht erneut anzupassen (siehe Abbildung 7). Das kommt dem Ziel der Steigerung der Performance entgegen.

Im Anschluss wird, wie schon in den vorhergehenden Releases, eine Prädikatsfunktion erstellt, die nun den neu erstellten Kontext benutzt (siehe Abbildung 8). Vorteilhaft ist, dass diese nun nicht neu angepasst werden muss, sondern immer für die durch den Kontext betroffenen Benutzer Gültigkeit besitzt.

Im letzten Schritt (Abbildung 9), erstellen wir nun die Policy, die die Prädikatsfunktion benutzt. Als Ergebnis dieser kontextbasierenden Prädikatsfunktion ergibt sich bei dem User ORA00, dass bei einem SELECT-Zugriff nur seine relevanten Daten aus der Tabelle Mitarbeiter angezeigt werden (siehe Abbildung 10).

Fazit

Fine Grained Access Control bietet gegenüber den herkömmlichen Verfahren die folgenden Vorteile:

- Weniger Administrationsaufwand für die Datenbankadministratoren durch weniger Views
- Ausschluss überflüssiger Informationen (Information Overload/Zeitmanagement) für den einzelnen Benutzer
- Das FGAC kann jederzeit ohne weitere Lizenzkosten implementiert werden.

Glossar

Prädikatsfunktion	Funktion zur Selektion von Daten.
FGAC	Fine Grained Access Control. Fein granulierte Zugriffskontrolle.
DML	Data Modification Language. Über DML-Kommandos werden Daten (Zeilen) eingefügt, gelöscht oder geändert.
DDL	Data Defnition Language. Über DDL-Kommandos werden Datenstrukturen gepflegt (z. B. Tabellen anlegen oder löschen).
VPD	Virtual Private Database. „Eigene“ virtuelle Datenbank.
Policy	Sicherheitsregel
Package	Stellt eine Sammlung logisch zusammengehöriger Funktionalitäten (Prozeduren, Funktionen, Typdeklarationen etc.) dar.

- Durch die innerhalb der Datenbank integrierte Funktionalität ist ein programmierbares Umgehen von Sicherheitsrichtlinien unmöglich.
- Das FGAC erweitert die Möglichkeiten der Zugriffskontrolle durch zeilen- und spaltenbasierte Datenselektion.

Diese nun feiner zu granulierende Informationsgewinnung ermöglicht dem Datenbankadministrator, dass er den Entscheidern keine überflüssigen Informationen mehr zur Verfügung stellt, die diese gar nicht haben sollen oder wollen. Das Fine Grained Access Control definiert nun eindeutig, von wem welche sensitiven Informationen innerhalb der Datenbank genutzt werden dürfen und welche nicht.

Klaus Günther (info@ordix.de).

```
BEGIN
DBMS_RLS.ADD_POLICY (object_schema => 'ORA00',
  object_name => 'MITARBEITER', policy_name => 'MIT_POLICY',
  function_schema => 'ORA00',
  policy_function => 'MIT_SECURITY.MITARBEITER_SEC',
  statement_types => 'SELECT');
  --, sec_relevant_cols => 'GEHALT',
sec_relevant_cols_opt=>DBMS_RLS.ALL_ROWS)
END;
/
```

Abb. 9: Erstellen der Policy, die die kontextbasierende Prädikatsfunktion nutzt.

```
select mitarbeitername, gehalt from mitarbeiter;

MITARBEITERNAME    GEHALT
ORA00              10000
```

Abb. 10: Ausgabe der Abfrage auf die Tabelle Mitarbeiter.

Seminar Oracle PL/SQL Aufbau mit LOB Programmierung

Diese Seminar richtet sich an fortgeschrittene Entwickler und Datenbankprogrammierer, die ihre PL/SQL-Kenntnisse insbesondere im LOB-Umfeld erweitern möchten.

Zielgruppe

Entwickler und Administratoren, die LOB-Kenntnisse erlangen bzw. vertiefen möchten.

Voraussetzungen

Kenntnisse in SQL und PL/SQL oder Teilnahme am Seminar „Oracle SQL“ und „Oracle Datenbankprogrammierung mit PL/SQL“.

Dauer: 3 Tage

Preis: 1.190,00 € zzgl. MwSt.

Inhalte

- PL/SQL-Designkonstrukte: Cursor-Variablen, Subtypen, Objekttypen
- Kollektionen (SQL/PL/SQL): assoziative Arrays, Nested Tables, Varrays, Vorteile von Kollektionen
- Fine Grained Access Control (FGAC): Prozessbetrachtung, Anwendungskontexte einrichten, Policies implementieren
- Externe Prozeduren: Vorteile externer Prozeduren, Komponenten externer Prozeduren, Aufruf externer Prozeduren
- PL/SQL Server Pages: Verwendungszweck, Aufbau und Aufruf
- Analyse- und Performance-Tricks
- LOBs: Definition von Datentypen, LOB in der Datenbank, Nutzung des DBMS_LOB Packages, Vorteile von LOBs

Termine

11.06.2007 - 13.06.2007 in Wiesbaden
17.09.2007 - 19.09.2007 in Wiesbaden
26.11.2007 - 28.11.2007 in Wiesbaden

Projektmanagement

… und Reden ist Gold

Dieser Artikel richtet sich an Projektmanager bzw. an Personen, die Projektmanager werden wollen.

Eigentlich müsste in unserer Branche, der Informations- und Kommunikationstechnologie, das Thema „Kommunikation in der Projektarbeit“ bedeuten, Eulen nach Athen zu tragen. In der Praxis zeigt sich aber vielfach, dass der Ausspruch „der Schuster hat die schlechtesten Schuhe“ leider zutreffender ist. Dabei ist eine effiziente und zielorientierte Kommunikation ein wesentlicher Erfolgsfaktor.

Kommunikation ist eine Schlüsselfunktion in der Projektarbeit

Der Erfolg eines Projektes hängt in weiten Bereichen von der Qualität und Quantität der Kommunikation ab. Hierbei sind folgende drei Schwerpunktbereiche zu berücksichtigen:

- Interner Informationsaustausch: damit Abstimmungsprozesse effizient durchgeführt, Entscheidungen schnell getroffen und Besprechungen ergebnisorientiert geführt werden.
- Informationsmanagement: damit alle Projektbeteiligten Zugriff auf alle Informationen haben, die sie für die erfolgreiche Durchführung des Projektes brauchen.
- Projektmarketing: damit das Projekt auf eine breite Unterstützung, insbesondere bei den Stakeholdern des Projektes (Mitarbeiter, Auftraggeber, zukünftige Nutzer, etc.) trifft.

Ist Projektmanagement auch Kommunikationsmanagement?

Ja! Vor allem zu Beginn eines Projektes ist es eine der wichtigsten Aufgaben des Projektleiters, sich über seine Kommunikationsstrategie im Klaren zu werden. Denn Kommunikation kann bis zu 50 Prozent der Projektarbeit bei den Beteiligten binden. Bei der Festlegung der Strategie stehen folgende Aspekte im Vordergrund:

- Welche Ziele sollen durch Kommunikation erreicht werden?
- Welche Zielgruppen mit welchen Anforderungen bzw. Bedürfnissen nach Information und Kommunikation müssen adressiert werden?
- Welche Kommunikationsmedien und -wege müssen genutzt werden, um o. g. Ziele und Zielgruppen zu erreichen?

Der Projektleiter sollte sich in der Vorbereitungsphase auch über die notwendigen Prioritäten der Kommunikationsziele klar werden: Geht es besonders um Effizienz der Projektarbeit oder um Vermeidung von Unruhe durch Informationsdefizite bei den Stakeholdern des Projekts? Wenn die Kommunikationsziele geklärt sind, ergeben sich auch schnell die Kommunikationsinhalte.

Die richtige Kommunikationsmenge

Besonders wichtig ist, dass von Anfang an ein gemeinsames Verständnis der Verantwortungszuordnung, Rollenzuteilung und Zusammenarbeitsregeln im Projekt erreicht wird. Jeder Beteiligte muss inhaltlich ausreichend informiert sein und darf sich aufgrund der Informationsmenge weder überfrachtet, noch ausgegrenzt fühlen.

Zuviel Kommunikation kann allerdings auch ein Projekt erheblich bremsen. Der Kommunikationsaufwand steigt exponentiell mit der Anzahl der Partner (Teilnehmer):

Die Anzahl B der Kommunikationsbeziehungen zwischen N Team-Mitgliedern verhält sich nach der Formel:

B = N * (N-1) / 2

Dies ist insofern wichtig zu beachten, als dass die erforderliche Kommunikationsmenge gerade zu Beginn eines Projekts besonders hoch ist.

Kommunikation ist wichtig – gerade am Anfang

Insbesondere in der Startphase ist die direkte Kommunikation im Projektteam wichtig, um sich kennen zu lernen, Vertrauen zu entwickeln, ein gemeinsames Verständnis von Zielen, Vorgehen und Zusammenarbeit im Projekt zu erreichen. Im Laufe der Zeit kann dann der Informationsaustausch zunehmend elektronisch erfolgen, wenn die gemeinsame Basis aufgebaut ist. Ein wöchentlicher Jour Fixe, sei es als Meeting oder als Telefonkonferenz, sollte allerdings auch im eingeschwungenen Zustand, wie der Name schon sagt, regelmäßig an einem festen Termin stattfinden. Für den Jour Fixe sollte sichergestellt werden, dass ausreichend Zeit eingeplant wird und es einen festgelegten Moderator gibt, damit am Ende nicht das Wichtigste zu kurz kommt, nämlich die Ergebnisabstimmung und klare Aufgabenzuordnungen.

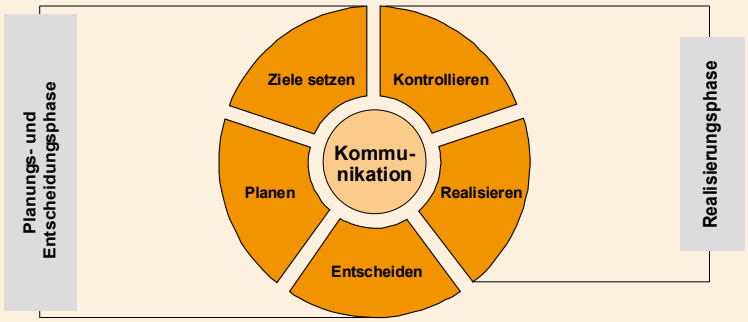


Abb. 1: Die Aufgaben des Projektmanagements sind in 2 Hauptphasen unterteilbar.

Asynchronous I/O

Eigentlich ist „kernelized asynchronous Input/Output“ eine Eigenschaft eines Unix-Systems. Asynchrone Kommunikation ist aber ebenso ein wichtiger Bestandteil der Kommunikationsstrategie in einem Projekt.

Häufig passiert es, dass nicht immer alle Beteiligten gleichzeitig erreichbar sind. Daher ist eine asynchrone Kommunikation unabdingbar. Eine kontinuierliche Kommunikation kann dann nur durch E-Mails gewährleistet werden. Ganz davon abgesehen, dass damit automatisch die Nachvollziehbarkeit des Informationsflusses deutlich erhöht wird.

Was ist nun Silber, was ist Gold?

Reden oder schreiben? Natürlich ist es einfach, schnell und bequem, E-Mails zu schreiben. Vor allem, wenn der Empfänger momentan nicht erreichbar ist. Damit wird im Allgemeinen auch unterstellt, dass der Adressat nun informiert ist. Aber ist er es wirklich? Wann liest er seine Mails? Wann und wie bekommen wir eine Antwort? Mitunter kommt die Antwort sogar recht zeitnah. Darauf wird wieder geantwortet usw. Da ist schnell ein Schwellwert erreicht, bei dem diese asynchrone Kommunikation nicht mehr effektiv ist. Spätestens nach der dritten Mail, die hin und her geht, ist eine direkte Kommunikation, persönlich oder am Telefon, sinnvoller.

Wer suchet, der findet.

Weiterhin ist zu beachten, dass bei Anfragen zu bestimmten Themen oder wenn Aufträge zu kommunizieren sind, die Schriftform – hier

Glossar

Stakeholder	Englischer Begriff für Interessensvertreter. Im Bereich Projektmanagement sind das alle Personen oder Parteien, die ein berechtigtes Interesse an dem Projekt haben, z. B. Projektmitarbeiter, Auftraggeber, zukünftige Nutzer, etc.
Kernelized asynchronous Input/Output	Eigentlich eine Eigenschaft eines Unix-Systems. In diesem Zusammenhang ist die asynchrone Kommunikation, z. B. über E-Mail gemeint.
WIKI	Ein web-basiertes Informationssystem, in das jeder sein Wissen einbringen kann – wie z. B. bei WIKIPEDIA

E-Mail – die einzig sinnvolle, weil nachvollziehbare Art der Kommunikation im Projekt ist. Deswegen müssen diese Mails strukturiert abgelegt werden. Betreffs sind so zu wählen, dass solche Mitteilungen später rasch gefunden werden können.

Es gibt Projektmanager, die glauben, mit der Schaffung einer elektronischen Ablage sei das Thema „Kommunikation im Projekt" komplett erledigt. In der Tat ist eine gut strukturierte Projektablage essentiell für das Informationsmanagement eines erfolgreichen Projekts. Dabei ist es nicht nur von Bedeutung, Informationen so abzulegen, dass man sie auch wiederfindet, sondern auch, dass deren Aktualität gewährleistet ist. Und die Frage „Wie erfahren Projektmitarbeiter, wann etwas abgelegt wurde?" muss selbstverständlich auch geklärt sein.

Es ist wirklich erstaunlich, aber wahr: Wir erleben immer wieder als Berater, dass in Projekten weder die Nomenklatur der Projektablage noch deren Verwendungsregeln festgelegt sind.

Versuchen Sie doch mal, vorangestellte Ziffern bei der Benennung Ihrer Ordner in Windows-Verzeichnissen einzuführen. Auch der Einsatz von WIKIs wird zu Recht immer beliebter.

Information: Hol- oder Bringschuld?

Was muss der Projektleiter liefern, was kann und darf er einfordern? In welchen Intervallen und in welchem Umfang müssen Informationen verteilt werden?

Die Frage nach der Hol- oder Bringschuld ist dabei so alt, wie das Thema Kommunikation selbst. Völlig fehl am Platz ist hier das Anspruchsdenken. Informationslücken und Spannungen entstehen, wenn der eine sagt, dass hätte man sagen sollen und der andere meint, man hätte ja fragen können. Daraus sollte kein „Henne/Ei-Problem" gemacht werden. Vermeiden Sie als Projektleiter den Eindruck, Sie würden etwas verbergen oder verheimlichen.

Jedoch muss weder jedes Thema, noch jedes Detail immer aktiv kommuniziert werden. Mitunter genügt es, Informationen zu hinterlegen – entweder in einer Ablage für E-Mails oder auf einer Intranet-Seite.

Die Qual der Wahl des richtigen Mediums

Sie denken: „Intranet, das ist perfekt!" Man kann alles ablegen, jeder kann lesen, wann immer er will. Präsentationen, Dokumente, einfache Texte wie Notizen. Einfach alles ablegen? Gut, jetzt haben wir zwar vergessen, den anderen Bescheid zu geben, dass es dort etwas Neues zu lesen gibt, aber sonst perfekt!

Nicht ganz. Auch hier ist darauf zu achten, wer die Zielgruppe ist und wie groß sie ist. Ein überschaubares Projektteam braucht für sich keine Intranetseite. Eher ein WIKI für schnelle Informationsbeschaffung und als Grundlage für ein einfaches Knowledge Management. Ein Projekt, das eine große Zielgruppe (z. B. bei einem Plattformwechsel) besitzt, hat gar keine andere Wahl, als die asynchrone Kommunikation per Web.

Was noch wichtiger ist: Auf eine gute Präsentation kann kein Projekt verzichten. Daher achten Sie am besten bereits bei der Projektplanung darauf, dass Sie ausreichend Ressourcen dafür planen. Und was nützen die schönsten Folien? Es muss dazu gesprochen werden. Reden ist eben Gold.

Kommunikation ist auch Marketing

Der Projektmanager ist der entscheidende Kommunikator seines Projektes und ist damit für das Projektmarketing verantwortlich. Er „vermarktet" sein Projekt intern und extern. Eine gute Vermarktung durch zielgruppengerechte Information erhöht nachweislich die Kundenzufriedenheit und die Motivation aller Beteiligten im Projekt. Der „gefühlte" und der tatsächliche Erfolg kann dadurch wesentlich gesteigert werden.

Ein Projektmanager sollte sein Projekt vor allem auch in kritischen Phasen gut verkaufen können. Und wenn vorher alle Informationen systematisch verteilt und strukturiert abgelegt wurden, kann der Projektleiter zu jedem Zeitpunkt treffsicher darauf zugreifen.

Fazit

Viele Projektmanager unterschätzen die Kommunikation als wichtigen Erfolgsfaktor in ihrem Projekt. Sie konzentrieren sich zu sehr auf die fachliche und technische Herausforderung. Sie laufen dabei Gefahr, dass sich die Stakeholder des Projektes nicht einbezogen fühlen und dass das Projektteam nicht effizient und zielgerichtet arbeiten kann. Kommunikation ist wichtig – heute wichtiger denn je. Da unsere Branche von der wachsenden Bedeutung der Kommunikation lebt, sollte sie eigentlich auch in der Projektarbeit Vorbild sein.

Benedikt Georgi, Rainer Restat (info@ordix.de).

Seminar IT-Projektmanagement

Der Teilnehmer erhält einen umfassenden, systematischen Überblick über alle Begriffe, Methoden und Arbeitstechniken zum Management von IT-Projekten. Er lernt, IT-Projekte zu planen, zu steuern und verfügbare Arbeitshilfen projektspezifisch anzupassen.

Zielgruppe

Führungskräfte aus Fachabteilungen, IT-Koordinatoren, IT-Revisoren, Organisatoren, IT-Führungskräfte, angehende Projektleiter, erfahrene Entwickler und Systemspezialisten.

Voraussetzungen

Vorteilhaft sind erste praktische Erfahrungen mit Projektarbeit.

Dauer: 5 Tage **Preis:** 1.890,00 € zzgl. MwSt.

Termine

30.07.2007 - 03.08.2007 in Wiesbaden
15.10.2007 - 19.10.2007 in Wiesbaden

Inhalte

- **Grundlagen:** Projektdefinition, Projektmanagement, Phasen- und Vorgehensmodelle
- **Projektvorbereitung:** Voruntersuchung, Projektantrag, Projektauftrag, Projekthandbuch, Projektorganisation, Gremien, Instanzen, Prozesse, Berichtswesen, Eskalationsverfahren
- **Projektplanung:** Leistungsumfang, Arbeitsaufträge, Arbeitsergebnisse, Mitarbeitereinsatz, Qualität, Termine, Budgets, Ressourcen, Risiken, Planungstechniken, Qualifikationsprofile, Projektkategorien
- **Projektdurchführung:** Ergebnisdokumentation, Abnahme, Zusammenarbeit mit Auftraggeber/Fachabteilungen, Projektsitzungen, Konfliktbewältigung, Projektberichte, Hilfsmittel, Informationswesen, Projektmarketing
- **Projekt-Controlling:** Regelkreise, Datenerfassung, Datenanalyse, Wirtschaftlichkeitsrechnungen, vor-/mit-/nachlaufende Kalkulation, Terminüberwachung, Budgetüberwachung, Qualitätsüberwachung, Eskalationsverfahren, Audits/Reviews, Statusberichte, Projektbüro
- **Qualitäts- und Risikomanagement:** Prozessqualität, Ergebnisqualität, Risikoanalyse, Risikobewertungen
- **Arbeits- und Hilfsmittel:** Formulare, Checklisten, Software
- Fallbeispiele, Übungen

Seminar Grundlagen des IT-Controlling

Der Teilnehmer erhält einen Überblick über Controlling-Konzepte und -Methoden speziell für den IT-Bereich und lernt, Arbeitstechniken und Vorgehensweisen auszuwählen und im IT-Umfeld einzusetzen.

Zielgruppe

IT-Führungskräfte und -Projektmanager, IT-Experten, die Controlling-Aufgaben übernehmen sollen, für den IT-Bereich zuständige Unternehmenscontroller.

Voraussetzungen

Kenntnisse/Erfahrungen im Bereich der IT, Grundkenntnisse aus Buchhaltung und Rechnungswesen.

Dauer: 3 Tage **Preis:** 1.190,00 € zzgl. MwSt.

Termine

23.07.2007 - 25.07.2007 in Wiesbaden
08.10.2007 - 10.10.2007 in Wiesbaden

Inhalte

- **Grundlagen des Controlling:** Regelkreise, Controlling-Verantwortung, Kennzahlen, Kennzahlensysteme, Benchmarking, Kostenstellen, Kostenarten, Kostenträger
- **Prozess-Controlling:** Budgetierung, Total Cost of Ownership (TCO), Prozesskostenrechnung, Qualitätssicherung, Risikosteuerung, Ressourcen-Steuerung, Leistungsaufschreibungen
- **Produkt-Controlling:** Leistungsverrechnung, Preisbildung bei IT-Leistungen, Service-Level-Agreements, Lebenszyklussteuerung
- **Projekt-Controlling:** Wirtschaftlichkeitsrechnungen, vor-/mit-/nachlaufende Kalkulation, Multiprojekt-Controlling, Programm-Controlling
- **Strategisches Controlling:** Vision, Strategie, Balanced Scorecard, Fortschreibung von Strategien
- **Berichtswesen:** Datenerhebung, Berichtssysteme, Kommunikationswege, Verantwortungen
- **Controlling-Organisation:** zentral, dezentral, Controlling-Aufwand, Qualifikationsprofile
- Fallbeispiele, Übungsaufgaben

XEN – Praxiseinsatz im Unternehmen (Teil III):

XEN Advanced XXL

Dieser Artikel richtet sich an Berater, Systemadministratoren und Entscheider, die sich mit dem Thema Virtualisierung auseinandersetzen möchten.

