Nr. 5 | November 2018 | ISSN 2366-7915 | www.doag.org

Red Stack Magazin

DOAG SOUG swiss oracle user group AOUG AUSTRIAN ORACLE USER GROUP

Tuning

SPEED IT UP!

Aus der Praxis

Einfachen und effektiven Code entwickeln

Im Interview

Ralf Schmilewski, myToys Group

Aktuell

Next Generation Engineered Systems

Martin Klier
DOAG Themenverantwortlicher Hochverfügbarkeit

Liebe Mitglieder, liebe Leserinnen und Leser,

das Tuning von IT-Systemen, also auch von Datenbanken, ist eine ambivalente Tätigkeit: Es treffen dabei regelmäßig hoher Druck, reizvolle Herausforderungen, große Wichtigkeit und Freude an der Arbeit aufeinander.

Welche Veränderung die Plattform-Verlagerung auf eine der vielen Cloud-Lösungen bringen wird, ist noch nicht endgültig abzusehen. Sicher ist, dass ein Cloud-Anbieter mit vielen laufenden Instanzen großes Know-how über die Leistungsfähigkeit der eingesetzten Software gewinnen, dieses algorithmisch auswerten und günstigstenfalls wieder aufbereitet an die Kunden zurückliefern kann. Ob eine von Menschen fortlaufend dynamisch belastete Plattform jemals „selbstheilend" oder „autonom" arbeiten kann, werden wir erleben. Ganz sicher jedoch hat die intensive Cloud-Erprobung im Rahmen der „Cloud First"-Strategie bei der Oracle-Datenbank 18c den produktiven Nutzern eine stabile und performante neue Version für „On-Premises" beschert.

Dies ist hervorzuheben, denn schnell rückt bei ungeplanten Vorfällen diese Performance in den Fokus der Aufmerksamkeit von Management und Geschäftsleitung. Zu Recht, sollten doch Leistung, Antwortzeit und Skalierbarkeit als Feature bereits bei der Planung und Umsetzung von Neubauten und Changes eine ebenso wichtige Rolle spielen wie fachliche Funktionen. Leider trifft der entstehende Druck dann schnell die falsche Zielgruppe: Die Tuner und Troubleshooter hatten oft wenig mit der nicht optimalen, initialen Umsetzung zu tun, sondern leisten den schwierigen Dienst einer Feuerwehr. Deshalb lege ich allen Kollegen ans Herz, sich an Planung und Auslegung schon der Neusysteme aktiv zu beteiligen und nach einem spontanen Tuning-Einsatz mindestens objektives Feedback an Entwickler, Architekten und deren beider Manager zu liefern.

Ich wünsche Ihnen nun viel Freude an der Lektüre des Red Stack Magazin zum Thema „Tuning" und viel Erfolg beim nächsten, hoffentlich proaktiven Optimierungseinsatz!

Ihr

Die Tuning-Tätigkeit zählt neben Installation und Patching zu den Kernaufgaben eines DBAs