In den vergangenen beiden Ausgaben der ORDIX News (3/2006, Seite 13 und 1/2007, Seite 23) berichteten wir über die Virtualisierungslösung XEN. Während diese ersten beiden Artikel eine Einführung in das Open Source Produkt und dessen Grundkonfiguration darstellten, werden wir in diesem Teil drei Funktionen anschauen, die insbesondere bei größeren Installationen interessant sind. Dabei untersuchen wir die Themen Vernetzung und Live-Migration. Abschließend folgt ein kleiner Erfahrungsbericht mit OpenSolaris als Gastsystem.

Anschluss an die Außenwelt

Ein Gastsystem innerhalb eines XEN-Servers stellt in der Regel kein Stand-Alone-System dar. Die Kommunikation über das Netzwerk ist eine wesentliche Voraussetzung für einen sinnvollen Einsatz der virtuellen Maschine. Hierfür stellt XEN virtuelle Netzwerkkarten zur Verfügung, welche auf verschiedene Arten genutzt werden können. Jede Netzwerkkarte bekommt eine eigene MAC-Adresse, die entweder automatisch vergeben oder fest zugewiesen werden kann. Insgesamt können einem Gast in der aktuellen XEN-Version 3.0.3 bis zu drei Netzwerkschnittstellen zugewiesen werden. Dabei stehen drei verschiedene Möglichkeiten der Nutzung zur Verfügung:

1) Bridging
Beim Bridging bzw. Switching werden die Netzwerkkarten der Gäste und das physikalische Interface des Wirtes mit einer virtuellen Bridge verbunden. Jeder Gast bekommt eine IP-Adresse aus dem Subnetzbereich des Firmennetzwerkes und ist direkt von außen ansprechbar (siehe Abbildung 1).

2) Routing
Als Alternative zum Bridging kann auch ein Routing (siehe Abbildung 2) eingerichtet werden. Beim Routing befinden sich Gast und Wirt in einem gemeinsamen, virtuellen Subnetz. Das Wirtsystem fungiert dabei als Router zwischen den Gastsystemen und der Außenwelt.

3) Network Adress Translation (NAT)
NAT wird in Verbindung mit dem Routing eingesetzt. Mit NAT werden die Gäste komplett hinter der IP-Adresse des Wirtes versteckt. Auf dem Wirt findet dabei eine IP-Adressumwandlung statt, wobei in jedem Paket die Adresse des Gastes durch die des Wirtes ersetzt wird. Für die Rechner im Unternehmensnetzwerk stellt es sich so dar, als ob alle Pakete vom Wirtrechner kommen.

Und wo wird das alles konfiguriert?

Wie die meisten XEN-basierten Einstellungen, werden auch die Netzwerkeinstellungen in der Datei **`/etc/xen/xend-config.sxp`** durchgeführt. Hier ist der richtige Ort, um Änderungen an der Netzwerkkonfiguration durchzuführen. Insgesamt ist die Datei sehr gut dokumentiert und mit Beispielen versehen. Der Verwendungszweck der Einträge ist recht schnell ersichtlich. In der mitgelieferten Konfiguration wird standardmäßig eine Bridge mit Namen **`xenbr0`** über die Einträge (**`network-script network-bridge`**) und (**`vif-script vif-bridge`**) konfiguriert. Die Einträge werden immer in runden Klammern in der Form (Schlüsselwort Wert) angegeben.

Abbildung 3 zeigt für jeden Netzwerktyp die Einstellungen. **`network-bridge`** und **`vif-bridge`** sind Shell-Skripte, welche sich im Verzeichnis **`/etc/xen/scripts`** befinden und individuell angepasst werden können.

Das Skript **network-bridge** erstellt die Bridge **xenbr0** und verbindet das physikalische Interface des Wirts damit, **vif-bridge** verbindet die Gäste mit der Bridge.

More about Bridges

Mit der Bridging-Konfiguration ist es möglich, eine benutzerdefinierte Bridge zu erstellen, die für XEN-Gäste genutzt werden kann. Auch ohne XEN kann eine Bridge auf dem Wirt als virtuelle Netzwerkschnittstelle genutzt werden. Für die Konfiguration wird das Paket **bridge-utils** benötigt, welches mit dem Betriebssystem mitgeliefert wird und für XEN zwingend Voraussetzung ist: Mit dem Kommando **brctl** haben Sie die Möglichkeit, eine Bridge für XEN-Gäste einzurichten bzw. generell eine virtuelle Netzwerkschnittstelle zu erstellen. Anhand eines dokumentierten Beispiels (siehe Abbildung 4) möchten wir aufzeigen, wie eine Bridge erstellt und mit einer Netzwerkkarte auf Wirt- und Gastseite verbunden wird.

Wie Abbildung 4 zeigt, lässt sich mit wenigen Befehlen eine Bridge erstellen und mit einer Netzwerkkarte verbinden. Genauso einfach wird auch ein Gast mit der Bridge „verheiratet“. Dazu muss in der Konfigurationsdatei des Gastes der Eintrag der Netzwerkkarte abgeändert werden. Angenommen, **/etc/xen/suse1.cfg** sei unsere Konfigurationsdatei, muss innerhalb der Netzwerkeinstellungen die Bridge mit angeben werden, z. B.

```
vif = ['AA:BB:CC:DD:EE:FF,
bridge=mybridge']
```

Live-Migration - Oder XEN zieht um

Eine für hochverfügbare Umgebungen ausgelegte Funktion ist die Live-Migration von einem XEN-Wirt auf einen anderen. Hierbei wird der Gast während des laufenden Betriebes von einem Wirt auf den anderen übertragen. IP-Adresse, MAC-Adresse und RAM bleiben die ganze Zeit über nutzbar. Technisch gesehen wird erstmals der komplette vom Gast genutzte Arbeitsspeicherinhalt auf den 2. Wirt übertragen. In weiteren Durchgängen wird dann immer wieder der geänderte Teil des Arbeitsspeichers übertragen, bis dieser so klein ist, dass der Gast kurz angehalten werden kann, das letzte Delta übertragen und der Gast auf dem 2. Wirt wieder fortgesetzt werden kann.

In der Praxis entsteht bei der Live-Migration eine kleine Pause im Millisekundenbereich.

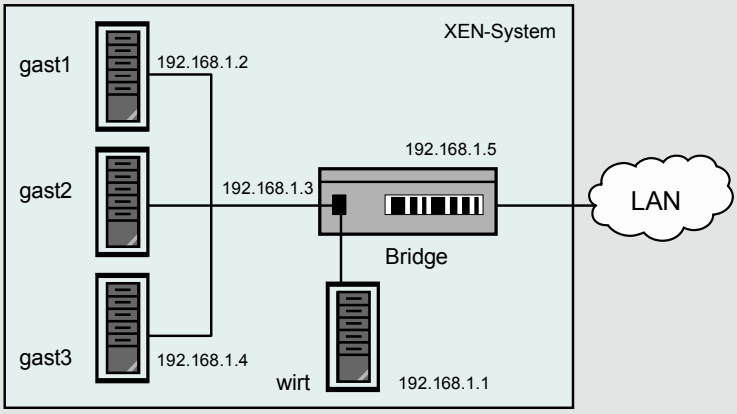


Abb. 1: Bridging bzw. Switching mit XEN.

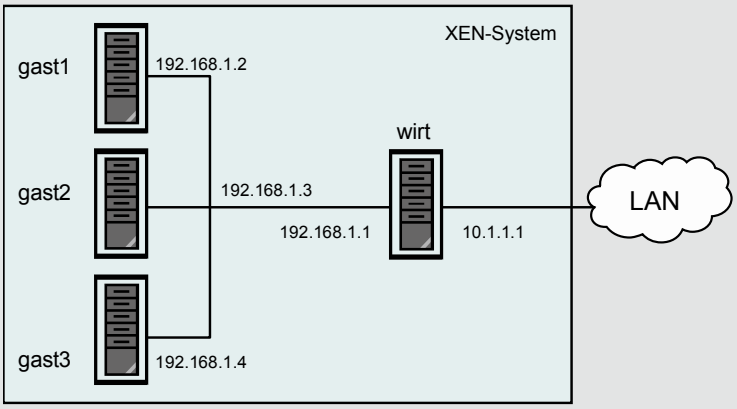


Abb. 2: Routing mit XEN.

Der Anwender, der den Gast gerade nutzt, bekommt nur ein kurzes „Ruckeln“ zu sehen, seine Arbeit wird dadurch aber nicht beeinträchtigt. Es gibt aber einige Voraussetzungen, die für eine erfolgreiche Migration erfüllt sein müssen.

Voraussetzungen für eine erfolgreiche Migration

- Der zu migrierende Gast muss sich auf einem gemeinsam zugreifbaren Medium befinden und von beiden Wirtservern über das selbe Zugriffsprotokoll (NFS, SAN, NAS, iSCSI, etc.) zugreifbar sein.

- Der Zielserver muss genügend freien Arbeitsspeicher haben, um den Arbeitsspeicherinhalt des Gastes aufzunehmen.
- Auf beiden Systemen muss dieselbe Prozessorarchitektur bestehen. Es kann nicht von einem 32-Bit-Prozessor auf einen 64-Bit-Prozessor oder umgekehrt migriert werden. Ausnahme: Ein 32-Bit-Betriebssystem läuft auf 64-Bit-Hardware.
- Der Zielserver muss für das Empfangen von Gästen konfiguriert sein.
- Beide XEN-Server müssen im selben Subnetz sein und dürfen nicht durch einen Router getrennt sein.

Etwas Gemeinschaft gehört dazu

Das Gastsystem muss vor und nach der Migration auf sein Dateisystem zugreifen können. Damit auf beiden Servern identische Daten zur Verfügung stehen, muss sich das Gastbetriebssystem auf einem gemeinsam genutzten Netzwerkspeicher oder Netzwerkdateisystem befinden. Als gemeinsames Medium kommen hier z. B. SAN, NAS, iSCSI, Infiniband oder NFS (siehe Abbildung 5) in Frage. Für das Clustering würde sich als Dateisystem GFS oder OCFS2 anbieten. Möchte man die XEN-Gäste hochverfügbar auslegen, benötigt man noch einen Dienst, der für das Umschalten bei Ausfall eines Knotens sorgt. Hierfür bietet sich die Open Source Lösung Heartbeat an. Als gemeinsamer Datenspeicher kann DRBD verwendet werden.

```
# Einträge für eine Bridge
(network-script network-bridge)
(vif-script vif-bridge)

# Einträge für das Routing
(network-script network-route)
(vif-script vif-route)

# Einträge für das NAT
(network-script network-nat)
(vif-script vif-nat)
```

Abb. 3: Auszug aus der Datei /etc/xen/xend-config.sxp.

```
# Bridge mit Namen mybridge erstellen
brctl addbr mybridge

# Bridge mit phys. Interface eth1 verbinden
brctl addif mybridge eth1

# Der Bridge kann auch eine IP zugewiesen werden
ifconfig mybridge 192.168.1.2
```

Abb. 4: Konfiguration einer eigenen Bridge und Bindung an eine Netzwerkkarte.

Der Ablauf einer Migration

Bei einer Migration muss XEN den vom Gast genutzten Arbeitsspeicherinhalt von einem Wirt auf den anderen übertragen. Der Dateisysteminhalt bleibt dabei unberührt. Das ist auch der Grund für den gemeinsam genutzten Datenspeicher. XEN unterscheidet hierbei Migration und Live-Migration. Nur die Live-Migration ermöglicht das kontinuierliche Weiterarbeiten mit dem Gast.

Eine normale Migration führt im Gegensatz zu einer Live-Migration zwar zu einer Unterbrechung bei der Ausführung, aber zum selben Ergebnis. Diese normale Migration könnte auch manuell durchgeführt werden: **`xm save suse1 /tmp/save`** würde den Gast **`suse1`** pausieren und den Arbeitsspeicherinhalt in die Datei **`/tmp/save`** schreiben. Nun müsste **`/tmp/save`** auf die Zielmaschine kopiert werden und dort mit dem Befehl **`xm restore /tmp/save`** wiederhergestellt werden. Bei der Migration werden IP-Adresse, MAC-Adresse und die aktuelle Gastkonfiguration mit übertragen.

Der neue Gastgeber bereitet sich vor

Um einen XEN-Server für den Empfang von Gästen vorzubereiten, müssen in der XEN-Konfigurationsdatei des Empfangsservers **`/etc/xen/xend-config.sxp`** (siehe Abbildung 6) einige Einträge vorgenommen werden. Mit **`(xend-relocation-port '8002')`** kann der Port, über den kommuniziert wird, festgelegt werden. Mit **`(xend-relocation-address '192.168.1.2')`** wird der Zugriff über eine bestimmte Netzwerkkarte zugelassen. Der Quellserver, von dem die Übertragung zugelassen wird, kann mit **`(xend-relocation-hosts-allow '192.168.1.1')`** angegeben werden.

Wichtig ist, dass XEN die Änderungen mit **`/etc/init.d/xend restart`** mitgeteilt werden. Jetzt kann auf dem Quellserver mit **`xm migrate suse1 192.168.1.2`** bzw. mit **`xm migrate –live suse1 192.168.1.2`** das **`suse1`**-Gastsystem mit oder ohne Unterbrechung auf den Zielserver migriert werden.

Migration im Detail

Während einer Migration sollen drei wichtige Punkte erfüllt sein:

- Die Ausfallzeit soll möglichst gering ausfallen.
- Die Migrationszeit soll so kurz wie nur möglich sein.
- Die Benutzer des Gastsystems sollen von der Migration nicht beeinflusst werden, z. B. sollen Netzwerkverbindungen nicht getrennt werden. Die Migration selbst teilt sich in 6 Phasen (Stages) auf.

Stage 0: Pre-Migration
Der aktive Gast läuft auf dem XEN-Wirt 1. Jetzt wird eine Verbindung zu XEN-Wirt 2 aufgebaut, der für den Empfang des Gastes bereit ist und die nötigen Ressourcen (RAM, CPU, etc.) verfügbar hat.

Stage 1: Reservation
XEN-Wirt 1 sendet eine Anfrage an XEN-Wirt 2, mit der die notwendigen Ressourcen auf XEN-Wirt 2 überprüft werden. Anschließend wird ein Container reserviert, welcher dem Container des Gastes auf XEN-Wirt 1 entspricht.

Stage 2: Iterative Pre-Copy
In der ersten Iteration werden alle Arbeitsspeicherseiten (Pages) übertragen. Anschließend werden nur noch die übertragen, die sich seit der letzen Iteration verändert haben.

Stage 3: Stop-and-Copy
Der Gast auf XEN-Wirt 1 wird gestoppt und der Netzwerkverkehr auf XEN-Wirt 2 umgelenkt. Danach wird der CPU-Zustand sowie der übrig gebliebene, geänderte Arbeitsspeicherinhalt übertragen. Nach Ausführung dieses Schrittes sind auf XEN-Wirt 1 und 2 zwei identische virtuelle Maschinen, XEN-Wirt 1 ist aber immer noch der aktive Server.

Stage 4: Commitment
XEN-Wirt 2 bestätigt, dass er einen identischen Gast besitzt. Somit kann XEN-Wirt 1 seinen Gast beenden und XEN-Wirt 2 wird aktiver Server.

Stage 5: Activation
Der Gast auf XEN-Wirt 2 ist aktiv.

Dieser Ablauf garantiert, dass immer nur ein Server aktiv ist und einen voll funktionsfähigen Gast besitzt. Bei möglichen Fehlern können so keine Daten verloren gehen.

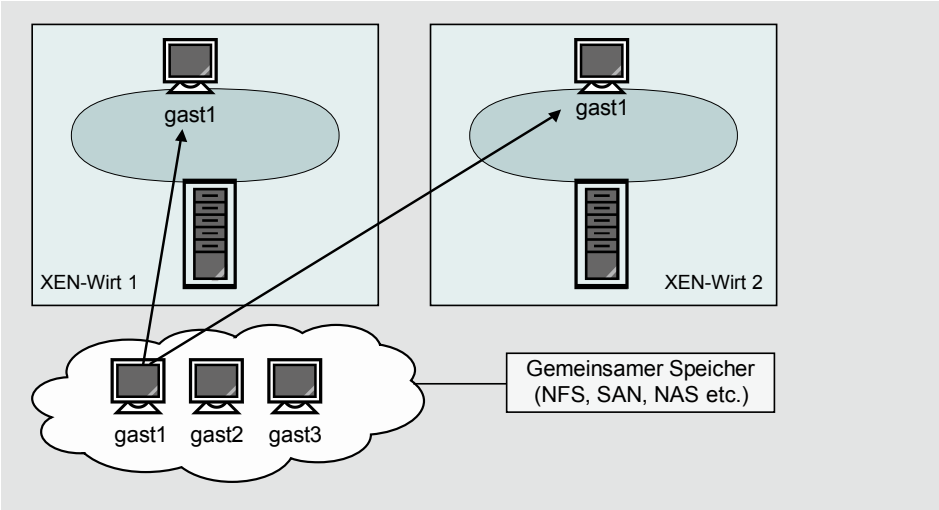


Abb. 5: Gemeinsamer Speicher bei zwei XEN-Servern.

```
(xend-relocation-port '8002')
(xend-relocation-address '192.168.1.2')
(xend-relocation-hosts-allow '192.168.1.1')
```

Abb. 6: Konfiguration des XEN-Wirtes für den Empfang von XEN-Gästen.

OpenSolaris, ein etwas anderer Gast

In der ORDIX News 1/2007 haben wir berichtet, wie ein Linux Betriebssystem als XEN-Gast installiert und konfiguriert werden kann. Im Prinzip gibt es drei Möglichkeiten:

1. Pakete in einer Change-Root Umgebung installieren (chroot-Befehl)
2. Auf Sektorebene (dd-Befehl) ein bestehendes Betriebssystem klonen
3. Auf Dateisystemebene (mit Befehlen wie tar, cpio, rsync etc.) ein bestehendes Betriebssystem klonen

XEN für OpenSolaris, welches aktuell nur für die Intel-Architektur verfügbar ist, kann sowohl als Wirt als auch als Gast konfiguriert werden. Die aktuelle Planung von SUN lässt erwarten, dass XEN Mitte des Jahres 2007 im offiziellen Solaris in beiden Architekturen enthalten sein wird. Wir möchten Ihnen aufzeigen, wie man OpenSolaris unter einem Linux XEN-Wirt als Gast ins Rennen schickt.

```
# Erstellung des Flash-Archivs nach /backup/system.flar des gesamten Betriebssystems
# /backup/system.flar selbst wird aber Excluded
flar create -n XENflarimage -x /backup/system.flar /backup/system.flar

# Erstellung eines Prototype Images aus dem Flash-Archiv
vbdcfg mkproto -f -a /backup/system.flar XENproto

# Erstellung des Festplatten-Images aus dem Protoytpe Image auf Festplattenpartition
vbdcfg mkdomU -d /dev/hda5 -e 00:00:11:AA:BB.CC XENproto solaris1
```

Abb. 7: Erstellung von Flash-Archiv, Prototype Image und Festplatten-Image.

Glossar

Wirt	Physikalisches System auf dem der XEN Kernel läuft und von dem die Gäste gesteuert und kontrolliert werden.
Gast	Virtuelles System, das auf derselben Hardware wie der Wirt läuft.
MAC-Adresse	Physikalische Adresse einer Netzwerkkarte, die für die Kommunikation genutzt wird.
OpenSolaris	Von einer Community fortgesetzte Weiterentwicklung von Solaris mit neuen Funktionalitäten, die von Sun geprüft und in das offizielle Solaris implementiert werden.
Block Device	Ein Block Device ist eine spezielle Datei (Gerätedatei), mit deren Hilfe auf die Hardware, z. B. Festplatte zugegriffen wird. Ein Block Device im Netzwerk lässt sich wie ein lokales Gerät nutzen.

OpenSolaris spricht XEN

Da man OpenSolaris nicht einfach ad hoc unter Linux installieren kann, muss hier ein anderer Weg zur Installation eines Gastes gewählt werden. Eine Möglichkeit wäre, den Gast wie bei Linux mit Unix-Mitteln auf Datei- oder Sektorebene zu klonen. Das hätte aber den großen Nachteil, dass nachträglich sehr viele manuelle Anpassungen durchgeführt werden müssten. Dies betrifft vor allem den Kernel mit den dazugehörigen Modulen.

Für OpenSolaris ist ein anderer Weg zu einem lauffähigen System vorgesehen, bei dem alle notwendigen Maßnahmen automatisch durchgeführt werden. Am besten man installiert OpenSolaris erst einmal auf ganz normalem Weg auf einem Alternativsystem. Die XEN-Pakete für OpenSolaris sind momentan noch nicht auf den Installations-CDs enthalten und müssen separat heruntergeladen [1] und von Hand installiert werden. Die OpenSolaris XEN-Pakete müssen immer installiert werden - vollkommen unabhängig von der Nutzung als Wirt oder als Gast.

Benötigte Pakete zur XEN-Installation

- SUNWPython64.pkg.bz2
- SUNWPython64-devel.pkg.bz2
- SUNWxenh.pkg.bz2
- SUNWxenr.pkg.bz2
- SUNWxenu.pkg.bz2
- SUNWonbld.pkg.bz2

OpenSolaris wird Gast

Die Pakete müssen auf das OpenSolaris System übertragen werden. Dort angekommen, werden die Pakete mit dem Befehl **`bunzip2`** entkomprimiert und mit dem Befehl **`pkgadd -d Paketname`** installiert.

Nun sind alle Vorbereitungen getroffen, um den XEN-Gast für eine Migration vorzubereiten. Die XEN-Pakete für OpenSolaris werden vor allem wegen des Werkzeugs **`vbdcfg`** benötigt, welches einen Prototyp eines OpenSolaris Betriebssystems erstellt. Der Ablauf beim Erstellen eines Gastes teilt sich in drei Teile auf (siehe Abbildung 7):

1. Erstellen eines Flash-Archivs: Mit dem Befehl **`flar`** haben Sie die Möglichkeit, ein so genanntes Flash-Archiv zu erstellen, welches einem Vollbackup des Betriebssystems entspricht.
2. Erstellen eines Prototype Images: Mit dem Befehl **`vbdcfg`** wird dann ein so genanntes „Prototype Image“ aus dem Flash-Archiv erstellt, welches als Template genutzt werden kann.
3. Erstellen des Festplatten-Images (VBD – Virtual Block Device) aus dem Prototyp.

Sie sehen, für die Erstellung eines Solaris-Gastes sind mehrere Schritte nötig. Aber alles in allem ist es leicht zu meistern. Sowohl das Erstellen des Flash-Archivs als auch das Erstellen des Templates benötigen einige Zeit zur Ausführung. An Plattenplatz sollte für das Flash-Archiv etwa die Größe des Gastes bereitgestellt werden, für das Prototype Image etwa das 2 - 3-fache der Größe des Gastes. Gerade bei Solaris bietet es sich an, ein ein-

mal erstelltes Gast-Image zu sichern und bei Bedarf für einen neuen Gast als Vorlage zu nehmen. Das Ausführen des Befehls `vbdcfg mkproto` erstellt ein Kernel Image und eine Konfigurationsdatei, die sofort für XEN genutzt werden kann, im Verzeichnis `/export/xc/xvm/solaris1`.

Fazit

XEN bietet eine ganze Menge Funktionen und Möglichkeiten, die vor allem für den Firmeneinsatz interessant sind. Wenn man die aktuelle XEN-Entwicklung beobachtet, tut sich Einiges. Im nächsten stabilen Linux Kernel 2.6.20 wird eine Schnittstelle für XEN enthalten sein, so dass kein Kernel Patch mehr notwendig sein wird, um XEN zu betreiben. Mitte 2008 wird auch Sun Solaris XEN beherrschen. Und auch einige weitere, interessante Funktionen stehen auf der Warteliste, über die wir in einer zukünftigen ORDIX News berichten.

Christian Fertsch (info@ordix.de).

Link

- [1] http://www.opensolaris.org/os/community/xen

Rückblick IT-Symposium der HP User Society:

ORDIX 007 erforscht Oracle Tracing und Hochverfügbarkeit

Die Fachkonferenz der HP Anwender fand vom 16. bis 20. April 2007 im CongressCenter in Nürnberg statt. Im Rahmen des Programms wurden den Teilnehmern auch aktuelle News zum Thema Oracle Datenbanken geboten. ORDIX präsentierte auf dem Symposium sein Oracle Know-how und lud zum fachlichen Austausch an die ORDIX Infoinsel ein.

007 nimmt Oracle Tracing ins Visier

Dank des großen Erfolges des ORDIX Vortrags "Oracle Tracing - Im Geheimdienst Ihrer Majestät" auf der DOAG Anwenderkonferenz im November 2006 trat Martin Hoermann, Senior Consultant der ORDIX AG auf dem diesjährigen IT-Symposium in Nürnberg auf.

Als „ORDIX 007“ erläuterte er wichtige Tracing-Techniken, Analyse-Strategien für Oracle Datenbanken und die Interpretation von Trace-Dateien.

Schwerpunkt des Vortrags war, die Anwendungsmöglichkeiten bei konkreten Performance-Problemen und Analysen herauszuarbeiten. Wann bietet welches Werkzeug Vor- bzw. Nachteile? Wie sind die Ergebnisse zu interpretieren? V$ Views, Raw Traces, TKPROF und TRCA wurden dabei unter die Lupe genommen. Zusätzlich erläuterte Hoermann die Unterschiede in den verschiedenen Oracle Versionen.

Seminar in Sachen Oracle Hochverfügbarkeit

In einem ganztägigen Spezialtraining wurden die Hochverfügbarkeitslösungen, die Oracle anbietet, vorgestellt und erläutert, wann welche Lösung am besten geeignet ist. Darüber hinaus wurde auch das Zusammenspiel zwischen anderen HV-Lösungen und Oracle diskutiert.

Sie konnten bei dem Vortrag und dem Seminar nicht dabei sein, interessieren sich aber für die von ORDIX vorgestellten Themen oder haben spezielle Oracle Fragen?

Sprechen Sie uns an. Wir beraten Sie gern! Schicken Sie eine E-Mail an info@ordix.de.

„Mission Oracle“-Referent Martin Hoermann.

Seminartermine

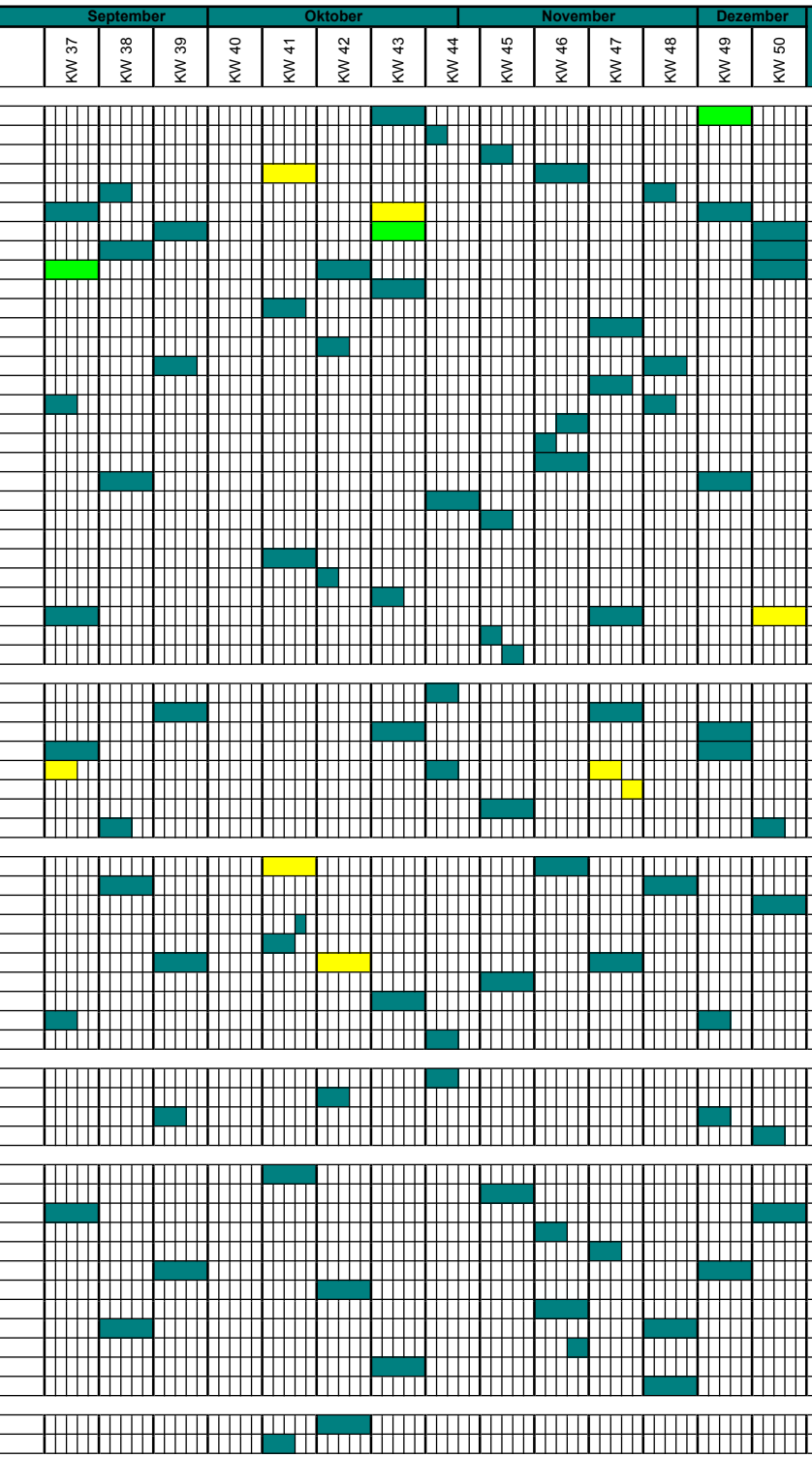
- heraustrennbare Übersicht -

	Preis in EURO*)**)	Mai		Juni				Juli				August					Sep.
		KW 21	KW 22	KW 23	KW 24	KW 25	KW 26	KW 27	KW 28	KW 29	KW 30	KW 31	KW 32	KW 33	KW 34	KW 35	KW 36
Datenbanken																	
Oracle SQL	1790,00																
Oracle SQL für Umsteiger	790,00																
Oracle SQL für Experten	1190,00																
Oracle Datenbankprogrammierung mit PL/SQL	1790,00																
Oracle PL/SQL Aufbau mit LOB Programmierung	1190,00																
Oracle Datenbankadministration Grundlagen	1890,00																
Oracle Datenbankadministration Aufbau	1890,00																
Oracle Backup und Recovery	1890,00																
Oracle Tuning und Monitoring	1890,00																
Oracle Troubleshooting Workshop	1890,00																
Oracle Real Application Cluster (RAC)	1490,00																
Oracle 10g Neuheiten	1890,00																
Oracle Security Workshop	1190,00																
Oracle Data Guard Workshop	1490,00																
Oracle RMAN Workshop	1490,00																
Oracle Grid Control Workshop	1090,00																
Oracle Advanced Queuing Workshop	1090,00																
Oracle Replikation Workshop	790,00																
Informix SQL	1590,00																
Informix Dynamic Server Administration	1790,00																
Informix Tuning und Monitoring	1890,00																
Informix Backup und Recovery mit ON-Bar	1090,00																
IBM DB2 UDB für Unix/Windows SQL Grundlagen	1790,00																
IBM DB2 UDB für Unix/Windows Administration	1890,00																
IBM DB2 UDB für Unix/Windows Version 9.1 Neuheiten	790,00																
MySQL Administration	1090,00																
Microsoft SQL Server Administration	1790,00																
Microsoft SQL Server 2005 Neuheiten	790,00																
Microsoft SQL Server Hochverfügbarkeits-Workshop	790,00																
Programmierung																	
Einführung in die objektorientierte Programmierung	1090,00																
Perl Programmierung Grundlagen	1590,00																
Perl Programmierung Aufbau	1590,00																
Shell, Awk und Sed	1590,00																
Einführung in XML	1090,00																
XML Programmierung unter Java mit DOM und SAX	790,00																
PHP Programmierung Grundlagen	1590,00																
PHP Programmierung Aufbau	1090,00																
Java-J2EE																	
Java Programmierung Grundlagen	1590,00																
Java Programmierung Aufbau	1590,00																
Java GUI Entwicklung mit Swing	1590,00																
J2EE/JEE für Entscheider	450,00																
Einführung in J2EE/JEE	1090,00																
JSP und Servlet Programmierung	1590,00																
EJB Programmierung	1590,00																
Web-Anwendungen mit JavaServer Faces (JSF)	1590,00																
Java Web Services	1090,00																
Entwicklung mit Hibernate	1090,00																
Web- und Applikations-Server																	
Apache Web-Server Installation und Administration	1090,00																
Tomcat Konfiguration und Administration	1090,00																
WebSphere Application Server Installation u. Admin.	1290,00																
Administration und Konfiguration für JBoss	1090,00																
Betriebssysteme																	
Unix/Linux Grundlagen für Einsteiger	1590,00																
Linux Systemadministration	1590,00																
Linux Netzwerkadministration	1590,00																
Server-Virtualisierung mit XEN	1090,00																
Linux Hochverfügbarkeits-Cluster	1190,00																
Unix/Linux Security	1590,00																
Solaris Systemadministration Grundlagen	1890,00																
Solaris Systemadministration Aufbau	1890,00																
Solaris 10 für erfahrene Systemadministratoren	1890,00																
Solaris Containers	790,00																
Solaris für Unix Umsteiger	1890,00																
Multivendor-Systemadministration	1890,00																
Projektmanagement																	
IT-Projektmanagement	1890,00																
Grundlagen des IT-Controlling	1190,00																

Wiesbaden
Saarbrücken
Lippstadt

*) Preise pro Seminar pro Teilnehmer in Euro. Alle Preise gelten zzgl. ges. MwSt.
**) Inhousepreise auf Anfrage.

Für weitere Informationen und Fragen zu individuell zugeschnittenen Seminaren, Ausbildungsreihen oder Inhouse-Schulungen stehen wir jederzeit gerne zur Verfügung. Auf Wunsch senden wir Ihnen auch unser komplettes Seminarprogramm zu.



Preis in EURO*)**)	
	Datenbanken
1790,00	Oracle SQL
790,00	Oracle SQL für Umsteiger
1190,00	Oracle SQL für Experten
1790,00	Oracle Datenbankprogrammierung mit PL/SQL
1190,00	Oracle PL/SQL Aufbau mit LOB Programmierung
1890,00	Oracle Datenbankadministration Grundlagen
1890,00	Oracle Datenbankadministration Aufbau
1890,00	Oracle Backup und Recovery
1890,00	Oracle Tuning und Monitoring
1890,00	Oracle Troubleshooting Workshop
1490,00	Oracle Real Application Cluster (RAC)
1890,00	Oracle 10g Neuheiten
1190,00	Oracle Security Workshop
1490,00	Oracle Data Guard Workshop
1490,00	Oracle RMAN Workshop
1090,00	Oracle Grid Control Workshop
1090,00	Oracle Advanced Queuing Workshop
790,00	Oracle Replikation Workshop
1590,00	Informix SQL
1790,00	Informix Dynamic Server Administration
1890,00	Informix Tuning und Monitoring
1090,00	Informix Backup und Recovery mit ON-Bar
1790,00	IBM DB2 UDB für Unix/Windows SQL Grundlagen
1890,00	IBM DB2 UDB für Unix/Windows Administration
790,00	IBM DB2 UDB für Unix/Windows Version 9.1 Neuheiten
1090,00	MySQL Administration
1790,00	Microsoft SQL Server Administration
790,00	Microsoft SQL Server 2005 Neuheiten
790,00	Microsoft SQL Server Hochverfügbarkeits-Workshop
	Programmierung
1090,00	Einführung in die objektorientierte Programmierung
1590,00	Perl Programmierung Grundlagen
1590,00	Perl Programmierung Aufbau
1590,00	Shell, Awk und Sed
1090,00	Einführung in XML
790,00	XML Programmierung unter Java mit DOM und SAX
1590,00	PHP Programmierung Grundlagen
1090,00	PHP Programmierung Aufbau
	Java-J2EE
1590,00	Java Programmierung Grundlagen
1590,00	Java Programmierung Aufbau
1590,00	Java GUI Entwicklung mit Swing
450,00	J2EE/JEE für Entscheider
1090,00	Einführung in J2EE/JEE
1590,00	JSP und Servlet Programmierung
1590,00	EJB Programmierung
1590,00	Web-Anwendungen mit JavaServer Faces (JSF)
1090,00	Java Web Services
1090,00	Entwicklung mit Hibernate
	Web- und Applikations-Server
1090,00	Apache Web-Server Installation und Administration
1090,00	Tomcat Konfiguration und Administration
1290,00	WebSphere Application Server Installation u. Admin.
1090,00	Administration und Konfiguration für JBoss
	Betriebssysteme
1590,00	Unix/Linux Grundlagen für Einsteiger
1590,00	Linux Systemadministration
1590,00	Linux Netzwerkadministration
1090,00	Server-Virtualisierung mit XEN
1190,00	Linux Hochverfügbarkeits-Cluster
1590,00	Unix/Linux Security
1890,00	Solaris Systemadministration Grundlagen
1890,00	Solaris Systemadministration Aufbau
1890,00	Solaris 10 für erfahrene Systemadministratoren
790,00	Solaris Containers
1890,00	Solaris für Unix Umsteiger
1890,00	Multivendor-Systemadministration
	Projektmanagement
1890,00	IT-Projektmanagement
1190,00	Grundlagen des IT-Controlling

Informationen und Anmeldung:

ORDIX AG
Westernmauer 12 - 16
33098 Paderborn
Tel.: 05251 1063-0

ORDIX AG
Kreuzberger Ring 13
65205 Wiesbaden
Tel.: 0611 77840-00

zentrales Fax: 0180 1 ORDIX 0
bzw. 0180 1 67349 0
E-Mail: training@ordix.de
Online-Anmeldung: http://training.ordix.de

Microsoft SQL Server 2005:

Neue Indexoption „included columns“

Dieser Artikel richtet sich sowohl an Administratoren von Microsoft SQL Servern als auch an Entwickler von Microsoft SQL Server Datenbanken, die bereits über Grundkenntnisse im Bereich Index-Design verfügen.

Indexe können ein effizientes Mittel zur Steigerung der Performance von Datenbank-Abfragen sein. Sie können die Anzahl von logischem und physikalischem I/O unter Umständen dramatisch reduzieren und so zu der gewünschten Performancesteigerung beitragen. Dieser Artikel beleuchtet die neue Indexoption „included columns“ (eingeschlossene Spalten). Anhand eines Beispiels wird der logische I/O der neuen Option mit den beiden bisherigen Varianten verglichen.

Als anschauliches Beispiel wird eine Tabelle zur Verwaltung von Adressen sowie die Abfrage aller Adressen mit dem Wohnort „Berlin“ verwendet. In der Tabelle befinden sich 10.000 Adressen, 5 Prozent davon aus Berlin. Den SQL-Befehl zum Erstellen der Tabelle finden Sie in Abbildung 1, den SQL-Befehl der Abfrage in Abbildung 2.

Um die Abfrage der Adressdaten zu beschleunigen, werden folgende drei Indexe angelegt:

- Ein einfacher Index auf der Spalte „Wohnort“
- Ein abdeckender Index auf alle in der Abfrage verwendeten Spalten („covering index“)
- Ein Index mit der neuen Option der eingeschlossenen Spalten („included columns“)

Für jeden Index wird dessen Größe ermittelt, sowie die Anzahl der Seiten, die für die Beispielabfrage gelesen werden müssen. Dabei wird teilweise mit Näherungswerten gerechnet und es werden nicht alle Verwaltungsinformationen oder -seiten mit einbezogen, da es hier nicht auf die genauen Werte, sondern vielmehr auf die Größenverhältnisse ankommt.

Zunächst wird jedoch die Größe der Tabelle berechnet und der Aufwand für den Fall abgeschätzt, dass die Abfrage ohne Index durchgeführt wird.

Jeder Datensatz belegt in diesem Beispiel 3.500 Bytes. Da in einer 8 KB großen Datenseite des SQL Servers somit 2 Datensätze gespeichert werden können, würden 10.000 Datensätze hier 5.000 Datenseiten belegen.

Wird kein Index verwendet, muss die komplette Tabelle gelesen werden und jeder Datensatz auf das Kriterium der WHERE-Klausel hin überprüft werden, also in unserem Fall 5.000 Datenseiten.

Ein einfacher Index auf der Spalte „Wohnort“

Um die Anzahl der zu durchsuchenden Seiten zu reduzieren, wird in einem ersten Schritt ein Index auf die Spalte „Wohnort“ gelegt (siehe Abbildung 3). Jetzt können über den Index die Datenseiten ermittelt werden, die die benötigten Datensätze enthalten. Diese Datenseiten müssen dann in einem separaten Schritt gelesen werden. Man spricht hierbei von Lookup-Zugriffen.

Bei der Berechnung der Größe des Indexes wird immer mit der untersten Ebene (Ebene 0, diese wird auch Blattebene genannt) begonnen. Dort besteht jeder Index-Datensatz aus dem Index-Feld (100 Bytes) und dem Verweis auf die Datenseite des zugehörigen Datensatzes (6 Bytes). In eine 8 KB große Indexseite passen demnach 76 Index-Datensätze. Bei 10.000 Datensätzen werden also 132 Seiten benötigt.

Zusätzlich wird eine weitere Ebene (Ebene 1) mit zwei Indexseiten benötigt, in der die Verweise auf die Indexseiten der Blattebene gespeichert werden. Eine weitere Ebene (Ebene 2) mit einer Seite bildet die Wurzelseite (root_page) als Ausgangspunkt für alle Abfragen. Siehe hierzu Abbildung 4.

Jetzt müssen circa fünf Prozent der Indexseiten auf Blattebene gelesen werden (7 Seiten), um die Adressen derjenigen Datenseiten zu ermitteln, die Adressen aus Berlin enthalten. Im besten Fall stehen je zwei Berliner Adressen zusammen auf einer Datenseite. Im schlechtesten Fall steht jede Berliner Adresse zusammen mit einer Nicht-Berliner Adresse auf einer Seite. Um die 500 Berliner Adressen zu ermitteln, müssen also zwischen 250 und 500 Datenseiten gelesen werden.

Zusammen mit den oberen Ebenen (Ebenen 1 und 2) des Index (3 Seiten) sind also insgesamt zwischen 260 und 510 Seiten zu lesen.

Ein abdeckender Index auf allen Spalten der Abfrage

Um die Anzahl der zu lesenden Datenseiten weiter zu reduzieren, gab es schon in den vergangenen Versionen des SQL Servers die Möglichkeit, zusammengesetzte Indexe zu erstellen. Werden neben den Spalten der WHERE-Klausel auch alle Spalten der SELECT-Klausel in den Index aufgenommen, so spricht man von einem abdeckenden Index („covering index"), siehe Abbildung 5.

Jetzt ist für die Durchführung der Abfrage ein Zugriff auf die Datenseiten nicht mehr notwendig, da alle benötigten Informationen bereits in den Indexseiten enthalten sind.

Die Anzahl der Indexseiten auf der Blattebene steigt hierbei jedoch auf 527, da jeder Index-Datensatz 406 Bytes groß ist und nur noch 19 Index-Datensätze in eine Indexseite passen. Auf der nächsthöheren Ebene werden nun 28 Indexseiten benötigt, um auf die 527 Seiten der Blattebene zu verweisen. Deshalb wird unterhalb der Wurzelseite eine weitere Ebene (Ebene 2) mit 2 Seiten benötigt, um auf die 28 Seiten der Ebene 1 zu verweisen. Siehe hierzu Abbildung 6.

Die Anzahl der zu lesenden Indexseiten steigt zwar auf circa 30 (5 Prozent von 527, zusätzlich einige Seiten der höheren Ebenen 1 bis 3), allerdings müssen keine Datenseiten mehr gelesen werden.