Wenn Performance-Probleme auftreten, muss schnell und effizient gehandelt werden

System-Statistiken geben Auskunft über die CPU- und I/O-Leistung eines Systems

Einleitung

Tuning

Aktuell

Datenbank

Entwicklung

Intern

Timeline

29. Juni 2018

Die fünfte Auflage der DOAG 2018 Exa & Middleware Days in Frankfurt ist ein Erfolg auf ganzer Linie. Rund 70 Teilnehmer widmen sich an zwei aufeinanderfolgenden Tagen den Schwerpunkten „Exadata" und „Oracle Database Appliance"; hinzu kommt erstmals das Thema „Middleware". Die Beschäftigung mit der Cloud avanciert immer mehr zum übergreifenden Motiv in allen Themenbereichen und buchstäblich zum Selbstläufer, was die beiden Keynotes am ersten und zweiten Konferenztag eindrucksvoll zeigen. Sachin Pikle wirft in seinem Keynote-Vortrag über die „Open Source Fn"-Plattform ein Schlaglicht auf die Möglichkeiten, die sich Entwicklern mit Serverless Computing und der Autonomie von jeglicher Infrastruktur eröffnen. Der Master Product Manager Gurmeet Goindi greift diesen roten Faden am nächsten Tag auf, als er sich in seiner Keynote mit dem Potenzial der autonomen Oracle-Datenbank beschäftigt. Dem Motto „selbstverwaltend, selbstsichernd, selbstreparierend" verpflichtet, möchte Oracle laut Goindi einen Paradigmenwechsel durch die selbstverwaltende Datenbank durchführen und benutzerverursachte Fehler nahezu gänzlich eliminieren. Damit bestätigt Oracle erneut den Eindruck, sich ganz der Arbeitsentlastung der Datenbank-Administratoren verschrieben zu haben. Diese sollen sich mehr den Inhalten, der Sicherheit und der Analyse von Daten widmen. „Lernen aus Lösungsbeispielen" könnte das Zweitmotto der DOAG 2018 Exa & Middleware Days lauten: Noch stärker als in den vorangehenden Jahren liegt das Gewicht auf Projekt-Berichten und der Interaktion mit den Zuhörern. Der Einblick in ihr Portfolio geht häufig mit wertvollen Instruktionen einher, die die Teilnehmer direkt auf ihren Laptops umsetzen. Dabei liegt der Fokus neben der omnipräsenten Cloud in erster Linie auf Inbetriebnahme und Implementierung, doch auch Security und Patching erhalten gebührend Beachtung. Bei eher gemäßigten Temperaturen bleiben auch die Köpfe kühl genug, um sich sowohl den Vorträgen als auch dem Treffen der Arbeitsgruppe Engineered Systems zu widmen. Bereits die Treffen der letzten Jahre haben gezeigt, dass es nicht an Diskussionsbedarf mangelt. Unter der Leitung des Themenverantwortlichen Stefan Panek am ersten Konferenztag bildet das Treffen in diesem Jahr keine Ausnahme. Gurmeet Goindi steht den Teilnehmern Rede und Antwort und widmet sich den Erfahrungsberichten der Anwesenden. Dabei entzünden sich die Gemüter an diversen Themenkomplexen. Insbesondere der Ausfall des Oracle-Kundenservice führe zu Unzufriedenheit, so der Tenor. Nach wie vor gibt es hier reichlich Handlungsbedarf. Erschwert wird dies durch häufige Wechsel im Bereich des Oracle-Support-Managements. Stefan Panek weist die Anwesenden darauf hin, unbedingt an der DOAG-Support-Umfrage im Herbst teilzunehmen. Mit Begeisterung nehmen die Teilnehmer die Ankündigung auf, dass Oracle auf den Exadata-Maschinen in absehbarer Zeit auch das Betriebssystem Oracle Linux 7 verfügbar macht. Für einen längeren Zeitraum wird der parallele Einsatz möglich sein, um Anwendern die Umstellung auf die neue Oracle-Linux-7-Version zu erleichtern.

Reichlich Gesprächsbedarf auf den DOAG 2018 Exa & Middleware Days

3. Juli 2018

Laut dem DOAG Legal Council können Zweifel an der Wirksamkeit der Oracle-Vertragswerke hinsichtlich Lizenzierung in virtualisierten Umgebungen begründet sein. Mit dem VLAN-Approval hat Oracle zwar Entgegenkommen gezeigt, langfristig stellt dies für Nutzer von Virtualisierungstechnologien wie VMware jedoch keine Lösung dar. Die Einzellösung bietet weder Planungs- noch Rechtssicherheit. Die DOAG hat deshalb zwei Gutachten anfertigen lassen, deren Inhalte im Rahmen des DOAG Legal Council diskutiert werden. Das Ergebnis: Teile der auf amerikanischer Rechtstradition und Marktmacht beruhenden Vertragswerke könnten nach Einschätzung der Rechtsanwälte des DOAG Legal Council nach deutschem und europäischem Recht nicht klar genug geregelt und damit unwirksam sein. Die Oracle-Lizenzpolitik ist Gegenstand dauerhafter und stärker werdender Kritik bei DOAG-Mitgliedern und -Interessenten. Bemängelt wird vor allem die Intransparenz bei der Lizenzierung von Oracle-Produkten in virtualisierten Umgebungen, insbesondere im Zusammenspiel mit VMwares Software vSphere. Die Lage könnte sich bald noch einmal verschärfen, denn der Support für vSphere 5.5 endet am 19. September 2018. Dann sind die Nutzer gezwungen, auf vSphere 6.5 oder 6.7 umzusteigen – und sehen sich mit noch schärferen Lizenzbedingungen konfrontiert. Die DOAG hat in den letzten Jahren bereits mehrere Klärungsversuche mit dem Oracle-Management angestoßen. Zwar zeigt Oracle ein Entgegenkommen hinsichtlich der Thematik und bietet mit dem VLAN-Approval eine Einzellösung an. „Eine solche Einzellösung stellt für die DOAG jedoch keine Lösung des Problems dar. Jede Änderung der Architektur und jedes Upgrade bedeutet, dass mit Oracle neu verhandelt werden muss", so der ehemalige DOAG-Vorstandssitzende und Ehrenmitglied Dr. Dietmar Neugebauer. „Einzellösungen schaffen keine Rechtssicherheit." Die DOAG hat Prof. Dr. Michael Bartsch (Bartsch Rechtsanwälte) sowie Dr. Peter Hoppen (Streitz Hoppen & Partner) mit der Erstellung von Gutachten beauftragt. Beide kommen zu dem Ergebnis, dass die Wirksamkeit dieser auf amerikanischer Rechtstradition und Marktmacht beruhenden Vertragswerke zu weiten Teilen nach deutschem und europäischem Recht sehr zweifelhaft ist. Das DOAG Legal Council unterstützt diese Einschätzung: „Bei einer nachträglichen Vergütung bestehender Softwareverträge setzt das deutsche AGB-Recht der Wirksamkeit von Klauseln enge Grenzen. Vieles spricht dafür, dass diese Grenzen im Fall der Oracle-Verträge überschritten sind. Daher halte ich es für unwahrscheinlich, dass die Rechtsprechung, einschließlich des EuGH, Oracle im vorliegenden Fall eine zusätzliche Vergütung in Form von nachträglichen Lizenzgebühren zubilligen würde. Eine Intensivierung der Softwarenutzung durch die Virtualisierungstechnologie ist nicht erkennbar", so Dr. Thomas Thalhofer, Mitglied des DOAG Legal Council. Im Detail kommen die Gutachter zu folgenden Ergebnissen: Oracle hat die Anforderungen in den Vertragsdokumenten nicht schriftlich dokumentiert. Stattdessen habe der Hersteller seine Auffassung lediglich mündlich gegenüber den Oracle-Partnern kommuniziert. Wegen

Verstoßes gegen das Transparenzgebot fehle außerdem die Grundlage für die Preisbildung. Ein Verstoß gegen das Transparenzgebot liegt dann vor, wenn aus Sicht des durchschnittlichen Kunden eine textliche Unklarheit dazu führt, dass er den Inhalt des AGB-Textes nicht sicher nachvollziehen kann. Außerdem hegt Bartsch Zweifel, ob wegen gleicher Hardware-Nutzung der Kaufpreis erhöht werden kann. Er kommt zu dem Schluss, dass keine nachträglichen oder zusätzlichen Lizenzgebühren verlangt und auch keine Änderungen der Lizenzgebühren vorgenommen werden können. Diesbezügliche AGB-Klauseln seien unwirksam und gäben Änderungen nicht her. Die Rechtsanwälte des DOAG Legal Council sehen die rechtlichen Erfolgsaussichten für Oracle als gering an. Daher legen sie dem Hersteller nahe, eine Klarstellung zugunsten der Anwender zu veröffentlichen. Damit würde für die Nutzer von Virtualisierungstechnologien Rechtsklarheit geschaffen und die Wahrscheinlichkeit gesenkt, dass sich die Anwender nach alternativen Produkten umsehen.

4. Juli 2018

Am Vortag des Java Forum Stuttgart tagt die Mitgliederversammlung des Interessenverbunds der Java User Groups e.V. (iJUG). Inzwischen sind 35 User Groups aus Deutschland, Österreich und der Schweiz im iJUG organisiert. Die Vertreter der deutschsprachigen Java User Groups sprechen sich mit deutlicher Mehrheit dafür aus, dem Antrag der Eclipse Foundation Europe GmbH auf Fördermitgliedschaft im iJUG stattzugeben. Gleichzeitig wurde einstimmig beschlossen, Solutions Member in der Eclipse Foundation zu werden. Somit besitzt der iJUG nun Stimmrecht bezüglich der Zukunft von Jakarta EE, vormals bekannt als Java EE.