Diese Lösung sieht auf den ersten Blick sehr gut aus, doch hat sie einen entscheidenden Nachteil: Bei jeder Änderung an einer der Index-Spalten muss nicht nur die entsprechende Datenseite, sondern auch der Index aktualisiert werden. In der vorhergehenden Lösung (siehe Abbildung 4) mit nur einem Index auf „Wohnort" musste der Index nur dann aktualisiert werden, wenn sich der Wohnort geändert hat.

```
CREATE TABLE Adressen
( Name      CHAR(100),   Vorname   CHAR(100)
, Strasse   CHAR(100),   Wohnort   CHAR(100)
, Telefon   CHAR(100),   Hinweise  CHAR(3000)   );
```

Abb. 1: SQL-Befehl zum Erstellen einer Tabelle zur Adressdaten-Verwaltung.

```
SELECT Name, Vorname, Strasse, Wohnort FROM Adressen
 WHERE Wohnort = 'Berlin';
```

Abb. 2: Selektion aller Adressen in Berlin.

```
CREATE INDEX idx_Adressen_Wohnort ON Adressen (Wohnort);
```

Abb. 3: Anlegen eines Index auf die Spalte „Wohnort".

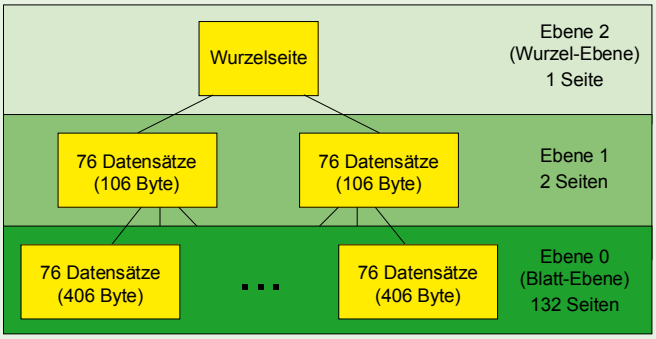


Abb. 4: Baumdarstellung des ersten Indexes.

```
CREATE INDEX idx_Adressen_ganze_Adresse
  ON Adressen (Wohnort, Name, Vorname, Strasse);
```

Abb. 5: Erstellung eines covering index.

Ein Index mit eingeschlossenen Spalten

Mit der Version 2005 hat der Microsoft SQL Server eine neue Indexoption bekommen: Die eingeschlossenen Spalten („included columns"). Die Werte dieser Spalten werden nur in der Blattebene des Indexes gespeichert, nicht in den übergeordneten Ebenen (Ebene 1 und höher). Der Index ist auch nicht

nach diesen Spalten sortiert, womit bei einer Datenänderung innerhalb dieser Spalten der Index nicht umsortiert werden muss.

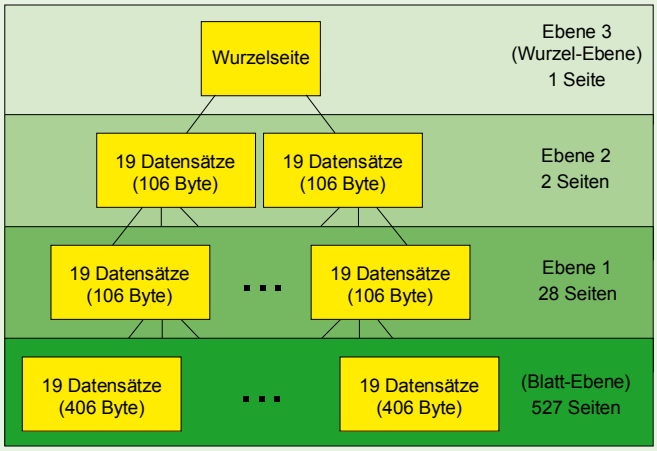


Abb. 6: Baumdarstellung des zweiten Indexes.

```
CREATE INDEX idx_Adressen_ganze_Adresse_Inc
   ON Adressen (Wohnort) INCLUDE (Name, Vorname, Strasse);
```

Abb. 7: Erstellung eines Indexes mit eingeschlossenen Spalten.

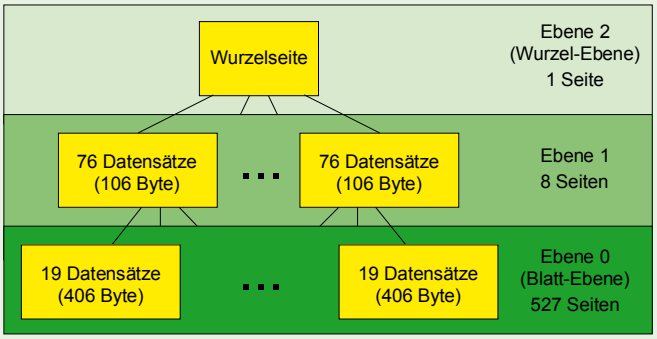


Abb. 8: Baumdarstellung des dritten Indexes.

Glossar

Seite	Speichereinheit des SQL Servers – immer 8 KB groß
Datenseite	Seite, in der Inhalte einer Tabelle gespeichert sind
Indexseite	Seite, die Teil eines Indexes ist
Blattebene	Unterste Ebene eines Indexes – enthält Verweise auf Datenseiten
Wurzelseite	Oberste Ebene eines Indexes und Ausgangspunkt für Abfragen im Index

Um diese Funktion optimal zu nutzen, werden die Spalten der WHERE-Klausel als Index-Spalten und die zusätzlichen Spalten der SELECT-Klausel als eingeschlossene Spalten in den Index aufgenommen (siehe Abbildung 7).

Hierdurch ist die Blattebene dieses Indexes genauso groß wie die Blattebene des zuvor besprochenen abdeckenden Indexes (siehe Abbildung 6). Es werden auch hier 527 Indexseiten für die Blattebene benötigt.

Die nächsthöhere Ebene (Ebene 1) enthält jedoch nur die Spalte „Wohnort“ sowie den Verweis auf die zugehörige Seite der Blattebene. So passen, wie im ersten Beispiel (siehe Abbildung 4), 76 Index-Datensätze in eine Seite. Es werden auf dieser Ebene also 8 Seiten benötigt. Diese 8 Seiten können alle von der Wurzelseite (Ebene 2) aus verkettet werden, so dass dieser Index gegenüber dem vorherigen Index um eine Ebene kleiner ist. Siehe hierzu Abbildung 8.

Im Vergleich zum einfachen Index aus Abbildung 4 können die Werte der Spalten der SELECT-Klausel jetzt ohne Zugriff auf die Tabellen-Seiten direkt aus den Indexseiten ermittelt werden. Nachteile dabei sind aber ein etwas größerer Index und die doppelte Datenhaltung. Da die Daten sowohl in den Tabellen-Seiten als auch in den Indexseiten gespeichert sind, vergrößert sich der Aufwand bei Datenänderungen.

Im Vergleich zum abdeckenden Index aus Abbildung 6 belegt der Index mit eingeschlossenen Spalten aus Abbildung 8 weniger Platz, besteht vielfach aus weniger Ebenen und muss nur reorganisiert werden, wenn die Werte der Index-Spalte („Wohnort“) geändert werden.

Fazit

Das hier vorgestellte Beispiel ist so gewählt, dass der Effekt der eingeschlossenen Spalten deutlich wird. Ob ein Index mit eingeschlossenen Spalten tatsächlich zu einer Steigerung der Performance führt, hängt aber von vielen Faktoren ab: z. B. vom Verhältnis zwischen lesenden und schreibenden Zugriffen sowie von der Größe der Tabelle und der betroffenen Spalten. Gerne unterstützen wir Sie bei der Analyse und Optimierung Ihrer Datenbankanwendung, nicht nur hinsichtlich der Einführung dieser neuen Indexoption.

Andreas Jordan (info@ordix.de).

Reihe EJB 3.0 (Teil III):

Persistenz für alle

Die Abbildung objektrelationaler Persistenz wurde mit EJB 3.0 von Grund auf erneuert. Mehr noch, es wurde ein einheitlicher Persistenzstandard für Java SE und Java EE definiert: die Java Persistence API (JPA). Wir werden in diesem Teil der Reihe ein wenig auf die neuen Möglichkeiten der API im Zusammenhang mit der EJB-Persistenz eingehen.

Dieser Artikel richtet sich an Entwickler, die einen Überblick über die Implementierung von Entity Beans und die neue Java Persistence API im Kontext der EJB 3.0 Spezifikation erhalten möchten.

In den EJB 2.x Versionen war die Abbildung objektrelationaler Persistenz keine einfache Angelegenheit. Obwohl schon immer zentraler Bestandteil der EJB-Spezifikation, haben sich viele Entwickler bei der Implementierung von Entity Beans schwer getan. Unnötige Komplexität, fehlende Möglichkeit der Vererbung oder der Polymorphie, nur 1:1-Zuordnung von Entity Bean zu Datenbanktabelle und eingeschränkter Leistungsumfang der integrierten Abfragemöglichkeit EJB-QL haben einem Entwickler das Leben schwer gemacht.

Java Persistence API

Das Ziel von EJB 3.0 ist es, die Komplexität zu verringern und die Leistungsfähigkeit und die Möglichkeiten zu steigern. Mit der neuen Java Persistence API sind nun Vererbung und Multitable Mapping möglich. Ferner wurde die Komplexität durch die POJO-Persistenzabbildung stark vereinfacht. Viele Dinge in der neuen API werden einem aus Persistenz-Frameworks wie Hibernate oder TopLink bekannt vorkommen. Dies kommt nicht von ungefähr, da die Entwickler dieser Frameworks maßgeblich am neuen EJB 3.0 Standard mitgearbeitet haben.

Mit der Java Persistence API haben wir nun einen einheitlichen Persistenzstandard sowohl für Java SE als auch für Java EE. Das bedeutet, dass Objekte, deren Persistenzabbildung mittels dieser API erfolgt, 1. innerhalb eines Java EE Application Server und 2. außerhalb, in einer reinen Java Laufzeitumgebung, ablaufen können, ohne dass eine Änderung nötig ist.

```
@Entity
@Table(name="KUNDE")
public class Kunde implements Serializable {
    private Integer id;
    private String name;

    @Id(generate= GeneratorType.AUTO)
    public Integer getId()  { return id; }

    @Column(name="NAME")
    public String getName() { return name; }
}
```

Abb. 1: Beispiel für eine Entity Bean.

```
@Stateless public class KundenManagerBean {

   //Injection der Referenz auf den Entity Manager
   @PersistenceContext private EntityManager entityManager;

   public Integer anlegenKunde(String name, ...) {
      Kunde kunde = new Kunde(name,...);
      entityManager.persist(kunde);
      return kunde.getId();
   }

   public Kunde findeKunde(Integer id) {
      return entityManager.find(Kunde.class, id);
   }
}
```

Abb. 2: Arbeitsweise des Entity Managers.

Eine Entity Bean ist nun ein einfaches POJO, das erst mittels einer bestimmten Annotation (`@Entity`) zur Entity Bean wird. Das Beispiel in Abbildung 1 zeigt solch eine Entity Bean,

die auf eine Tabelle KUNDE abgebildet ist. Die Tabelle KUNDE hat die Spalten NAME und ID (Primary Key).

Entity Manager

Der Entity Manager ist ein zentrales Element der Java Persistence API. Er stellt alle persistenzbezogenen Funktionalitäten zur Verfügung. Diese Funktionalitäten waren in den vorherigen Versionen direkt in der Entity Bean enthalten. Da für diese Funktionalitäten nun der Entity Manager zuständig ist, können die Entitäten nun als leichtgewichtige POJOs implementiert werden, was die Komplexität erheblich reduziert. Über ihn können Objekte geladen, gelöscht, gespeichert, aktualisiert und gesucht werden. Abbildung 2 zeigt die Arbeitsweise des Entity Managers an einem Beispiel: einen Kunden anlegen, speichern und suchen.

Ein Client hat zudem noch die Möglichkeit, über den Entity Manager Abfragen über Entity Beans durchzuführen. Der Manager bietet folgende Möglichkeiten:

- EJB-QL-Abfragen (siehe Abbildung 3)
- Benannte EJB-QL-Abfragen (siehe Abbildung 4)
- Native SQL-Abfragen

Multitable Mapping

Wie eingangs erwähnt, war es bisher nur möglich, eine Entity Bean auf eine Tabelle abzubilden. Dies reichte in vielen Entwicklungsprojekten nicht aus, da sich Geschäftsobjekte häufig über mehrere Tabellen einer relationalen Datenbank verteilen. Das führte dazu, dass Tabellen denormalisiert oder aber der SQL-Code vom Entwickler selbst implementiert werden musste (Stichwort BMP - Bean-managed Persistence).

In EJB 3.0 ist es nun möglich, die Daten für ein Objekt aus mehreren verschiedenen Tabellen zu erhalten. Eine Entity Bean ist nun ein logisches Geschäftsobjekt, das seine Daten aus mehreren Tabellen aus der Datenbank erhält. In der Entity Bean wird einfach über Annotations (`@SecondaryTable`) angegeben, aus welcher Tabelle das Objekt seine Daten bezieht. Das Beispiel in Abbildung 5 zeigt ein solches Multitable Mapping. Die in dem Beispiel verwendeten Tabellen sind:

- KUNDE mit den Spalten ID (PK), NAME
- ADRESSE mit den Spalten ID (PK), KUNDE_ID (FK), STRASSE

Die Fremdschlüsselbeziehung zwischen den Tabellen wird, falls nicht explizit anders angegeben, über den Primärschlüssel (PK) der Primärtabelle und über einen Fremdschlüssel (FK) der Sekundärtabelle abgebildet. Der Name dieses Fremdschlüssels setzt sich aus dem Namen der Primärtabelle und dem Namen des Primärschlüssels der Primärtabelle zusammen. Sollte dies nicht der Fall sein, so muss die Fremdschlüsselbeziehung der Tabellen über eine spezielle Annotation in der Entity Bean erfolgen.

Die Zuordnung der Attribute zu der jeweiligen Tabellenspalte erfolgt bei den `get`-Methoden über die Annotation `@Column`. Enthält diese Annotation nicht das Element `secondaryTable`, so erfolgt das Mapping über die Primärtabelle, wie es im Beispiel bei der `get`-Methode `getName()` der Fall ist. Bei dem Attribut `strasse` erfolgt das Mapping über die Sekundärtabelle ADRESSE, da in der zugehörigen Annotation `secondaryTable="ADRESSE"` steht.

```
@Stateless public class KundenManagerBean {
   ...
   public List<Kunde> findeKunden(String name) {
      Query q = entityManager.createQuery
         ("from Kunde k where k.name = :kname");
      q.setParameter("kname",name);
      List<Kunde> kunden = (List<Kunde>) q.getResultList();
      return kunden;
   }
}
```

Abb. 3: Beispiel für EJB-QL (Query Language).

```
@Entity
@Table(name="KUNDE")
@NamedQuery(name="alleKunden",queryString="from Kunde")
public class Kunde implements Serializable {
   ...
}

@Stateless
public class KundenManagerBean {
   ...
   public List<Kunde> alleKunden() {
      Query q = entityManager.createNamedQuery("alleKunden");
      List<Kunde> kunden = (List<Kunde>) q.getResultList();
      return kunden;
   }
   ...
}
```

Abb. 4: Beispiel für benannte EJB-QL: Named Query.

Beziehungsgeflechte

Zwischen den verschiedenen Objekten kann es ganz unterschiedliche Beziehungen geben. Die Art der Beziehung wird, wie erwartet, in den jeweiligen Entity Beans durch Annotations bestimmt. Es können folgende Beziehungen zwischen Objekten abgebildet werden:

- Eins-zu-Eins-Beziehungen (1:1) über die Annotation **`@OneToOne`**
- Eins-zu-Viele-Beziehungen (1:n) über die Annotation **`@OneToMany`**
- Viele-zu-Eins-Beziehungen (n:1) über die Annotation **`@ManyToOne`**
- Viele-zu-Viele-Beziehungen (n:m) über die Annotation **`@ManyToMany`**

Die folgenden Beispiele sollen die Möglichkeiten der Beziehungsgeflechte ein wenig veranschaulichen:

Eins-zu-Eins-Beziehungen

In dem Beispiel in Abbildung 6 soll jeder Kunde genau eine Adresse haben. Die Methode **`getAdresse`** von Kunde soll also diese Adresse zurückgeben. Der Kunde muss zwingend eine Adresse haben. Dies wird in der Annotation **`@OneToOne`** durch das Element **`optional=false`** erreicht. Die Annotation **`@JoinColumn`** wäre hier gar nicht nötig gewesen, da durch die Namensgebung der Fremdschlüssel als solcher erkannt wird. Die dazugehörigen Tabellen lauten:

- KUNDE mit den Spalten ID (PK), NAME, ADRESSE_ID (FK)
- ADRESSE mit den Spalten ID (PK), ORT, STRASSE

Eins-zu-Viele-Beziehungen

Hier gibt es zu jedem Kunden mehrere Adressen. Dies wird durch die Annotation **`@OneToMany`** bei der Methode **`getAdressen()`** erreicht (siehe Abbildung 7). Weitere Angaben sind in diesem Fall nicht nötig, da die Beziehung der Tabellen in diesem Fall durch die Namensgebung gegeben ist. Die dazugehörigen Tabellen lauten:

- KUNDE mit den Spalten ID (PK), NAME
- ADRESSE mit den Spalten ID (PK), ORT, STRASSE
- Join-Tabelle KUNDE_ADRESSE mit den Spalten ID (PK), KUNDE_ID, ADRESSE_ID

```
@Entity
@Table(name="KUNDE")
@SecondaryTable (name="ADRESSE")
public class Kunde implements Serializable {
   private Integer id;
   private String name;
   private String strasse;
   ...
   @Column(name="NAME")
   public String getName()    { return name; }

   @Column(name="STRASSE", secondaryTable="ADRESSE")
   public String getStrasse() { return strasse; }
}
```

Abb. 5: Beispiel für Multitable Mapping.

```
@Entity
@Table(name="ADRESSE")
public class Adresse implements Serializable {
   int id;
   String ort;
   String strasse;
   ...
}

@Entity
@Table(name="KUNDE")
public class Kunde implements Serializable {
   int id;
   String name;
   Adresse adresse;

   @OneToOne(optional=false)
   @JoinColumn(name="ADRESSE_ID")
   public Adresse getAdresse() { return adresse; }
   ...
}
```

Abb. 6: Beispiel für eine Eins-zu-Eins-Beziehung.

```
@Entity
@Table(name="ADRESSE")
public class Adresse implements Serializable {
   int id;
   String strasse;  ...
}

@Entity
@Table(name="KUNDE")
public class Kunde implements Serializable {
   int id;
   String name;
   Collection<Adresse> adressen;

   @OneToMany
   public Collection<Adresse> getAdressen() { return adressen;}
   ...
}
```

Abb. 7: Beispiel für eine Eins-zu-Viele-Beziehung.

```
@Entity
@Table(name="ADRESSE")
public class Adresse implements Serializable { ...
   Kunde kunde;

   @ManyToOne
   public Kunde getKunde() { return kunde; }
}

@Entity
@Table(name="KUNDE")
public class Kunde implements Serializable { ...
   Collection<Adresse> adressen;

   @OneToMany(mappedBy="kunde")
   public Collection<Adresse> getAdressen() { return adressen; }
}
```

Abb. 8: Beispiel für eine Viele-zu-Eins-Beziehung.

```
@Entity
@Table(name="ADRESSE")
public class Adresse implements Serializable { ...
   Collection<Kunde> kunden;

   @ManyToMany
   public Collection<Kunde> getKunden() { return kunden; }
}

@Entity
@Table(name="KUNDE")
public class Kunde implements Serializable { ...
   Collection<Adresse> adressen;

   @ManyToMany(mappedBy="kunde")
   public Collection<Adresse> getAdressen() { return adressen; }
}
```

Abb. 9: Beispiel für eine Viele-zu-Viele-Beziehung.

```
@Entity
@Table(name="GESCHAEFTSPARTNER")
@Inheritance(strategy=SINGLE_TABLE)
@DiscriminatorColumn(name="GP_TYPE",
@discriminatorType=STRING,length=20)
public class Geschaeftspartner implements Serializable {
   int id;
   String name1;
   String name2; ...
}

@Entity
@DiscriminatorValue("KUNDE")
public class Kunde extends Geschaeftspartner {
   String attributkunde; ...
}

@Entity
@DiscriminatorValue("LIEFERANT")
public class Lieferant extends Geschaeftspartner {
   String attributlieferant; ...
}
```

Abb. 10: Beispiel für Vererbung mittels Single Table per Class Hierarchy.

Viele-zu-Eins-Beziehungen

In dem Beispiel in Abbildung 8 möchten wir zu einer gegebenen Adresse den dazugehörigen Kunden ermitteln. Dies erreichen wir durch die Annotation **`@ManyToOne`** bei der Methode **`getKunde()`** in der Klasse **`Adresse`**. Da wir nicht mit einer Join-Tabelle arbeiten, müssen wir durch das Element **`mappedBy="kunde"`** in der Annotation **`@OneToMany`** in der Klasse **`Kunde`** die Beziehung festlegen. Die in diesem Beispiel verwendeten Tabellen lauten:

- KUNDE mit den Spalten ID (PK), NAME
- ADRESSE mit den Spalten ID (PK), ORT, STRASSE, KUNDE_ID (FK)

Viele-zu-Viele-Beziehungen

In unserem letzten Beispiel zu Beziehungsgeflechten kann jeder Kunde mehrere Adressen haben, aber es können auch zu einer Adresse mehrere Kunden existieren (siehe Abbildung 9). Erreicht wird dies durch eine Join-Tabelle und die Annotation **`@ManyToMany`**, die wir sowohl bei der Methode **`getKunden()`** in der Klasse **`Adresse`**, als auch bei **`getAdressen()`** in der Klasse **`Kunde`** verwenden. Die entsprechenden Tabellen sind die gleichen, wie in unserem Beispiel für Eins-zu-Viele-Beziehungen (siehe Abbildung 7).