5. Juli 2018

In der Stuttgarter Liederhalle findet am 5. Juli 2018 das Java Forum Stuttgart statt. Die rund 2.000 Teilnehmer erhalten fast fünfzig Vorträge in sieben parallelen Streams; eine gute Möglichkeit, sich umfassend über Themen zu Java sowie im Java- und JVM-Umfeld zu informieren. Gleich zu Beginn verteilen Wolfgang Taschner, Chefredakteur der Java aktuell, Stefan Koospal, Vorstand des iJUG, und Jürgen Thierack von der Java User Group München die aktuelle Ausgabe der Java aktuell an die eintreffenden Besucher. Die ganze Palette mit den Zeitschriften ist im Nu verteilt. Das Konzept des Java Forum Stuttgart erweist sich als erfolgreich. Das Treffen ist als Low-Budget-Veranstaltung für alle Beteiligten ausgelegt. Die Einnahmen durch Sponsoren, Aussteller und Sponsored Talks werden vor allem zur Erzielung einer attraktiven Besuchergebühr eingesetzt. Die Organisation erfolgt durch die Java User Group Stuttgart, unterstützt durch viele ehrenamtliche Helfer.

Impression vom diesjährigen Java Forum Stuttgart

6. Juli 2018

Der DOAG-Vorstand beschließt auf einer Sitzung in Frankfurt mit den Stream- und Community-Leitern das Programm zur DOAG 2018 Konferenz + Ausstellung. Die ausgewählten Vorträge versprechen wieder eine interessante und spannende Veranstaltung.

11. Juli 2018

DOAG-Vorstand Robert Szilinski kommt nach Berlin, um als Moderator einen Workshop mit der DOAG-Geschäftsstelle zur IT-Strategie durchzuführen. Ziel ist es, die IT-Prozesse zu konsolidieren und die Abläufe zu optimieren.

31. Juli 2018

Das Programm der DOAG 2018 Konferenz + Ausstellung ist online und wartet darauf, ausgiebig studiert zu werden. Sage und schreibe 461 Sessions in acht Streams sorgen sowohl für thematische Vielfalt als auch für inhaltliche Tiefe. In diesem Jahr sind die Streams-Datenbank, Infrastruktur, Development, DevOps, Data Analytics, Strategie & Innovation, Middleware, Applications sowie zahlreiche Community-Sessions vertreten. Die drei Keynote-Speaker Axel Pols, Lars Vollmer und Jean-Philippe Hagmann runden das Angebot mit ihren hochkarätigen Vorträgen ab. Natürlich wartet auch der opulent gestaltete Community-Abend am Mittwoch mit zahlreichen Überraschungen auf: Mit dem Casino-Programm, der LED- und Cocktail-Show sowie dem vielfältigen Musikprogramm ist für Entspannung und Spaß gesorgt. Die reichhaltigen Buffets werden auch in diesem Jahr sowohl die Liebhaber zünftiger als auch die raffinierter Küche glücklich machen.

Die DOAG 2018 Konferenz + Ausstellung findet in diesem Jahr vom 20. bis 23. November 2018 in Nürnberg statt

13. August 2018

Fried Saacke, DOAG-Vorstand und Geschäftsführer, und Wolfgang Taschner, Chefredakteur des Red Stack Magazin, führen in Berlin mit Ralf Schmilewski, Leiter Anwendungsentwicklung ERP der myToys Group, ein Interview für das Red Stack Magazin. Im Mittelpunkt steht die Oracle E-Business Suite; myToys ist das Unternehmen mit den meisten Anwendern dieser Business-Lösung in Deutschland. Für Ralf Schmilewski ist es besonders wichtig, dass es bei Oracle eine kontinu-

ierliche technologische Weiterentwicklung gibt und dass Oracle dabei auch die Anwender einbezieht und viel mit ihnen kommuniziert.

16. August 2018

Die Mitarbeiterinnen und Mitarbeiter der DOAG-Geschäftsstelle erhalten einen Erste-Hilfe-Kurs, damit sie bei einem Notfall auf einer Veranstaltung richtig reagieren und mit anpacken können. Unter der Leitung von Jochen Gürtler, Ausbildungsleiter des Vereins Alpines Rettungswesen, werden viele Fälle sehr praxisnah geübt.

27. August 2018

Für das Konferenzprogramm der JavaLand 2019 gab es insgesamt 637 Einreichungen, die das Programmkomitee derzeit mit kritischem Expertenauge prüft. Das sind ein Viertel mehr Einreichungen als im vergangenen Jahr. Auch das Newcomer-Programm für Referenten ohne Bühnenerfahrung fand mit 60 Einreichungen reges Interesse; im Vorjahr waren es 23. Beim Schulungstag haben die Stream-Leiter dieses Mal die Qual der Wahl zwischen 37 Einreichungen von 23 Partnern – keine leichte Entscheidung, aber alle Teilnehmer können sich schon jetzt auf das fertige Konferenzprogramm freuen, das Ende September online veröffentlicht wird.