Vererbung

Mit EJB 3.0 und der darin definierten Java Persistence API ist nun auch Vererbung und Polymorphie für persistente Objekte wie z. B. Entity Beans möglich. EJB 3.0 stellt für die Umsetzung der Vererbung bzw. die Abbildung der Vererbungshierarchie eines Objektmodells auf eine relationale Datenbank verschiedene Strategien zu Verfügung:

Single Table per Class Hierarchy (SINGLE_TABLE)

Bei dieser Strategie werden alle Klassen einer Vererbungshierarchie auf eine einzige Tabelle in der Datenbank abgebildet. Alle Informationen dieser Klassen werden in denormalisierter Form in der Datenbank gespeichert. Zur Unterscheidung der Klassen wird in der Tabelle ein Kennzeichen, der so genannte „Discriminator“, eingeführt (siehe Abbildung 10). Gibt man bei der Programmierung nicht an, welche Strategie man verwenden möchte, so ist dies die Standardstrategie.

Single Table per Concrete Entity Class `(TABLE_PER_CLASS)`

Hier wird jede Klasse einer Vererbungshierarchie auf eine separate Tabelle abgebildet. Jede Tabelle enthält alle Daten einer Klasse - auch die geerbten Attribute. Festgelegt wird diese Strategie durch die Annotation `@Inheritance(strategy = TABLE_PER_CLASS)`.

Joined Subclass Strategy `(JOINED)`

Diese Strategie ähnelt der vorhergehenden, nur enthalten die Tabellen der Subklassen nur die spezifischen Attribute der jeweiligen Subklasse, also ohne die geerbten Attribute. Diese werden in der Tabelle der Basisklasse gespeichert. Jede Tabelle der Subklasse enthält den Fremdschlüssel der Tabelle der Basisklasse. Diese Strategie wird durch die Verwendung der Annotation `@Inheritance (strategy=JOINED)` bestimmt.

Als letztes Beispiel in unserem Artikel soll die Vererbung mittels der **Single Table per Class Hierarchy** Strategie veranschaulicht werden (siehe Abbildung 10). In diesem Beispiel gibt es eine Basisklasse `Geschaeftspartner`, von der die Subklassen `Kunde` und `Lieferant` erben. Um diese Klassen zu unterscheiden, gibt es in der Datenbanktabelle als Discriminator ein Kennzeichen (GP_TYPE). Die dazugehörige Tabelle lautet:

- GESCHAEFTSPARTNER mit den Spalten GP_TYPE, ID (PK), NAME1, NAME2, ATTRIBUTKUNDE, ATTRIBUTLIEFERANT

Detachment

In den meisten Fällen ist es nun nicht mehr notwendig, Data Transfer Objects (DTO) zu verwenden. Die unter der Überwachung eines Entity Managers stehenden POJOs bzw. Objekte können als so genannte **Detached Objects** einfach serialisiert zum Client transferiert werden. Das Objekt bzw. das mit `@Entity` annotierte POJO muss einfach nur das Interface `java.io.Serializable` implementieren.

Fazit

Dieser Artikel gibt einen ersten Überblick über die neue Java Persistence API. Auch wenn nicht alle Eigenschaften und Details behandelt werden konnten, wird deutlich, dass sich durch die neue JPA nun mit erheblich niedrigerem Aufwand eine Anbindung an relationale Datenbanken realisieren lässt. Mit JPA wurde ein einheitlicher Standard geschaffen, sowohl für Java SE als auch für Java EE. Da nun Entity Beans konsequent als POJOs umgesetzt werden, ist ein großer Teil der Komplexität der alten Versionen beseitigt worden. Dies ist ein großer Schritt nach vorne und hilft dabei, die Verwendung von JEE im Allgemeinen und Entity Beans im Besonderen in Projekten verstärkt einzusetzen. Dies hat sich in unseren Projekten bereits in der Praxis bestätigt.

Oliver Kaluza (info@ordix.de).

Glossar

Java SE/JSE	Java Standard Edition
Java EE/JEE	Java Enterprise Edition. Erweiterung von Java für Server-Anwendungen.
Persistenz	Die Fähigkeit, Objekte in nicht-flüchtigen Speichermedien wie Dateisystemen oder Datenbanken zu speichern.
POJO	Abkürzung für Plain Old Java Object. Dabei handelt es sich um ein normales Objekt in der Programmiersprache Java.
Annotations	Anmerkungen im Java Source Code, die zur Compile- oder Laufzeit des Programms ausgewertet werden können. Annotations sind seit Java 5 Bestandteil der Java Sprache. Als geistigen Vater dieser Annotations kann man das Open Source Projekt XDoclet ansehen. XDoclet erlaubte es, auch schon in früheren Java Versionen mit Annotations zu programmieren.
EJB	Enterprise Java Beans sind standardisierte Komponenten, aus denen JEE-konforme Anwendungen erstellt werden, die auf einem JEE-Application Server laufen. Für unterschiedliche Zwecke definiert der JEE-Standard verschiedene Arten von EJBs: Session Beans, Entity Beans und Message-driven Beans.
Hibernate	Dies ist ein Open Source Persistence Framework für Java. Das Framework ermöglicht es, den Zustand eines Objekts in einer relationalen Datenbank zu speichern und aus entsprechenden Datensätzen wiederum Objekte zu erzeugen. Dies bezeichnet man auch als Object Relational Mapping (OR-Mapping, kurz ORM). Es befreit den Entwickler von der Programmierung von SQL-Abfragen und hält die Applikation unabhängig vom SQL-Dialekt der verwendeten Datenbank.
EJB-QL	EJB-QL (Query Language) ist eine deklarative Abfragesprache, ähnlich der bei relationalen Datenbanken verwendeten Structured Query Language (SQL), aber maßgeschneidert für das abstrakte Persistenzschema von Entity Beans.
Data Transfer Object	Das Data Transfer Object fasst in einer verteilten Umgebung [z. B. J(2)EE] zu übertragende Daten in einem neuen Objekt zusammen, um die Anzahl der entfernten Methodenaufrufe zu reduzieren und somit das Netzwerk zu entlasten.
Diskriminator	Er wird bei der Vererbung verwendet. Die Unterscheidung in Ober- und Unterklasse erfolgt mit Hilfe eines Unterscheidungsmerkmals, dem so genannten Diskriminator. Dieser definiert den für die Strukturierung maßgeblichen Aspekt.

Lang laufende Transaktionen und Optimistic Locking – Reihe Hibernate (Teil IV):

Lange Gespräche mit Hibernate

Dieser Artikel richtet sich an Java-Entwickler, die bereits mit dem Umgang von Hibernate vertraut sind.

Die meisten Anwendungsfälle funktionieren immer nach dem Schema: laden, anzeigen, ändern, speichern. Die Arbeitsschritte sind klar abgegrenzt und der Entwickler hat leichtes Spiel. Was ist aber, wenn es in einer Web-Anwendung gilt, sich über mehrere Arbeitsschritte hinweg Änderungen zu merken, ohne diese zu speichern, z. B. in einem mehrschrittigen Registrierungsprozess? Plötzlich könnte alles schrecklich kompliziert sein – mit Hibernate jedoch nicht.

Die Session, ganz alltäglich

Die Session ist in einer Anwendung auf Basis von Hibernate der Schlüssel zur Interaktion mit der Datenbank. Es gibt sehr viele Möglichkeiten zur Konfiguration der Session und noch viele Möglichkeiten mehr bei der Anwendung. Im Verlauf des Artikels werden wir uns das Verhalten der Session bezogen auf den Zeitpunkt des Speicherns von veränderten Daten (Flush) sowie ihre Lebensdauer näher anschauen.

Das Standardverhalten, das allen Hibernate-Entwicklern vertraut sein sollte, ist, dass sich die Session immer parallel zu einer Datenbanktransaktion „bewegt“. Direkt nach dem Beziehen der Session wird eine Transaktion über diese gestartet. Danach lädt und verändert die Applikationslogik im Rahmen dieser Transaktion ein persistentes Objekt. Am Ende wird die Transaktion committed und die Session schreibt dabei die Änderungen an von ihr geladenen Objekten in die Datenbank (`flush`). Anschließend beendet sich die Ses-

sion selbst (`close`), wobei auch alle von der Session geladenen Objekte in den entkoppelten (`detached`) Zustand wechseln.

Für viele Anwendungsfälle ist dieses als *Session per Request* bezeichnete Verhalten ideal. Der Entwickler muss sich nicht um das Speichern seiner Veränderungen kümmern. Er muss die Objekte nur durch die Session laden und verändern, alles andere erledigt Hibernate von selbst. Aber was ist, wenn einer der letzten beiden Schritte – das Schreiben und Entkoppeln – nicht sofort erfolgen soll?

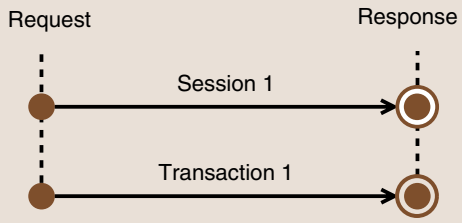


Abb. 1: Das Session per Request Modell.

Ein Beispiel: Wizard

Für die folgenden Betrachtungen soll das Beispiel eines mehrstufigen Benutzerdialogs (Wizard) dienen. Mehrere Masken folgen aufeinander und der Benutzer kann beliebig zwischen den Masken vor und zurück navigieren. Erst am Ende dieser Abfolge, also nach dem Ausfüllen der letzten Maske, sollen die veränderten Daten gespeichert werden. In der Anwendung hätte man zunächst einen initialen Aufruf, der die Daten für die erste Maske lädt und zur Anzeige bringt. Danach folgt jeweils zwischen der Darstellung von zwei Masken das Entgegennehmen der geänderten Daten aus der vorherigen Maske und dann das Laden der Daten für die folgende Maske. Am Ende des Gesamtprozesses würden die Daten der letzten Maske entgegengenommen werden und dann der speichernde Aufruf erfolgen.

Ein Problem dabei ist, dass jede Datenbankinteraktion durch die Session innerhalb einer Transaktion stattfinden muss. Hibernate verpflichtet den Entwickler dazu. Wie oben erwähnt, ist das Standardverhalten der Hibernate Session aber so, dass Änderungen am Ende einer Transaktion durch einen impliziten `flush()` gespeichert werden. Session per Request würde hier folglich bewirken, dass Änderungen sofort beim Wechsel von einer Maske zur nächsten gespeichert werden.

Hibernate hilft mit Long Conversation

Ein erster, einfacher Ansatz, um dieses Problem zu umgehen, würde vorsehen, die Transaktion im initialen Aufruf zu starten und erst am Ende des Gesamtprozesses zu speichern. Dies hat aber den gravierenden Nachteil, dass die Transaktion auch die User-Think-Time, also die Zeit, die ein User benötigt, um die nächste Maske auszufüllen, umspannt. Während dies für einen oder wenige Anwender sicherlich noch kein (großes) Problem ist, wird dies mit wachsender Anzahl von Benutzern zum K.O.-Kriterium. Über kurz oder lang werden die vielen offenen Transaktionen die Datenbank ausbremsen – die Anwendung skaliert nicht.

Die Lösung des Problems ist einfach und besteht in einer so genannten Long Conversation. Darunter versteht Hibernate die Entkoppelung von Session und Transaktion, indem kein impliziter Flush mehr erfolgt. Stattdessen muss der Entwickler am Ende eines Gesamtprozesses dafür sorgen, dass seine Anwendung den Flush explizit durch Aufruf von `Session.flush()` ausführt. Dies kann konfiguriert werden, indem der FlushMode der Session zu Beginn des Gesamtprozesses mit `Session.setFlushMode (FlushMode .NEVER)` umgestellt wird. Der Ablauf der Anwendung wäre nun wie oben geschildert. In allen Masken können Änderungen vorgenommen werden, ohne dass der Stand innerhalb der Datenbank verändert würde. Erst während der abschließenden Transaktion würde der Flush der Änderungen ausgelöst.

Wie funktioniert die Umsetzung?

Bei der Umsetzung einer Long Conversation bietet Hibernate die Möglichkeit, die Lebensdauer der Session zu steuern. Zur Auswahl stehen zwei Varianten (Patterns):

- Detached Objects (Session per Request with DetachedObjects)
- Extended Session (Session per Conversation)

Der Unterschied zwischen den Varianten besteht darin, ob nach der Transaktion eines jeden Maskenaufrufs auch die Session geschlossen und die persistenten Objekte somit entkoppelt werden oder nicht. Bei Detached Objects ist dies der Fall (siehe Abbildung 2).

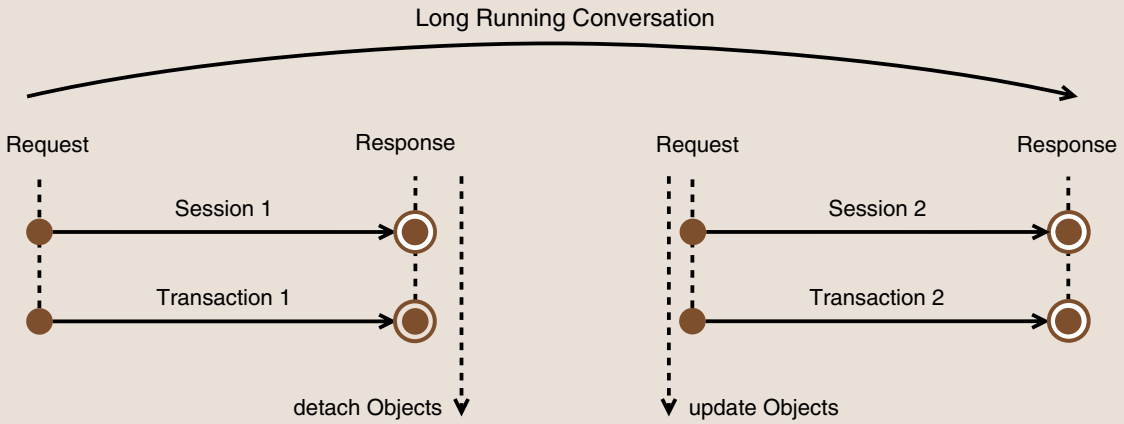


Abb. 2: Das Session per Request with Detached Objects Modell.

Beim Start der Anwendungslogik der nächsten Maske wird eine neue Session generiert und mit Hilfe von `Session.update(Object)` würde ein verändertes Objekt an die neue Session gebunden.

Bei der Alternative Extended Session (siehe Abbildung 3) wird auf das Schließen der Session verzichtet. Beim Start der Anwendungslogik der nächsten Maske wird die aus dem vorherigen Request vorhandene Session weiterverwendet. Dazu muss lediglich eine neue Transaktion über `Session.beginTransaction()` gestartet werden. Da die persistenten Objekte nicht von ihrer Session entkoppelt werden, ist es bei dieser Vorgehensweise auch nicht notwendig, ein `Session.update()` durchzuführen.

Parallele Prozesse: Wir sind nicht allein

Die Sperren in der Datenbank wurden durch die geschilderten Maßnahmen auf ein Minimum reduziert. Somit wurde auf technischer Basis dafür gesorgt, dass die Anwendung gut skaliert und auch viele Anwender gleichzeitig tragbar sind. Implizit hat sich nun aber ein neues Problem in Form von parallel laufenden Prozessen ergeben. Technisch ist es nun möglich, dass zwei (oder mehr) Benutzer den gleichen Anwendungsfall und die gleichen Daten parallel nutzen und dabei unterschiedliche Änderungen vornehmen. Es kommt zum Konflikt. Ohne eine Steuerung dieser Nebenläufigkeit werden immer die Änderungen, die zuletzt gespeichert werden, wirksam (last commit wins).

Ein in solchen Situationen häufig genutzter Mechanismus ist die „Optimistic Concurrency Control“. Als optimistisch wird dieser Ansatz deshalb bezeichnet, weil er nicht auf Sperren setzt, die es ja gerade zu vermeiden gilt. Statt dessen geht man davon aus, dass alle anderen Zugriffe keine Konflikte erzeugen. Vor dem bzw. beim Schreiben in die Datenbank findet eine Kontrolle statt, ob Änderungen, die in Konflikt zu den eigenen stehen, vorgenommen wurden. Wenn nein, werden die Daten übernommen. Wenn ja, ist es an der Anwendung, eine Lösung zu finden. Ein typisches Vorgehen würde den Anwender über die parallel durchgeführten Änderungen informieren und z. B. einen manuellen Abgleich ermöglichen.

Automatische Versionierung

Die Kontrolle, ob parallele Änderungen stattgefunden haben, kann beliebig implementiert werden. Als effektiv hat sich die Versionierung von Datensätzen mit Versionsnummern oder Zeitstempeln erwiesen. Dafür muss das Datenmodell um eine Spalte für die Version erweitert werden. Ist eine Anpassung des Datenmodells nicht möglich, kann auch ein Vergleich aller (veränderten) Werte zwischen dem persistenten Objekt und der Datenbank erfolgen. Beide Ansätze werden von Hibernate unterstützt, so dass der notwendige Abgleich nicht mehr vom Entwickler der Anwendung implementiert werden muss.

Die Versionierung muss lediglich im Mapping der persistenten Klasse konfiguriert werden.

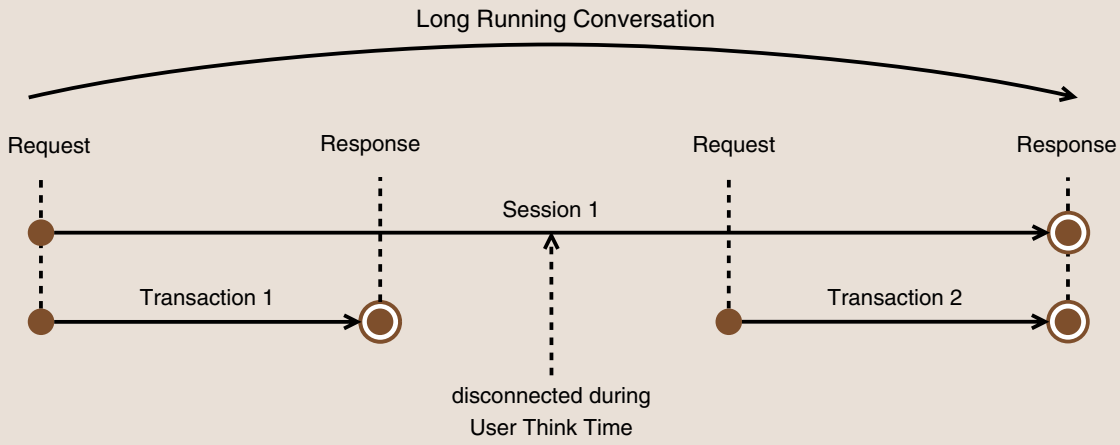


Abb. 3: Das Session per Conversation Modell.

Dazu wird die entsprechende Tabellenspalte mit Hilfe des Elements `<version>` angegeben. Beim Laden eines persistenten Objekts durch die Session wird die Version ebenfalls geladen. Beim `flush()` werden vor dem Schreibzugriff in der Datenbank die Versionen von Session und Datenbank für das zu speichernde Objekt verglichen. Sind die Versionen identisch, wird die Version von Hibernate automatisch aktualisiert und die Änderungen anschließend persistiert. In einem parallel laufenden Prozess würde die Prüfung der Versionen anschließend fehlschlagen und die Anwendung muss entsprechend reagieren können.

Ist eine Anpassung des Datenbankschemas nicht möglich, z. B. weil eine ältere Anwendung das Schema nutzt, kann Hibernate auch die Werte einer Instanz einzeln mit dem Stand der Datenbank vergleichen. Dazu muss im `<class>`-Element des Mappings das Attribut `optimistic-lock` auf `all` gesetzt werden. Wenn parallele Änderungen an einem Objekt nicht schädlich sind, so lange sich die Änderungen nicht überlagern, kann auch `dirty` als Wert gesetzt werden. In diesem Fall sind beim `flush()` nicht alle, sondern nur die veränderten Felder für den Abgleich mit der Datenbank relevant.

Fazit

Hibernate unterstützt den Entwickler bei der Entwicklung von skalierbaren Anwendungen enorm. Mit der Entkopplung von Session und Transaktion wird die Implementierung atomarer Operationen auf das Setzen eines `FlushMode.NEVER` und explizites Aufrufen von `Session.flush()` reduziert. Dass dabei möglicherweise Konflikte durch parallele Änderungen entstehen, kann Hibernate nicht verhindern. Aber mit den Automatismen für die Versionierung bzw. Überprüfung auf gleichzeitige Veränderungen bietet es zwei hilfreiche Werkzeuge, um der Lage Herr zu werden. Mehr Details hierzu erfahren Sie auch in unserem Seminar „Entwicklung mit Hibernate“. Nähere Informationen und Anmeldungsmöglichkeiten dazu finden Sie im Internet unter http://training.ordix.de.

Michael Heß (info@ordix.de).

Glossar

User Think Time	Die Zeitspanne, in der die Anwendung auf Benutzereingaben wartet.
Long Conversation	Bezeichnet einen Anwendungsfall, der mehrere Zyklen von User Think Time enthält.
Wizard	Eine Abfolge von Dialogen, die in mehreren Schritten Daten vom Benutzer abfragt. Wird häufig zur Vereinfachung von Installationen oder Konfigurationen eingesetzt.
Flush	Schreiben der veränderten Daten aus den POJOs in die Datenbank.
FlushMode	Steuert den Zeitpunkt des Flush. Standard ist das Ende der Transaktion.
Detached Objects	Persistente Objekte, deren Session geschlossen wurde, wechseln in diesen Zustand. Sie können zu einem späteren Zeitpunkt mit einer neuen Session verknüpft werden und somit wieder in den Zustand persistent wechseln.

MySQL 5.1 New Features (Teil I)

Zeitgesteuerte MySQL-Jobs: Sag mir Quando, sag mir wann!

Der Artikel richtet sich an Datenbankadministratoren, die nach einer plattformunabhängigen Jobsteuerung für ihre MySQL Datenbanken suchen.