Die JavaLand 2019 verspricht wieder ein interessantes Programm für die Teilnehmer

31. August 2018

Fried Saacke, DOAG-Vorstand und Geschäftsführer, und Wolfgang Taschner, Chefredakteur der DOAG-Zeitschriften, machen sich Gedanken über die nahtlose Übergabe des Red Stack Magazin, der Java aktuell und der DOAG Business News an die Kommunikationsabteilung der DOAG, weil Wolfgang Taschner Anfang 2019 in den Ruhestand geht.

Dr. Dietmar Neugebauer
Ehemaliger DOAG-Vorstandsvorsitzender

Aus der Ferne betrachtet: Performance und Tuning – nur für Oracle Insider?

Beim Stöbern in meinen Unterlagen sind mir die Papiere eines Workshops mit dem Titel „Oracle Release 7.3 Performance und Tuning“ aus dem Jahre 1997 in die Hände gefallen, gehalten von den beiden DOAG-Botschaftern Dierk Lenz und Günter Unbescheid. Wenn ich jetzt die Ankündigungen aus dem Jahre 2018 lese, etwa die der Trivadis Performance Days und der Oracle Real World Performance Tour, oder auf eine Vielzahl von Performance- und Tuning-Vorträge auf der DOAG Konferenz schaue, stellt sich für mich schon die Frage, ob sich da in der Oracle-Datenbank-Software in den letzten zwanzig Jahren wirklich etwas verändert hat. Werfen immer neue, komplexere Features diese immer neuen Performance- und Tuning-Fragen auf? Ohne diesen Datenbank-Gurus zu nahe treten zu wollen, die sicherlich die Oracle-Software bis ins kleinste Detail kennen, aber sind nicht auch die meisten von ihnen schon die letzten fünfzehn bis zwanzig Jahre mit diesem Thema unterwegs?

Jetzt kommt die Oracle Autonomous Datenbank auf den Markt, inzwischen auch als „autonomous transaction processing database“ vorgestellt. Ist das ein Paradigmenwechsel in der Datenbank-Entwicklung? Noch dazu, wenn die Datenbank in der Cloud läuft und Oracle-Insider gar nicht mehr darauf zugreifen können?

Ich fände es nicht schlecht, wenn die Datenbank sich selbst optimiert, denn Performance und Tuning ist für mich wesentlich mehr als die Cache-, Lese- und Schreib-Optimierung der Datenbank-Prozesse. Die Optimierung einer Anwendung fängt bei der Eingabe des Anwenders an und hört bei der Ausgabe des Ergebnisses auf. Dazwischen gibt es so viele Schnittstellen, von denen die Datenbank, die von vielen nur als nicht-intelligenter Datenspeicher gesehen wird, nur eine ist.

Es geht darum, die technischen Anforderungen und Randbedingungen aller Schnittstellen zu verstehen und zu analysieren. Inzwischen können sehr viele Tools Hilfestellung geben, aber eines habe ich immer wieder erlebt: Bei diesen Schnittstellen gibt es auch eine nicht zu vernachlässigende menschliche Komponente. Doch wer möchte schon der Verursacher eines Performance-Problems sein? Allein wenn man hier nach dem sogenannten „Verursacher“ sucht, ist dies schon der falsche Ansatz. Bei der Analyse eines Problems muss es gelingen, das vorhandene Wissen aller Beteiligten Schritt für Schritt ohne Schuldzuweisung in ein meist als Krisenmanagement zusammengestelltes Team zu bringen. Hier geht es weniger um Bits und Bytes, sondern mehr um die Soft-Skill-Eigenschaften der Beteiligten.

Hier würde ich mir bei den Oracle-Insidern mehr Vermittlung ihrer langjährigen Erfahrung wünschen. Die DOAG selbst versucht ihren Konferenzbesuchern mit den Soft-Skill-Vorträgen zumindest Anregungen mitzugeben – auch für Oracle-Insider!

Ralf Schmilewski (rechts) im Gespräch mit Fried Saacke

„Schneller und verlässlicher Support ist in der Cloud unerlässlich …“

Neben technischen Lösungen wie der Datenbank hat Oracle auch Business Solutions im Angebot. Fried Saacke, DOAG-Vorstand und Geschäftsführer, und Wolfgang Taschner, Chefredakteur des Red Stack Magazin, sprachen darüber mit Ralf Schmilewski, Leiter Anwendungsentwicklung ERP der myToys Group.

Der Stammsitz von myToys in Berlin ist in einem beeindruckenden historischen Gebäude. Was hat es damit auf sich?

Ralf Schmilewski: Das Gebäude in der Potsdamer Straße 192 wurde in den Jahren 1938 und 1939 als Hauptsitz für die deutsche Milchwirtschaft erbaut und steht heute unter Denkmalschutz. In einem noch original ausgestatteten Raum wurde vor einigen Jahren sogar eine Szene mit Tom Hanks für den Film „Bridge of Spies“ gedreht.

In welchem Geschäftsfeld ist myToys tätig?

Ralf Schmilewski: myToys ist der führende Online-Shop für Spielzeug und Produkte rund ums Kind in Deutschland. Im Prinzip bietet myToys alles, was man für das Leben mit Kindern braucht, von Spielsachen über Babyartikel bis hin zur Kindermode. Doch das Angebot besteht längst nicht mehr nur aus Produkten für Kinder; über die Jahre hat sich myToys zu einer Unternehmensgruppe

entwickelt, zu der unter anderem auch die Shopping-Community limango und der Online-Schuh-Shop mirapodo gehören.

Was sind dabei die besonderen Herausforderungen?
Ralf Schmilewski: Ich bin jetzt seit mehr als achtzehn Jahren im Unternehmen und seitdem hat sich alles sehr verändert. Der Markt ist heute ständig im Umbruch und wir müssen schnell auf sich ändernde Bedingungen reagieren. Insbesondere beim Online-Shop dürfen wir uns keine großen Ausfallzeiten leisten.

Wie lösen Sie IT-technisch diese Aufgaben?
Ralf Schmilewski: Unsere IT erstreckt sich im Wesentlichen rund um den Warenhandel. Insbesondere im Bereich „ERP" arbeiten wir sehr viel mit Standard-Software, daneben kommt allerdings auch individuell entwickelte Software zum Einsatz. Im Mittelpunkt der Standard-Software steht die Oracle E-Business Suite, die wir bereits seit dem Jahr 2000 betreiben.

Aus welchen Gründen haben Sie sich für die Oracle E-Business Suite entschieden?
Ralf Schmilewski: Die Software bietet uns einen stabilen Rahmen, an dem wir uns orientieren können. Auf der anderen Seite ist die Oracle E-Business Suite flexibel genug, um sich an unsere speziellen Bedürfnisse anpassen zu können.

Die MyToys Group
Die myToys Group ist der Nummer-eins-Online-Händler für Family-Shopping in Europa. Zur mehrfach ausgezeichneten Markenfamilie, die im Geschäftsjahr 2016/17 einen Gesamtumsatz von mehr als 617 Millionen Euro erwirtschaftet hat, gehören die Shopping-Angebote von myToys, ambellis, mirapodo, yomonda und limango. Die MyToys Group zählt damit zu den erfolgreichsten E-Commerce-Unternehmen Deutschlands.

Mit Sitz in Berlin betreibt die Unternehmensgruppe unter der Marke myToys seit dem Jahr 1999 den Nummer-eins-Online-Shop für Spielzeug und Produkte rund ums Kind in Deutschland sowie gleichnamige Filialen. Die MyToys Group ist damit einer der führenden Multi-Channel-Anbieter für Kindersortimente im deutschsprachigen Raum.

Im Jahr 2010 wurde der Damenmode-Online-Shop ambellis gestartet. Seit 2013 gehören der Online-Schuhshop mirapodo sowie der Shopping-Club limango zur Markenfamilie. Im Jahr 2016 ging der „Home & Living"-Shop yomonda online. Durch das innovative Multi-Shop-Konzept haben Kundinnen seit 2013 die Möglichkeit, mit nur einem Kundenkonto und einem gemeinsamen Warenkorb in den Online-Shops der MyToys Group einzukaufen. Das Unternehmen beschäftigt derzeit mehr als 2.000 Mitarbeiter und belegt damit Platz drei der größten Arbeitgeber der Digitalbranche in Deutschland.