Es geht Schlag auf Schlag: Wiederum stellt die Firma MySQL AB eine neue Datenbankversion, MySQL 5.1, mit einigen neuen, interessanten Funktionen vor. Eines der Highlights dürfte sicherlich der neue Event Manager sein, über den nun endlich zeitgesteuerte Aufgaben direkt in der Datenbank realisiert werden können. Im folgenden Artikel möchten wir Ihnen diese Funktion vorstellen und den praktischen Einsatz an einigen Code-Beispielen illustrieren.

Einsatzmöglichkeiten

Sowohl Datenbankanwender als auch -administratoren sind oftmals mit wiederkehrenden Aufgaben beschäftigt. Sei es das regelmäßige Erstellen von Materialized Views mit den Umsatzzahlen des letzten Monats oder Jahres, das Prüfen von Tabellen oder Indexen, das nächtliche Laden von Daten oder das periodische Aufräumen von Datenbanken und/oder Tabellen.

Bislang wurden solche Aufgaben durch das Schreiben von SQL-Skripten gelöst, die dann über Betriebssystemmittel (Crontab oder geplante Tasks) von außerhalb der Datenbank gesteuert wurden. Hierzu war es notwendig, das Passwort des ausführenden Users im Klartext beim Aufruf zu hinterlegen. Alternativ hatte man natürlich die Möglichkeit, vollständig auf eine Authentifizierung zu verzichten. Beide Varianten waren in der Praxis eher unbefriedigend.

Aktivierung

Die Steuerung von Jobs kann von nun an datenbankintern über einen eigenen Prozess er-

folgen. Dieser Job-Prozess ist beim Start der MySQL Datenbank in der neuen Version zwar aktiviert, führt zunächst jedoch keine Aufträge aus. Er befindet sich in einer Art Standby-Modus. Um die Jobsteuerung zu aktivieren, muss der globale Konfigurationsparameter **event_scheduler** umgesetzt werden:

0 Scheduler ist deaktiviert
1 Scheduler ist aktiviert und führt Jobs aus
2 Scheduler ist aktiviert, führt aber keine Jobs aus (Default-Einstellung)

Die Aktivierung des Schedulers kann wahlweise zur Laufzeit über **set**-Anweisungen (siehe Abbildung 1) oder aber dauerhaft über die Konfigurationsdatei (**my.ini** bzw. **my.cnf**) vorgenommen werden.

```
mysql>show variables like '%event%';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | 2     |
+-----------------+-------+
1 row in set (0.00 sec)

mysql>set global event_scheduler = 1;
Query OK, 0 rows affected (0.00 sec)
```

Abb. 1: Ermitteln und Aktivieren der Jobsteuerung.

Jobs

Um Aufträge erstellen zu dürfen, benötigt der Datenbank-User das „EVENT“-Privileg. Das Anlegen von Jobs erfolgt dann über eine wohlbekannte DDL-ähnliche Syntax (siehe Abbildung 2). Besondere Beachtung verdient hier die **schedule**-Klausel. Über diese Syntax können beliebig komplexe Zeitpläne erzeugt werden.

```
CREATE EVENT [IF NOT EXISTS] event_name
   ON SCHEDULE schedule
   [ON COMPLETION [NOT] PRESERVE]
   [ENABLE | DISABLE]
   [COMMENT 'comment']
   DO sql_statement;

schedule:
AT timestamp [+ INTERVAL interval]
   | EVERY interval [STARTS timestamp] [ENDS timestamp]

interval:
quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
   WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
   DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

Abb. 2: Die Syntax zur Erstellung eines Jobs.

Prinzipiell können zwei Arten von Ausführungsplänen unterschieden werden:

- Zyklische Pläne, z. B. alle 5 Minuten
- Terminierte Pläne, z. B. jeden 1. Montag im Monat um 05:30 Uhr

Zusätzlich können Zeitpläne mit Bedingungen, ähnlich den IF-Anweisungen aus Programmiersprachen, kombiniert werden, z. B. „Führe den Auftrag alle fünf Minuten aus, wenn der User XYZ eingeloggt ist“. Einige exemplarische Zeitpläne und Jobs können der aktuellen MySQL-Dokumentation [1] entnommen werden.

```
mysql>  use events;
mysql>  create table test.current_user
       (
       id bigint,
       user varchar(16),
       host varchar(16),
       command varchar(16)
       );
mysql> create event test.log_user
on schedule every 30 second
do
insert into events.current_user
select id, user, host, command
from information_schema.processlist;
```

Abb. 3: Beispiel für einen einfachen Job, welcher die User-Aktivität überwacht.

Fallstricke

Die vollständige Qualifikation von Datenbankobjekten (**<DB_NAME>.<OBJECT_NAME>**) ist prinzipiell immer eine gute Sache. Gerade beim Umgang mit Events sollte man aber besonders achtsam sein. Events (Jobs) sind wie alle anderen Objekte datenbankspezifisch. In unserem Beispiel (siehe Abbildung 3) liegt das Event in der Datenbank **events**. Der voll qualifizierte Name dieses Objektes lautet also: **events.log_user**. Das Event sucht damit alle anderen, nicht voll qualifizierten Objekte, wie z. B. Tabellen, in genau diesem Kontext.

Darüber hinaus werden Aufträge immer mit den Rechten des Besitzers, also des Users, der den Job angelegt hat, ausgeführt. Hat dieser nicht ausreichende Berechtigungen für den Zugriff auf die benötigten Tabellen, schlägt der Job fehl.

Kontrolle

Dem Administrator stehen, wie bei MySQL üblich, mehrere Wege offen, um Informationen über seine Events einzuholen. Zum einen stehen die zwei Systemtabellen **`mysql.event`** und **`information_schema.events`** zur Verfügung, in denen u. a. auch die letzten Ausführungszeiten der Events protokolliert werden. Zum anderen präsentiert das neue Kommando **`SHOW [FULL] EVENTS`** Auszüge aus den Systemtabellen. Darüber hinaus kann, ähnlich wie bei Tabellen, das Kommando **`SHOW CREATE EVENT event_name`** genutzt werden, um Informationen über den Aufbau des Events zu bekommen. Das Kommando **`ALTER EVENT event_name ENABLE/DISABLE;`** aktiviert bzw. deaktiviert nachträglich bestehende Aufträge.

Überwachung

Automatismen können die Arbeit eines Datenbankadministrators erleichtern. Dennoch sollte man sich nicht 100%ig auf diese Mechanismen verlassen. Getreu dem Motto: „Vertrauen ist gut, Kontrolle ist besser".

Eine Möglichkeit, sich über die Ausführungen eines Events zu informieren, ist das allgemeine Error Log des Servers (siehe Abbildung 4).

Besonderes Augenmerk sollte hierbei auf den Eintrag Manager **`thread started with id 1`** gelegt werden. Dieser Eintrag gibt darüber Auskunft, ob der Scheduler überhaupt beim Serverstart aktiviert wurde.

Die nachfolgenden Zeilen zeigen uns den Verlauf unseres Auftrages **`log_user`** (siehe Abbildung 3), der alle 30 Sekunden erneut ausgeführt wird. Wünschenswert ist hier, dass die Jobs mit einem **`RetCode=0`** (Return Code) beendet werden. Dies zeigt an, dass der Job ohne Fehler beendet wurde.

Eine weitere Möglichkeit, sich den Status des Schedulers anzeigen zu lassen, bietet die Prozessliste:

```
Version: '5.1.11-beta'  socket: ''  port: 3306  MySQL Community Server (GPL)
060720 10:30:31 [Note] SCHEDULER: Manager thread booting
060720 10:30:31 [Note] SCHEDULER: Loaded 0 events
060720 10:30:31 [Note] SCHEDULER: Manager thread started with id 1
060720 10:33:12 [Note] SCHEDULER: [events.log_user of root@localhost] executing in thread 5
060720 10:33:12 [Note] SCHEDULER: [events.log_user of root@localhost] executed. RetCode=0
060720 10:33:17 [Note] SCHEDULER: [events.log_user of root@localhost] executing in thread 6
060720 10:33:17 [Note] SCHEDULER: [events.log_user of root@localhost] executed. RetCode=0
```

Abb. 4: Auszug aus der allgemeinen Error-Log des Servers.

```
drop table if exists event_error;

create table event_errors
   (
   nr int primary key auto_increment,
   job varchar(20),
   jobdate timestamp
   );

drop event if exists log_user;

delimiter //
create event log_user on schedule every 5 second
do
  begin
  declare exit handler for sqlexception
    insert into event_errors (job) values ('log_user');
    insert into events.monitor select * from nowhere;
  end
//
```

Abb. 5: Event zum Protokollieren auftretender Fehler in der Tabelle `event_errors`.

```
select id, user, host, command, state,
info from processlist
where user = 'event_scheduler';
```

Job Monitoring

Weitaus eleganter und sicherer ist jedoch die Nutzung von **`BEGIN/END`**-Blöcken im Event selber, da diese Blöcke mit einem ERROR-Handler verknüpft werden können (siehe Abbildung 5). Ein ERROR-Handler kann beispielsweise genutzt werden, um aufgetretene Probleme direkt in eine Tabelle zu protokollieren. Während der Blick in die Prozessliste nur darüber Aufschluss gibt, ob das Scheduling generell gestartet wurde, können die ERROR-Handler jobspezifisch gestaltet werden.

In unserem Beispiel werden in einem Fehlerprotokoll der Ausführungszeitpunkt und der

Name des entsprechenden Jobs in der Tabelle `events.event_errors` geloggt. Natürlich können in so einem ERROR-Handler auch andere Maßnahmen getroffen werden, wie z. B. das Erzeugen von nicht vorhandenen Tabellen usw.

Fazit

Die MySQL-Events sind eine weitere sinnvolle Funktion, welche jeden Administrator und Entwickler erfreuen dürfte. Im Zusammenhang mit den erst kürzlich eingeführten Triggern und Prozeduren (siehe ORDIX News 1/2006, Seite 30) ergeben sich sowohl für den Anwender als auch für den Administrator vielfältige Möglichkeiten, um lästige, wiederkehrende Aufgaben zu automatisieren, ohne auf datenbankexterne Mechanismen zurückgreifen zu müssen. In der nächsten Ausgabe stellen wir Ihnen dann die neu geschaffene Möglichkeit der Partitionierung vor.

Matthias Jung (info@ordix.de).

Link

► [1] http://dev.mysql.com/doc/refman/5.1/en/events.html

Glossar

View	Eine View ist eine Sicht auf den Inhalt einer oder mehrerer Tabellen. Hinter einer View verbirgt sich datenbankintern eine SELECT-Anweisung.
Materialized View	Materialized Views sind spezielle Sichten auf Tabellen, deren Inhalt temporär physikalisch gespeichert wird, um den Aufwand zur Berechnung zu minimieren.
Crontab	Die Crontab bzw. der Cron-Dienst dient der zeitlichen Steuerung von wiederkehrenden Aufgaben. In der Crontab werden diese Aufgaben in einer Tabellen-ähnlichen Struktur terminiert.
DDL	Data Definition Language. Mit DDL-Kommandos werden Datenstrukturen gepflegt (z. B. Tabellen anlegen oder löschen).

Larry Ratlos:

Filtern von Groß- und Kleinbuchstaben mit grep

Auch in Larrys Büro soll ein „Frühjahrsputz“ im Bereich Benutzerdaten durchgeführt werden. Auf seinem Linux System sollen alle Benutzerkennungen von nun an in Kleinbuchstaben vereinheitlicht und das Chaos zwischen Groß- und Kleinschreibung damit beseitigt werden. Zur Bereinigung soll ein kleines Shell-Skript dienen.

Die Benutzernamen sind in der 1. Spalte der Datei `/etc/passwd` gespeichert:

```
root:x:0:0:root:/root:/bin/bash
chef:x:100:100:Chef:/home/chef:/bin/bash
Tim:x:101:100:Tim:/home/Tim:/bin/bash
larry:x:102:100:Larry:/home/larry:/bin/bash
```

Um nun die großgeschriebenen Benutzernamen ausfindig zu machen, setzt Larry folgendes `grep`-Kommando ab:

```
grep '^[A-Z]' /etc/passwd
```

„Komisch“, denkt Larry beim Anblick der Ausgabe. „Warum werden denn alle Benutzerkennungen ausgegeben und nicht, wie ich vermutet hatte, nur die 3. Zeile? Und vor allem: Wie bekomme ich denn nun das gewünschte Ergebnis?“

Können Sie Larry helfen? Dann senden Sie Ihren Lösungsvorschlag **bis zum 22. Juni 2007 an kniffel@ordix.de**. Selbstverständlich werden die schnellsten Kniffler wieder belohnt.

Lösung der Aufgabe 1/2007

Im Januar wollte Larry seine Datenbankblöcke optimal gefüllt wissen - jedoch ohne eine Tabellen-Reorganisation. Die passende Lösung lieferte ihm Herr Peter Hombach von der Koenig & Bauer AG:

Man kann sich den Reorg sparen, wenn man schon beim Insert eine Sortierung mit Hilfe der Funktion NLSSORT vornimmt:

```
insert into iot
   select * from basis
   order by
   nlssort (from_uid,'nls_sort=binary'),
   nlssort (to_uid,'nls_sort=binary');
```

Reihe Oracle Objekttypen von A - Z (Teil III):

„E“ wie Evaluation Context

Der Artikel richtet sich an Entwickler und Administratoren, die eine Replikation von Daten zwischen verschiedenen Datenbanken aufsetzen wollen.

In unserer Reihe der Oracle Objekttypen sind wir nun beim Buchstaben „E“ angekommen und stellen Ihnen den Oracle Objekttyp „Evaluation Context“ vor. Der Evaluation Context ist ein Datenbankobjekt, in dem Variablenwerte gespeichert und/oder Tabelleninhalte abgelegt werden, so dass sie von Rule Conditions referenziert werden können. Diese Datenbankobjekte können bei der Funktionalität Oracle Streams (siehe ORDIX News 2/2006, Seite 4) benutzt werden.

Wofür benötigt man Evaluation Context?

Bei Oracle Streams werden aus den Redolog-Daten Befehle und Daten extrahiert. Dieses Prinzip findet beispielsweise bei der Logical Standby Datenbank Anwendung. Mit Streams ist es darüber hinaus möglich, mit Hilfe von Regeln nur bestimmte Informationen aus dem Redolog-Strom zu gewinnen. Die Regeln bestehen aus Bedingungen, die auf Daten angewendet werden. Als Analogie: Stellen Sie sich die Rule Condition als `where`-Bedingung und den Evaluation Context als `from`-Klausel vor.

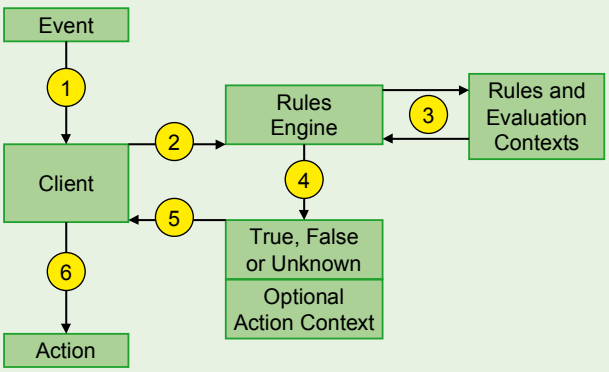


Abb. 1: Rule Evaluation Context.

Die Interpretation und Auswertung der Daten übernimmt der Evaluation Context. Die Rule Condition wird in dem Schema ausgewertet, in dem sich der Evaluation Context befindet.

Zusammenhang zwischen Evaluation Context und Rule Condition

Ein Alias `dep` zeigt auf eine Tabelle `department` im Schema `hr`. Die Variablen `loc_id1` und `loc_id2` haben beide den Typ NUMBER. Die Rule Condition lautet `dep.location_id` in (`:loc_id1, :loc_id2`) und ist in einem Evaluation Context namens `hr_evaluation_context` enthalten. In diesem Fall ergibt die Rule Condition TRUE für die Datensätze der Tabelle mit zutreffender `Location-Id` zu `loc_id1` oder `loc_id2`. Die Regel kann ohne die Informationen aus dem Evaluation Context nicht ausgewertet werden.

Implizite oder explizite Variablenwerte

Die Variablenwerte können explizit bei der Evaluierung des Evaluation Context oder implizit durch Events (Ereignisse) gesetzt werden. Eine Evaluierung wird über Parameter durch den Aufruf von `dbms_rule.evaluate` durchgeführt. Implizite Variablen werden bei der Erstellung des Evaluation Context mit Aufruf von `CREATE_EVALUATION_CONTEXT` im Package DBMS_RULE_ADM gesetzt.

Rule Set Evaluation Prozess

Der Prozess der Evaluierung wird wie folgt durchlaufen:

1. Ein vom Client definiertes Event wird ausgelöst.
2. Der Client initiiert die Evaluierung eines Rule Sets durch Senden von Informationen mit der Prozedur **DBMS_RULE.EVALUATE** an die Rules Engine. Sowohl Rule Set als auch Evaluation Context spezifiziert der Client.
3. Die Rules Engine wertet die zugehörigen Rule Sets durch den entsprechenden Evaluation Context aus.
4. Die Ergebniswerte der zugehörigen Rules sind TRUE, FALSE oder NULL. Optional gibt es auch einen Action Context.
5. Der Client erhält die Ergebnisse von der Rules Engine.
6. Mit diesen Ergebnissen führt der Client Aktionen aus.

Abbildung 1 illustriert diese Vorgehensweise.

Anlegen eines Evaluation Context

Mit der Syntax aus Abbildung 2 wird ein Evaluation Context angelegt. Dort wird eine explizite Variable **priority** vom Typ NUMBER dem Evaluation Context **supportctx** zugewiesen. Ein zusätzlicher Kommentar wird in der Datenbank angelegt, um die Bedeutung dieses Evaluation Context zu erklären.

Anlegen eines Rule Sets

Ein Evaluation Context bekommt notwendigerweise Rule Sets und/oder Rules zugewiesen. Im Beispiel in Abbildung 3 wird im Evaluation Context **supportctx** ein Rule Set **regelset** angelegt. Ein zusätzlicher Kommentar dient der Lesbarkeit und wird im Data Dictionary abgelegt.

Anlegen von Rules

Mit dem Beispiel in Abbildung 4 werden drei Regeln angelegt, denen jeweils ein Action Set zugeordnet ist. Der Action Context besteht aus Name-Wert-Kombinationen. Der Name „Center“ wird bei Rule **r1** mit dem Wert „Paderborn“ belegt, bei der Rule **r2** mit „Berlin“.

```
DECLARE
vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
   vt := SYS.RE$VARIABLE_TYPE_LIST ( SYS.RE$VARIABLE_TYPE
      ('priority', 'NUMBER', NULL, NULL));
   DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT
      ( evaluation_context_name => 'supportctx',
      variable_types => vt,
      evaluation_context_comment => 'Support Problem Definition');
END;
/
```

Abb. 2: Beispiel zum Anlegen eines Evaluation Context.

```
BEGIN
DBMS_RULE_ADM.CREATE_RULE_SET ( rule_set_name => 'regelset',
   evaluation_context => 'supportctx',
   rule_set_comment => 'Support Regeln');
END;
/
```

Abb. 3: Beispiel zum Anlegen eines Rule Sets, Zuordnung zum Evaluation Context.

```
DECLARE
   ac SYS.RE$NV_LIST;
BEGIN
   ac := SYS.RE$NV_LIST(NULL);
   ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('Paderborn'));
   DBMS_RULE_ADM.CREATE_RULE( rule_name => 'r1',
      condition => ':priority > 2',
      action_context => ac,
      rule_comment => 'Probleme niedriger Priorität');
   ac := SYS.RE$NV_LIST(NULL);
   ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('Berlin'));
   DBMS_RULE_ADM.CREATE_RULE( rule_name => 'r2',
      condition => ':priority <= 2',
      action_context => ac,
      rule_comment => 'Probleme hoher Priorität');
   ac := SYS.RE$NV_LIST(NULL);
   ac.ADD_PAIR('ALERT',
      SYS.AnyData.CONVERTVARCHAR2('Erwin Mustermann'));
   DBMS_RULE_ADM.CREATE_RULE( rule_name => 'r3',
      condition => ':priority = 1',
      action_context => ac,
      rule_comment => 'Dringendes Problem');
END;
/
```

Abb. 4: Beispiel zum Anlegen von Rules.

```
BEGIN
DBMS_RULE_ADM.ADD_RULE( rule_name => 'r1', rule_set_name => 'regelset');
DBMS_RULE_ADM.ADD_RULE( rule_name => 'r2', rule_set_name => 'regelset');
DBMS_RULE_ADM.ADD_RULE( rule_name => 'r3', rule_set_name => 'regelset');
END;
```

Abb. 5: Beispiel für die Zuordnung von Rules zu einem Rule Set.

```
CREATE OR REPLACE PROCEDURE
   problem_dispatch (priority NUMBER)
IS
   vv           SYS.RE$VARIABLE_VALUE;
   vvl          SYS.RE$VARIABLE_VALUE_LIST;
   truehits     SYS.RE$RULE_HIT_LIST;
   maybehits    SYS.RE$RULE_HIT_LIST;
   ac           SYS.RE$NV_LIST;
   namearray    SYS.RE$NAME_ARRAY;
   name         VARCHAR2(30);
   cval         VARCHAR2(100);
   rnum         INTEGER;
   i            INTEGER;
   status       PLS_INTEGER;
BEGIN
   vv := SYS.RE$VARIABLE_VALUE('priority', SYS.AnyData.CONVERT
NUMBER(priority)); vvl := SYS.RE$VARIABLE_VALUE_LIST(vv);
   truehits := SYS.RE$RULE_HIT_LIST();
   maybehits := SYS.RE$RULE_HIT_LIST();
   DBMS_RULE.EVALUATE( rule_set_name => 'regelset',
      evaluation_context => 'supportctx', variable_values => vvl,
      true_rules => truehits, maybe_rules => maybehits);
   FOR rnum IN 1..truehits.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE('Nutze Regel '|| truehits(rnum).rule_name);
      ac := truehits(rnum).rule_action_context;
      namearray := ac.GET_ALL_NAMES;
      FOR i IN 1..namearray.count loop
         name := namearray(i);
         status := ac.GET_VALUE(name).GETVARCHAR2(cval);
         IF (name = 'CENTER') then
            DBMS_OUTPUT.PUT_LINE('Zuweisung erhält: ' || cval);
            ELSIF (name = 'ALERT')
         THEN
            DBMS_OUTPUT.PUT_LINE('Alarm erhält: '|| cval);
         END IF;
      END LOOP;
   END LOOP;
END;
/
```

Abb. 6: Beispiel zur Benutzung von Evaluation Context.

```
SQL> EXECUTE problem_dispatch(1);
Nutze Regel "SUPPORT"."R3
Alarm erhält: Erwin Mustermann
Nutze Regel "SUPPORT"."R2"
Zuweisung erhält: Berlin

PL/SQL-Prozedur erfolgreich abgeschlossen.

SQL> EXECUTE problem_dispatch(2);
Nutze Regel "SUPPORT"."R2"
Zuweisung erhält: Berlin

PL/SQL-Prozedur erfolgreich abgeschlossen.

SQL> EXECUTE problem_dispatch(3);
Nutze Regel "SUPPORT"."R1"
Zuweisung erhält: Paderborn

PL/SQL-Prozedur erfolgreich abgeschlossen.
```

Abb. 7: Aufruf und Ausgaben zu den Beispielen.

Die Rule **r3** bekommt den Namen „Alert“ mit dem Wert „Erwin Mustermann“.

Zuweisung von Rules zu einem Rule Set

Mit der Syntax aus Abbildung 5 werden die Rules zu einem Rule Set zugeordnet und gleichzeitig zu einem Rule Set zusammengefasst. Die Zuordnung erfolgt über **DBMS_RULE_ADM.ADD_RULE** sowie die Namen von Rule und Rule Set.

Nutzung des Evaluation Context in einer Prozedur

In der Abbildung 6 wird die Benutzung vom Evaluation Context innerhalb einer Prozedur verdeutlicht. Diese kann dann z. B. wie in Abbildung 7 aufgerufen werden und erzeugt die dort dargestellten Ausgaben.

Zu Beginn der Prozedur erhält die explizite Variable **priority** den Übergabeparameter. Die Evaluierung wird über den Aufruf von **DBMS_RULE.EVALUATE** angestoßen. Zurückgegeben werden die Rules, die mit TRUE bewertet wurden und die so genannten Maybe Rules.

Auf das Thema Maybe Rules kann hier aber nicht weiter eingegangen werden. Informationen hierzu finden Sie in der Dokumentation „Oracle Streams Concepts und Administration“ [1]. Anschließend werden die zurückgegebenen Regeln und die zugehörigen Aktionen ausgewertet und ausgegeben.

Data Dictionary

Aus den folgenden Views des Data Dictionary können Informationen zu den erstellten Objekten abgefragt werden:

- USER_EVALUATION_CONTEXTS
- USER_RULES
- USER_RULE_SETS
- USER_RULE_SET_RULES

Fazit

Das vorgestellte Beispiel ist nur ein einfaches Beispiel zur Benutzung von expliziten Variablen. Darüber hinausgehende Möglichkeiten werden aus Platzgründen nicht vorgestellt. Auch hier sei nochmal auf die Dokumentation

Glossar

Rule	Regel, nach der ausgewertet wird, ob Daten verwendet oder verworfen werden.
Rule Set	Gruppe von Regeln
Rule Condition	Komponente einer Regel, die aus Ausdrücken und/oder Bedingungen besteht. Das Ergebnis ist TRUE, FALSE oder NULL.
Rules Engine	Ein Built-In als Funktionalität, um die Rule Sets zu evaluieren.
Action Context	Optionale Information, die einer Rule zur späteren Verwendung in einer Meldung zugewiesen wird.
Oracle Streams	Funktionalitäten, mit denen Daten zwischen Datenbanken verteilt werden können.
Event	Ereignis, das von einem Client oder der Rules Engine angestoßen wird.

verwiesen. Für die Administration der Rules, Rule Sets und Evaluation Contexts gibt es eine ausgeprägte Rechtestruktur, auf die hiermit nur hingewiesen wird.

Beate Künneke (info@ordix.de).

Link

- [1] Dokumentation „Oracle Streams Concepts und Administration“: http://www.oracle.com/technology/documentation/database10gR2.html

ORDIX auf der JAX 2007:

ORDIX Vorträge auf der

Mit über 140 Vorträgen lockte die JAX in diesem Jahr zahlreiche Anhänger der Java Community vom 23. bis 27. April 2007 in die Rhein-Main-Hallen in Wiesbaden. Zum zweiten Mal präsentierte auch ORDIX sein Java Know-how mit einem Stand und zwei Vorträgen dem interessierten Fachpublikum der Java Messe.

Java, XML und Webservices

Einblick in aktuelle Entwicklungen der vielschichtigen Themenwelt um Java, XML und Webservices erhofften sich die Besucher der Java Messe auch in diesem Jahr. Sowohl in den Vorträgen als auch in der begleitenden Ausstellung nutzten sie die Gelegenheit, sich direkt mit Fachleuten zahlreicher Firmen über die aktuellen Trends auszutauschen.

ORDIX Vorträge zu Hochverfügbarkeit, Lastverteilung und Java Persistence API

Die Forderung nach Hochverfügbarkeit und Lastverteilung von Webcontainern wird immer lauter. Michael Heß, Consultant der ORDIX AG, verriet in seinem Beitrag „Loadbalancing und Clustering mit Tomcat 6“, wie man dieser Anforderung gerecht werden kann.

Ausgehend von einer einzelnen Tomcat Instanz, wurden die benötigten Konfigurationsschritte für eine komplette Clusterumgebung mit vorgeschaltetem Apache Webserver aufgezeigt.

ORDIX Referent Oliver Kaluza präsentierte „Vererbungsstrategien und Polymorphie mit der Java Persistence API“. Die Prinzipien der Vererbung und Polymorphie sind Grundlagen der objektorientierten Programmierung. Sie waren bisher für Entity Beans und in der EJB Spezifikation nicht vorgesehen. Dieses Manko wird mit EJB 3.0 und dem darin definierten Java Persistence API (JPA) beseitigt.

Die Session zeigte anhand von Beispielen auf, wie auf dieser neuen Basis mit Entity Beans komplexe persistente Objektnetze zu modellieren und technisch umzusetzen sind.

Sie interessieren sich für die von ORDIX vorgestellten Themen, konnten aber bei den Vorträgen nicht dabei sein oder haben spezielle Java-Fragen? Wir beraten Sie gern! Schicken Sie einfach eine E-Mail an info@ordix.de.

IBM DB2 UDB Version 9.1 „Viper“ – New Features (Teil II):

Mehr Sicherheit in DB2 durch LBAC

Dieser Artikel richtet sich an Datenbank- und Systemadministratoren sowie Sicherheitsbeauftragte.

In der ORDIX News Ausgabe 1/2007, Seite 37, gaben wir einen ersten Überblick über die zahlreichen Neuerungen der DB2 Version 9.1. Die weiteren Artikel dieser Reihe gehen nun mehr in die Tiefe einzelner neuer Funktionen, beschreiben technische Details und verdeutlichen den Umgang anhand kleinerer Beispiele. In diesem Artikel beginnen wir mit interessanten Einzelheiten über die Neuerung LBAC!

Was ist LBAC?

LBAC steht für „Label-Based Access Control“ und ist in DB2 9.1 eine neue Funktion aus dem Bereich Sicherheit. Mit Hilfe dieser „kennzeichenbasierten Zugriffskontrolle“ können für einzelne Zeilen und/oder Spalten einer Datenbanktabelle Lese- und Schreibrechte für einzelne Benutzer vergeben und gesteuert werden. Somit kann der Zugriff auf eine Tabelle erlaubt werden, bestimmte Bereiche dieser Tabelle bleiben allerdings für einzelne Benutzer im Verborgenen.

Beispiel I: Sicherheit auf Spaltenebene

Im ersten Beispiel nehmen wir an, dass eine Tabelle für Mitarbeiterdaten existiert. Eine Spalte dieser Tabelle beinhaltet das Gehalt. Mittels LBAC kann nun festgelegt werden, dass der Vorstand des Unternehmens auf die Spalte „Gehalt“ Lese- und Schreibzugriff bekommt. Die Mitarbeiter der Verwaltung erhalten lediglich einen Lesezugriff und alle übrigen Mitarbeiter bekommen überhaupt keinen Zugriff auf die Spalte mit dem Gehalt.

Beispiel II: Sicherheit auf Zeilenebene

Im zweiten Beispiel gehen wir davon aus, dass eine Tabelle existiert, in der alle Tätigkeiten von verschiedenen Projekten erfasst werden.

Mittels LBAC wird nun festgelegt, dass ein Mitarbeiter Datensätze, die sich auf Projekte beziehen, an denen auch er tätig ist, lesend und schreibend zugreifen darf. Projektleitern wird erlaubt, die Datensätze von den Projekten einzusehen, für die sie die Leitungsposition übernehmen. Der Vorstand erhält Lesezugriff auf alle Datensätze der Tabelle, jedoch darf auch er keine Veränderungen vornehmen.

Funktionsweise

Die Zugriffssteuerung erfolgt über so genannte „Security Label“ (Sicherheitskennzeichen), die sowohl Benutzern als auch Datensätzen einer Tabelle zugewiesen werden.

Möchte nun ein Benutzer auf einen Datensatz zugreifen, wird sein Sicherheitskennzeichen mit dem des Datensatzes verglichen. Nur bei einer Übereinstimmung erhält der Benutzer Zugang zu den geforderten Tabellendaten.

Wenn die Sicherheitskennzeichen nicht übereinstimmen, sieht es für den Benutzer entweder so aus, als ob der Datensatz nicht existiert (LBAC auf Zeilenebene) oder aber er bekommt eine Fehlermeldung für unzureichende Berechtigung (LBAC auf Spaltenebene).

Bei der Vergabe des Sicherheitskennzeichens wird zusätzlich festgelegt, ob der Benutzer einen Vollzugriff auf die Daten erhält oder aber lediglich einen lesenden Zugriff.

Voraussetzungen

Sicherheitskennzeichen dürfen nicht von jedem Benutzer zugewiesen werden, sondern können

```
CREATE SECURITY LABEL COMPONENT <name>
{
ARRAY[ <element>, <element>, ...] |
TREE( <element> ROOT, <element> UNDER <element>, ...) |
SET{ <element>, <element>, ...}
}
```

Abb. 1: Syntax zur Erstellung einer `security label component`.

```
create security label component slc_gehalt
   set {'Vertraut'};
```

Abb. 2: Security Label Component vom Typ set (Beispiel I).

```
create security label component slc_projekte
   tree( 'Vorstand'     ROOT,
         'PL_Projekt_A' UNDER 'Vorstand',
         'PL_Projekt_B' UNDER 'Vorstand',
         'Projekt_A'    UNDER 'PL_Projekt_A',
         'Projekt_B'    UNDER 'PL_Projekt_B',
         'Projekt_C'    UNDER 'Vorstand'
       );
```

Abb. 3: Security Label Component vom Typ tree (Beispiel II).

ausschließlich durch einen Sicherheitsadministrator erfolgen. Bei diesem handelt es sich um einen Benutzer, der über die SECADM-Berechtigung verfügt, die ebenfalls mit der DB2 Version 9.1 neu eingeführt wurde (siehe ORDIX News 1/2007, Seite 38).

Benötigte LBAC-Komponenten

Sicherheitskennzeichen sind aber nur ein Bestandteil von LBAC. Daneben gibt es noch zwei weitere wichtige Bestandteile, so dass zum Einrichten von LBAC die folgenden Komponenten benötigt werden:

- Security Label Component
- Security Policy
- Security Label

Bei der Konfiguration von LBAC muss die obige Reihenfolge eingehalten werden.

Security Label Component

Die Komponente Security Label Component dient der Darstellung von Vertraulichkeitsstufen und -strukturen. Insgesamt gibt es folgende drei verschiedene Typen:

- ARRAY: dient der Speicherung verschiedener Vertraulichkeitsstufen. Je weiter vorne ein Element im Array eingeordnet ist, desto vertraulicher ist es.
- TREE: dient der Darstellung einer Hierarchie von Vertraulichkeitsstufen. Es gibt ein so genanntes ROOT-Element, dem weitere Elemente untergeordnet sind. Die untergeordneten Elemente können wiederum weitere Elemente aufnehmen.
- SET: definiert eine Menge von Vertraulichkeitsstufen. Dadurch kann hier nur die Überprüfung durchgeführt werden, ob ein Element in der SET-Liste vorhanden ist. Es existieren keinerlei Hierarchien.

```
CREATE SECURITY POLICY <name>
   COMPONENTS <security label component>
   WITH DB2LBACRULES
   {
      restrict not authorized write security label
     |overwrite not authorized write security label
   };
```

Abb. 4: Syntax zur Erstellung einer `security policy`.

```
create security policy sp_gehalt
   components slc_gehalt
   with db2lbacrules
   restrict not authorized write security label;
```

Abb. 5: Die Security Policy speichert die Komponente für die Sicherheit auf Spaltenebene (Beispiel I).

```
create security policy sp_projekte
   components slc_projekte
   with db2lbacrules
   restrict not authorized write security label;
```

Abb. 6: Die Security Policy speichert die Komponente für die Sicherheit auf Zeilenebene (Beispiel II).

```
CREATE SECURITY LABEL <security policy>.<name security label>
   COMPONENT <security label component> '<Vertrauensstufe>';
```

Abb. 7: Syntax zur Erstellung eines `security label`.

Die Syntax zur Erstellung einer Security Label Component“ zeigt Abbildung 1.

Für das Beispiel I ist es ausreichend, eine Security Label Component zu deklarieren, die vom Typ SET ist (siehe Abbildung 2).

Abbildung 3 zeigt die Security Label Component für Beispiel II. Damit die Hierarchiestufen aus diesem Beispiel dargestellt werden können, wird die Security Label Component TREE verwendet, bei der an oberster Stelle der Vorstand steht. Danach folgen die Projektleiter, denen wiederum Projekte unterstellt sind. Besitzt ein Projekt keinen Projektleiter, ist es direkt dem Vorstand untergeordnet.

Security Policy

Die Security Policy dient der Aufnahme von Sicherheitskennzeichen und beschreibt die Richtlinien und Bedingungen für die eigentliche Zugriffskontrolle. Wichtigster Bestandteil einer Security Policy ist die gerade zuvor kennengelernte Security Label Component. Ohne diese Komponente kann eine Security Policy nicht erstellt werden, da hier die oben erwähnten Richtlinien durch die Vertraulichkeitsstufen festgelegt werden. Einzelne Sicherheitskennzeichen werden durch Zuordnung zu einer Security Policy in eine Vertraulichkeitsstufe eingeordnet.

Eine Security Policy kann verschiedenen Tabellen zugeordnet werden. Diese Zuordnung hat durch die SQL-Statements **`CREATE`** oder **`ALTER`** zu erfolgen. (Achtung, eine Tabelle kann immer nur eine Security Policy aufnehmen).

Die Syntax zur Erstellung einer Security Policy finden Sie in Abbildung 4. Bei der Erstellung gibt es zwei Optionen, die die Vergabe der Sicherheitskennzeichen regeln:

1. **`restrict not authorized write security label`**
   Das Einfügen oder Aktualisieren eines Datensatzes schlägt fehl, wenn der Benutzer nicht autorisiert ist, explizit ein Sicherheitskennzeichen für den Datensatz zu schreiben.
2. **`overwrite not authorized write security label`**
   Beim Einfügen oder Aktualisieren eines Datensatzes wird diesem automatisch das Sicherheitskennzeichen des Benutzers vergeben.

Die Abbildungen 5 und 6 zeigen die Erstellung der Security Policy für die Beispiele I und II.

Security Label

Das Security Label ist die wichtigste Komponente von LBAC, da anhand eines Vergleiches dieser Kennzeichen entschieden wird, ob ein Benutzer einen Datensatz/Spaltenwert sehen darf oder nicht.

Als Basis für ein Sicherheitskennzeichen dient immer eine Security Policy. Bei der Namensgebung des Kennzeichens muss die Policy mit angeführt werden. Die Syntax entspricht dabei zum Beispiel dem **select** auf eine Tabelle eines anderen Schemas. Zuerst wird der Name der Policy verlangt, dann folgt ein Punkt (.) und anschließend der Name des Labels. Durch diese Vorgehensweise ist direkt festgelegt, welcher Security Policy ein Sicherheitskennzeichen zugeordnet ist.

Weiterhin wird beim Anlegen des Sicherheitskennzeichens die Angabe der Security Label Component verlangt. Jedes Kennzeichen kann immer nur eine Vertraulichkeitsstufe einer Security Label Component beinhalten.

Die Syntax zur Erstellung eines Sicherheitskennzeichens zeigt Abbildung 7.

Die Abbildungen 8 und 9 zeigen die Erstellung der Sicherheitskennzeichen für die beiden Beispiele.

```
create security label sp_gehalt.sl_gehalt
   component slc_gehalt 'Vertraut';
```

Abb. 8: Security Label für die Security Komponente set (Beispiel I).

```
create security label sp_projekte.projekt_a
   component slc_projekte 'Projekt_A';

create security label sp_projekte.projekt_b
   component slc_projekte 'Projekt_B';

create security label sp_projekte.projekt_c
   component slc_projekte 'Projekt_C';

create security label sp_projekte.pl_projekt_a
   component slc_projekte 'PL_Projekt_A';

create security label sp_projekte.pl_projekt_b
   component slc_projekte 'PL_Projekt_B';

create security label sp_projekte.vorstand
   component slc_projekte 'Vorstand';
```

Abb. 9: Security Label für die Security Komponente tree (Beispiel II).

LBAC-Aktivierung

Neben den Daten müssen natürlich auch die Benutzer mit einem entsprechenden Sicherheitskennzeichen ausgestattet werden. Die Zuweisung wird mit Hilfe des **Grant**-Kommandos vorgenommen.

Dabei kann nun auch festgelegt werden, ob der Benutzer Vollzugriff (for all access) oder nur Lesezugriff (for read access) auf die Daten erhält. Die Abbildungen 10 und 11 zeigen mögliche **Grant**-Kommandos für die beiden Beispiele.

```
grant security label sp_gehalt.sl_gehalt
   to user klose
   for all access;

grant security label sp_gehalt.sl_gehalt
   to user gruber
   for read access;
```

Abb. 10: Mögliches Grant-Kommando: Vergabe von Security Labels für die Sicherheit auf Spaltenebene (Beispiel I).

Tabelle für LBAC anpassen

Erster Schritt, um eine Tabelle mit LBAC zu sichern, ist die Zuweisung einer Security Policy. Diese Zuweisung kann mit dem **CREATE**- oder mit dem **ALTER TABLE**-Befehl erfolgen. Im Weiteren muss man unterscheiden, ob die Tabelle auf Zeilen- oder auf Spaltenebene geschützt werden soll.

Wenn eine Tabelle auf Spaltenebene geschützt werden soll, so muss die zu schützende Spalte um die Klausel **SECURED WITH** und um den Namen des Sicherheitskennzeichens erweitert werden. Die Abbildung 12 zeigt, wie die Spalte „Gehalt“ der Mitarbeitertabelle mit dem Kennzeichen „sl_gehalt“ geschützt wird.

Wenn eine Tabelle auf Zeilenebene geschützt werden soll, muss eine zusätzliche Spalte vom Datentyp **DB2SECURITYLABEL** angelegt werden. In dieser wird dann das Sicherheitskennzeichen des Datensatzes gespeichert, welches bei einer Abfrage mit dem Kennzeichen des Benutzers verglichen wird (siehe Abbildung 13).

```
grant security label sp_projekte.projekt_a
   to user joschko for all access;

grant security label sp_projekte.vorstand
   to user klose for read access;

grant security label sp_projekte.pl_projekt_b
   to user gruber for read access;
```

Abb. 11: Mögliches `Grant`-Kommando: Vergabe von Security Labels für die Sicherheit auf Zeilenebene (Beispiel I).

```
create table mitarbeiter (
   nr        int,
   name      varchar(25),
   adresse   varchar(25),
   gehalt    int SECURED WITH sl_gehalt
) security policy sp_gehalt;
```

Abb. 12: Sicherheit auf Spaltenebene (Beispiel I).

```
create table projekttaetigkeiten
(
   nr           DECIMAL(6)   NOT NULL,
   projekt      DECIMAL(6)   NOT NULL,
   mitarbeiter  DECIMAL(4)   NOT NULL,
   datum        DATE,
   taetigkeit   VARCHAR(200),
   security     DB2SECURITYLABEL
) security policy sp_projekte;
```

Abb. 13: Sicherheit auf Zeilenebene (Beispiel II).

Glossar

Alter	Mit dem SQL-Befehl **`alter`** können innerhalb einer Datenbank Objekte (Tabellen, Indizes, Views) *bearbeitet* werden.
Create	Mit dem SQL-Befehl **`create`** können innerhalb einer Datenbank Objekte (Tabellen, Indizes, Views) *angelegt* werden.
Grant	Mit dem SQL-Befehl **`grant`** können Zugriffsrechte innerhalb einer Datenbank *vergeben* werden.
LBAC	Label-Based Access Control. Ermöglicht die Steuerung von Zugriffen auf bestimmte Datensätze oder einzelne Spaltenwerte von Datensätzen.
Revoke	Mit dem SQL-Befehl **`revoke`** können Zugriffsrechte innerhalb einer Datenbank *entzogen* werden.
SECADM	Eine Sicherheitsadministrator-Berechtigung, die Datenbankbenutzern zugeordnet werden kann.
SQL	Structured Query Language. Sie dient als Kommunikationsinstrument mit der Datenbank.

Beim Hinzufügen der zusätzlichen Spalte vom TYP **`DB2SECURITYLABEL`** wird für jeden Wert, der sich bereits in der Tabelle befindet, das Kennzeichen des Benutzers in dieser Spalte eingetragen.

Sicherheitskennzeichen zuweisen

Beim **`Insert`** wird jedem Datensatz ein entsprechendes Kennzeichen zugewiesen. Dieses ist abhängig vom Label des Benutzers, welcher den Datensatz einfügen möchte. Vorausgesetzt wird natürlich, dass der Benutzer im Besitz eines Sicherheitskennzeichens mit Schreibberechtigung für die Tabelle ist.

Explizit kann einem Datensatz in der Spalte vom Typ DB2SECURITYLABEL das Sicherheitskennzeichen auch durch die Funktion **`SECLABEL_BY_NAME()`** zugeordnet werden. Als Übergabeparameter werden die Security Policy und das Security Label verlangt:
`SECLABEL_BY_NAME('sp_projekt', 'sl_projekt_a')`

Bei einem **`select`** auf diese Spalte erhält man das jeweilige Sicherheitskennzeichen des Datensatzes, sofern man berechtigt ist, den Datensatz auch zu sehen.

Fazit

Hinter LBAC verbirgt sich mit Sicherheit eine sehr gute Methode, um bestimmte Daten vor unberechtigten Zugriffen zu schützen. Die Möglichkeit, Benutzern einzelne Datensätze einer Tabelle zu entziehen, ist alleine mit dem Vergeben (**`grant`**) bzw. Entziehen (**`revoke`**) von Rechten nicht zu realisieren.

Weiterhin kann man eventuell auch auf die Erstellung der ein- oder anderen View verzichten, indem man bereits zuvor Einschränkungen an Tabellen durch LBAC vornimmt.

Anfangs erscheint LBAC vielleicht ein bisschen kompliziert und unüberschaubar. Auch dieser Artikel und die darin enthaltenen Beispiele beschreiben lediglich einen kleinen Teil dieser neuen Funktion. Je intensiver man sich allerdings mit dem Thema auseinandersetzt und viele Funktionen einfach selbst ausprobiert, desto verständlicher und einfacher zu handhaben wird das Ganze.

Thorsten Schuhmacher (info@ordix.de).

Oracle und XML - Ein besonderer Cocktail (Teil III):

Performance Tuning

In der ORDIX News 1/2007, Seite 52, haben wir den Datentyp XMLType vorgestellt. In diesem Artikel beleuchten wir die Möglichkeiten der Optimierung von Abfragen. Dabei soll im Detail vor allem auf das Query Rewrite und die Indizierung von XMLType eingegangen werden.

Dieser Artikel richtet sich an Datenbankentwickler und Softwarearchitekten, die XML im Zusammenspiel mit Oracle Datenbanken einsetzen.

Werden XML-Dokumente in der Datenbank als XMLType abgelegt, so kann ihr Inhalt in Select-Statements einbezogen werden. An dieser Stelle werden die Möglichkeiten der Optimierung solcher Abfragen durch Query Rewrite und Indizierung vorgestellt.

Alle Beispiele beziehen sich auf die Daten der Tabelle, deren Insert-Statement in Abbildung 1 dargestellt ist. Das zugehörige, registrierte XML-Schema zeigt Abbildung 2. Die Beispiele wurden unter Oracle 10g erstellt.

Objektrelational oder CLOB?

Schon die Speicherungsart des XML-Dokuments beeinflusst die Performance der Abfragen. In der letzten Ausgabe der ORDIX News wurden die beiden Speicherungsarten eingehend vorgestellt. Aber wann sollte welche Speicherungsart verwendet werden, um möglichst performant zu arbeiten?

Um diese Frage zu beantworten, sollte zunächst überlegt werden, welche XML-Daten abgespeichert werden und was nach dem Speichern mit den Daten passieren soll.

1. Unstrukturierte Speicherung als CLOB

Die Speicherung in einem CLOB ist dann vorteilhaft, wenn die Struktur der XML-Dokumente stark variiert oder wenn große Datenmengen zu speichern sind. Veränderungen in der Struktur sind für eine strukturierte Speicherung nicht hinnehmbar: Jedes gespeicherte Dokument muss zum registrierten XML-Schema passen.

Soll immer nur auf das gesamte XML-Dokument zugegriffen werden, so ist in jedem Fall die Speicherung in einem CLOB zu empfehlen. Damit lässt sich der Aufwand beim Einfügen und bei Abfragen verringern. Bei der strukturierten Speicherung muss das Dokument beim Einfügen analysiert und in mehrere Tabellen gesplittet werden. Bei Abfragen werden Dokumente wieder aus den einzelnen Tabellen zusammengesetzt. Diese Vorgänge können gerade bei sehr großen XML-Dokumenten viel Zeit in Anspruch nehmen.

2. Strukturierte, objektrelationale Speicherung

Muss dagegen (über XPath) auf einzelne Elemente des XML-Dokuments zugegriffen werden, so ist es vorteilhaft das XML-Dokument objektrelational abzuspeichern. Durch das Splitten des Dokuments auf Tabellen muss bei Abfragen oder Updates nicht mehr das gesamte XML-Dokument gelesen bzw. geschrieben werden. Ein gezielter Zugriff auf einzelne Elemente ist durch die strukturierte Speicherung möglich. Nur bei dieser Art der Speicherung ist ein Query Rewrite möglich.

Query Rewrite

Um per SQL auf einzelne Knoten des XML-Dokuments zuzugreifen, werden verschiedene XMLType-Funktionen verwendet. Die Lokalisierung der Knoten erfolgt bei den Funktionen durch XPath. Beispiele sind u. a.

- **`extract:`**
  Extrahiert Knoten aus dem Dokument.
- **`extractValue:`**
  Extrahiert genau einen Knoten.
- **`existsNode:`**
  Prüft, ob der Knoten existiert.

Bei der objektrelationalen Speicherung werden bei der Registrierung eines XML-Schemas diverse Objekte in der Oracle Datenbank angelegt. Wie zuvor beschrieben, werden die XML-Dokumente beim Insert gesplittet und in mehreren Datenbankobjekten gespeichert. Die Oracle Datenbank verwendet, sofern es möglich ist, automatisch den direkten Objektzugriff. D. h. SQL-Anweisungen werden so umgeschrieben, dass nicht auf das gesamte XML-Dokument zugegriffen wird, sondern auf die einzelnen Objekte, in denen Teile des XML-Dokuments gespeichert wurden.

```
Insert into auto values
 (1,'Peugeot',XMLTYPE('<?xml version="1.0" encoding="UTF-8"?>
 <Auto xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" http://www.ordix.de/herstel-
lerliste.xsd">
  <Herstellerliste>
  <Hersteller id="1">
  <Name>Motorenwerke R&#252;desheim</Name>
  </Hersteller>
  <Hersteller id="2">
  <Name>Best-Blech</Name>
 </Hersteller>
 </Herstellerliste>
 </Auto>'));
```

Abb. 1: Ausgangstabelle für alle Beispiele.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" element
FormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Auto">
    <xs:complexType>
   <xs:sequence>
        <xs:element name="Herstellerliste">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Hersteller" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name"/>
             </xs:sequence>
          <xs:attribute name="id"/>
        </xs:complexType>
      </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Abb. 2: In den Beispielen verwendetes XML-Schema.

```
Select extractvalue(herstellerliste,'Auto/Herstellerliste/Her-
steller[@id="1"]/Name/text()') "Name" from auto;
```

Abb. 3: SQL-Anweisung mit XML-Funktion.

Diese Art des Umschreibens von SQL-Anweisungen mit XMLType-Funktionen wird Query Rewrite genannt. In Abbildung 3 ist eine SQL-Anweisung zu sehen, die den Namen des Herstellers mit der ID 1 ausgibt. In Abbildung 4 ist sinngemäß dargestellt, wie diese Anweisung von Oracle intern umgeschrieben wird.

Voraussetzungen für Query Rewrite

Ein Query Rewrite kann natürlich nur bei der objektrelationalen Speicherung stattfinden, denn bei der dokumentenbasierten Speicherung als CLOB sind die Strukturen der XML-Dokumente der Datenbank nicht bekannt. Außerdem kann ein Rewrite nicht bei allen SQL-Funktionen durchgeführt werden. Im Folgenden sind einige SQL-Funktionen dargestellt, bei denen u. a. ein Query Rewrite möglich ist:

- **extract**
- **existsNode**
- **extractValue**
- **updateXML**
- **insertChildXML**
- **deleteXML**
- **XMLSequence**

Nicht alle XPath-Ausdrücke können in ein SQL-Statement umgewandelt werden. Die wesentlichen Einschränkungen sind:

- Auf Funktionen innerhalb des XPath-Ausdrucks muss (bis auf wenige Ausnahmen) verzichtet werden.
- Wildcards, die zu mehr als einem XML-Knoten führen, sollten nicht verwendet werden.
- Außer den XPath-Achsen „child“ und „attribute“ sollten keine XPath-Achsen verwendet werden.
- XPath-Variablen sollten nicht verwendet werden.
- In der Hierarchie darf immer nur zu den Kind-Elementen navigiert werden.

Kein Query Rewrite?

Werden die zuvor genannten Anforderungen nicht erfüllt, so kann kein Query Rewrite stattfinden. Die SQL-Anweisungen werden aber

natürlich trotzdem ausgeführt. In diesem Fall wird zunächst das gesamte XML-Dokument gelesen. Anschließend wird im Hauptspeicher ein DOM-Baum aufgebaut und anhand des DOM-Baums die gewünschte Operation durchgeführt.

Die Erstellung eines DOM-Baums ist sehr ressourcenintensiv. Der DOM-Baum ist im Hauptspeicher etwa 5 bis 10 Mal so groß, wie das XML-Dokument auf der Festplatte. Ist das XML-Dokument auf der Festplatte also 500 KB groß, so benötigt es im Hauptspeicher schon 2,5 bis 5 MB. Wenn nun davon ausgegangen wird, dass in jedem Datensatz der durchsuchten Tabelle ein XML-Dokument analysiert werden muss, so führt dies leicht zu einem enormen Aufwand.

Die Erstellung von DOM-Bäumen sollte auf jeden Fall durch ein fehlerfrei laufendes Query Rewrite verhindert werden, wenn die SQL-Anweisung performant durchlaufen soll.

Tracing

Dauert eine SQL-Abfrage sehr lange, sollte zunächst untersucht werden, ob die Datenbank für diese Abfrage Query Rewrite anwenden kann. Dazu kann man, wie in Abbildung 5 beschrieben, dafür sorgen, dass eine Fehlermeldung ausgegeben wird, wenn ein Query Rewrite nicht möglich ist.

Wird jetzt eine SQL-Abfrage abgesetzt, bei der kein Query Rewrite möglich ist (z. B. eine Abfrage mit einer XPath-Wildcard), so wird, wie in Abbildung 6 dargestellt, eine Fehlermeldung ausgegeben. Allerdings lässt sich anhand der Fehlermeldung nicht identifizieren, warum das Query Rewrite bei der Abfrage nicht möglich war. Dafür muss ein erweitertes Tracing, wie in Abbildung 7 beschrieben, eingeschaltet werden.

War bei der Abfrage, wie in Abbildung 6, kein Query Rewrite möglich, so kann man anhand der Trace Files erkennen, welcher Teil der Abfrage zu dem Problem geführt hat. Damit die Trace Files geschrieben werden, muss man dafür sorgen, dass das Tracing von Oracle eingeschaltet ist. Im Oracle SQL Developer kann man das Statement z. B. über den Button „Autotrace“‘ ausführen und so sicherstellen, dass Trace Files geschrieben werden.

Die Trace-Dateien findet man, sofern man die Standardeinstellungen von Oracle beibehalten hat, im Verzeichnis **<Installationsverzeichnis>/admin/<Name der DB>/udump**. In Abbildung 8 ist ein Auszug aus einem solchen Trace File dargestellt.

```
select SYS_NC000013$ from auto where....;
```

Abb. 4: Sinngemäße Darstellung einer von Oracle umgeschriebenen SQL-Anweisung.

```
alter session set events ='19021 trace name context forever,
level 0x1';
```

Abb. 5: Einschalten des Tracing für Query Rewrite.

```
select extractvalue(herstellerliste,'Auto/Herstellerliste/Her-
steller[@id="1"]/Name/text()') "Name" from auto;

from auto
      *
ERROR at line 2:
ORA-19022: XML XPath functions are disabled
```

Abb. 6: Ausgabe für Anweisung ohne Rewrite.

```
alter session set events='19027 trace name context forever,
level 0x2000';
```

Abb. 7: Einschalten des erweiterten Tracing für Query Rewrite.

In unserem Beispiel sehen wir im Trace File die Meldung „anytype with child axis“. Dies deutet darauf hin, dass im XML-Schema bei einem Element vergessen wurde, einen Datentyp anzugeben und trotzdem der Kindknoten angesprochen wurde. In einem solchen Fall ist kein Query Rewrite möglich.

Auch welcher Knoten betroffen ist, wird im Trace File mitgeteilt. Es handelt sich um den Knoten „Name“. In Abbildung 2 ist zu sehen, dass dem Element „Name“ tatsächlich kein Datentyp zugewiesen ist.

Wurde anhand des Tracings identifiziert, was ein Query Rewrite verhindert, kann die SQL-Anweisung so umformuliert werden, dass ein Query Rewrite möglich wird. In unserem Beispiel muss also das XML-Schema angepasst werden. Für jedes Element und Attribut sollte der Datentyp eingetragen werden. In Abbildung 9 ist ein Auszug aus dem angepassten XML-Schema zu sehen.

```
...
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name" type="xs:string"/>

  </xs:sequence>
  <xs:attribute name="id" type="xs:int"/>
</xs:complexType>
...
```

Abb. 8: Auszug aus der Trace-Datei.

```
...
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name" type="xs:string"/>

  </xs:sequence>
  <xs:attribute name="id" type="xs:int"/>
</xs:complexType>
...
```

Abb. 9: Auszug aus XML-Schema, das Query Rewrite erlaubt.

```
select extractvalue(herstellerliste,'Auto/Herstellerliste/Her-
steller[@id="1"]/Name/text()') "Name" from auto;
Name
--------------------------------------------------------------------
Motorenwerke Rüdesheim
```

Abb. 10: Abfrage, für die Query Rewrite durchgeführt wird.

```
CREATE TABLE auto(
   autoid NUMBER,
   autotyp VARCHAR2(100),
   herstellerliste XMLTYPE
)
XMLTYPE COLUMN herstellerliste
XMLSCHEMA "http://www.ordix.de/herstellerliste.xsd"
ELEMENT "Auto"
varray herstellerliste.xmldata."Herstellerliste"."Hersteller"
store as table hersteller_oct ((primary key (nested_table_id,
array_index)) organization index)
/
```

Abb. 11: Create-Statement bei dem eine 1:n-Beziehung in eine eigene Tabelle ausgelagert wird.

```
create index idx_herstellerid on hersteller_oct("id");
```

Abb. 12: Anlegen eines Indexes.

Nach dem Umformulieren der SQL-Anweisung oder in unserem Fall des XML-Schemas kann bzw. sollte über das Tracing kontrolliert werden, ob die Umformulierung zu einem Query Rewrite führt. Wird beim Ausführen der Abfrage keine Fehlermeldung mehr ausgegeben, so kann davon ausgegangen werden, dass ein Query Rewrite erfolgt ist.

In Abbildung 10 ist die Abfrage dargestellt, für die das Query Rewrite durch die Oracle Datenbank durchgeführt werden konnte.

Indizierung

Query Rewrite ist nicht nur für sich allein gesehen für die Performance von SQL-Anweisungen auf XML-Dokumente in der Datenbank wichtig. Auch auf die Indizierbarkeit von Elementen der XML-Daten wirkt sich das Query Rewrite aus. Ohne ein Query Rewrite an dieser Stelle ist ein Index immer wirkungslos.

Es sollte also unbedingt darauf geachtet werden, dass ein Query Rewrite erfolgt. Abbildung 11 zeigt, wie ein Index auf XML-Daten angelegt werden kann. In unserem Beispiel ist es sinnvoll, einen Index auf das Attribut „ID“ anzulegen. Auch beim Anlegen von Indexen wendet die Datenbank das Query Rewrite an.

Im Beispiel gibt es, wie in Abbildung 2 zu sehen ist, eine 1:n-Beziehung. In der Herstellerliste können sich mehrere Hersteller befinden. Die 1:n-Beziehung wird objektrelational als Array abgelegt. Standardmäßig wird ein Array als Byte-Strom in einem Attribut als CLOB abgespeichert. Um hier indizieren zu können, muss die 1:n-Beziehung in eine eigene Tabelle ausgelagert werden. Dies ist schon beim Anlegen der Tabelle zu beachten. In Abbildung 11 ist zu sehen, wie eine solche 1:n-Beziehung in eine eigene Tabelle ausgelagert werden kann.

Beim Anlegen der Tabelle wird zusätzlich die „Nested Table“ **`hersteller_oct`** angelegt. Hier werden die Elemente der 1:n-Beziehung abgelegt. Für jedes Element ist eine Zeile in der Tabelle vorhanden. Auf diese Tabelle wird nun, wie in Abbildung 12 beschrieben, der Index angelegt, der bei Abfragen auf die Tabelle „auto“, die als Suchkriterium Elemente der 1:n-Beziehung verwendet, genutzt wird.

Für die Tabelle sollten Statistiken erzeugt und vor allem auch aktuell gehalten werden, damit der Optimizer den Index auch tatsächlich richtig nutzt.

Glossar

CLOB	CLOB (Character Large Object) ist der Datentyp für ein Datenbankfeld zur Speicherung von großen Textdaten (bis zu 4 GB).
DOM	DOM (Document Object Model) ist ein API für den Zugriff auf XML-Dokumente. Das API erlaubt dynamisch die Struktur, das Layout und den Inhalt von XML-Dokumenten zu ändern.
XML	Extensible Markup Language (XML). XML ist eine so genannte META-Sprache zur Beschreibung von Dokumenten. Ein Vorteil von XML ist der vereinfachte Austausch von Daten, da XML-Formate in einer strengen Grammatik definiert werden können und so die Implementierung von zuverlässigen Schnittstellen erlaubt.
XML-Schema	Ein XML-Schema beschreibt die Struktur von XML-Dokumenten und erlaubt ihre inhaltliche Überprüfung.
XPath	XPath stellt Funktionen und Ausdrücke zur Verfügung, um Knoten innerhalb von XML-Dokumenten zu lokalisieren. Mit XPath können auch Ausdrücke ausgewertet und Berechnungen durchgeführt werden.

Vorgehensweise beim Tuning von Abfragen

Natürlich gibt es keine allgemeingültige Vorgehensweise, mit der Abfragen in jedem Fall beschleunigt werden können. Dazu bedarf es immer einer genauen Analyse im konkreten Projekt. Allerdings lässt sich aus den oben dargestellten Möglichkeiten des Tunings eine allgemeine Vorgehensweise ableiten:

1. Speicherungsform prüfen.
2. Query Rewrite sicherstellen.
3. Indexe erstellen, Statistiken aktualisieren, Optimizer Hints setzen, usw.

Fazit

Wie der Artikel zeigt, gibt es in der Oracle XML-DB einige Besonderheiten, die bei der Optimierung der Performance zu beachten sind. Konkrete Maßnahmen sind naturgemäß stark vom Projekt abhängig. Die dargestellten Einflussgrößen sind bei Abfragen mit XML-Dokumenten unbedingt zu berücksichtigen. Anhand von Testreihen sollte immer geprüft werden, ob die durchgeführten Maßnahmen tatsächlich zum Erfolg führten.

Kathrin Hammerschmidt (info@ordix.de).

Impressum

Herausgeber:
ORDIX AG
Aktiengesellschaft für Softwareentwicklung, Beratung, Schulung und Systemintegration, Paderborn

Redaktion: Helma Jenniches, Sascia Brinkmann

V.i.S.d.P.: Benedikt Georgi, Wolfgang Kögler

Anschrift der Redaktion:
ORDIX AG
Westernmauer 12 - 16
33098 Paderborn
Tel.: 05251 1063-0
Fax: 0180 1673490

Gestaltung/Layout: Sascia Brinkmann

Auflage: 9.000

Druck: Druckerei Reike GmbH, Paderborn

Autoren dieser Ausgabe:
Marius Dorlöchter, Christian Fertsch, Benedikt Georgi, Klaus Günther, Kathrin Hammerschmidt, Stefanie Heither, Michael Heß, Andreas Jordan, Matthias Jung, Oliver Kaluza, Wolfgang Kögler, Beate Künneke, Roman Peters, Rainer Restat, Dustin Schmitt, Markus Schreier, Thorsten Schuhmacher

Die Zeitschrift ORDIX News wird von der ORDIX AG an ausgesuchte Kunden verteilt und kann für 2,20 Euro bestellt werden. Sie finden sowohl die neueste Ausgabe als auch ältere Ausgaben im Archiv der ORDIX News im Internet unter: **http://www.ordix.de.** Schauen Sie mal rein!

Der Kontakt zu unseren Lesern ist uns sehr wichtig. Für Anregungen, Kritik und Anmerkungen zu den Themen, aber auch für interessante Ideen sind wir immer offen und dankbar. Wir freuen uns auf Ihr Feedback an **redaktion@ordix.de**.

ORDIX AG

Ergreifen Sie die Chance ...

... zur Optimierung Ihrer Datenbanken!

Lange Antwortzeiten? Schlechte Verfügbarkeit? Sicherheitslücken?
Wir optimieren Ihre Datenbanksysteme ganzheitlich!

Wir bieten:

- Installation, Konfiguration & Migration
- Performance-Analysen & Tuning
- Hochverfügbarkeitslösungen
- Entwicklung
- Backup- & Recovery-Konzepte
- Administration, Monitoring
- Seminare, Coaching

... für Oracle, Informix, DB2, MS SQL Server und MySQL.

Weitere Informationen finden Sie im Internet unter **http://www.ordix.de.**
Für ein persönliches Gespräch wenden Sie sich an Jana Kreschel: vertrieb@ordix.de