Anfang 2014 wurde die Logistik des Unternehmens in Gernsheim zentralisiert – mit mehr als 70.000 m² Lagerfläche einer der größten und modernsten Logistik-Standorte in Deutschland.

Wo sehen Sie die wesentlichen Stärken der Oracle E-Business Suite?
Ralf Schmilewski: Wir profitieren stark davon, dass wir die individuellen Anpassungen direkt in der Software vornehmen können und keine Erweiterungen drumherum erstellen müssen. Darüber hinaus bemerke ich eine große technologische Kompetenz bei Oracle in diesem Umfeld. Bei der Oracle E-Business Suite bekommen wir Technologie und Business aus einer Hand und haben damit auch nur einen einzigen Ansprechpartner.

In welchem Umfang setzen Sie die Oracle E-Business Suite ein?
Ralf Schmilewski: Wir haben momentan achthundert Named-User lizenziert und sind damit meines Wissens das Unternehmen mit den meisten E-Business-Suite-Anwendern in Deutschland. Aktuell arbeitet das ERP-Entwicklungsteam an etwa siebzig E-Business-Suite-Projekten pro Quartal. Bevor Änderungen jedoch in die EBS-Produktions-Instanz überführt werden können, müssen sie getestet werden. Dazu hatten wir bisher zwei Nicht-Produktionssysteme eingesetzt: ein Entwicklungs- und ein Testsystem. Dieses Jahr haben wir die DataOps-Plattform von Delphix eingeführt, mit der Klone der Produktions-Instanz in einem Bruchteil der üblichen Zeit und mit minimalen Speicheranforderungen bereitgestellt werden können. Die Anzahl der Nicht-Produktionsumgebungen lässt sich einfach skalieren, was parallele Entwicklungs-Streams möglich macht. Zudem bietet DataOps viele dynamische Daten-Kontrollfunktionen wie das Rücksetzen der Umgebung auf zuvor gesetzte Lesezeichen. Damit ist die Basis für ein Continuous Delivery geschaffen. Ich werde im Rahmen eines Vortrags auf der DOAG 2018 Konferenz über unsere Erfahrungen damit berichten.

Ist es für Sie ein Nachteil, dass Oracle mit der E-Business Suite in Deutschland nicht so sehr verbreitet ist?
Ralf Schmilewski: Eine größere Verbreitung würde für uns auch größere Netzwerke für den Erfahrungsaustausch bieten, was sicher

von Vorteil wäre. Wir versuchen zwar, uns auch über die DOAG international stärker zu engagieren, das ist allerdings nicht ganz so einfach.

Warum sind die Business-Lösungen von SAP nie bei Ihnen zum Einsatz gekommen?

Ralf Schmilewski: Diese Entscheidung ist vor rund zwanzig Jahren hier gefallen. Heute wäre ein Wechsel zu SAP im laufenden Betrieb gar nicht machbar und würde sich wirtschaftlich auch nicht rechnen. Deshalb stellt sich diese Frage für mich überhaupt nicht.

Oracle drängt seine Anwender sehr stark zu Cloud-Lösungen. Welche Rolle spielen diese in Ihrem Unternehmen?

Ralf Schmilewski: Die Bedeutung der Cloud wächst bei uns, wir betreiben dort auch schon erste Shop-Anwendungen. Auch im Bereich „ERP" strecken wir unsere Fühler in die Cloud aus. Ich bin gerade dabei, ein Projekt aufzusetzen, um unsere Sun-Spark-Infrastruktur durch eine X86-Architektur abzulösen. Im Zuge dessen soll auch unsere Entwicklungs- und Test-Infrastruktur in die Cloud migriert werden, um hier flexibler agieren zu können.

Hilft es Ihnen, wenn Oracle jetzt in Frankfurt ein Cloud-Rechenzentrum betreibt?

Ralf Schmilewski: Das ist für uns interessant und wir reden auch bereits mit Oracle darüber. Es ist vor allem dann relevant, wenn es über die Entwicklungs- und Test-Umgebung hinausgeht.

Sind die Oracle Engineered Systems für Sie ein Thema?

Ralf Schmilewski: Im Data-Warehouse-Umfeld ist bei uns eine Exadata im Einsatz, für den ERP-Bereich sind diese Lösungen weniger interessant.

Wie sind Ihre Erfahrungen mit Oracle Support?

Ralf Schmilewski: Wir arbeiten schon sehr lange mit dem Support zusammen und haben bereits einige Hürden erfolgreich gemeistert. Derzeit läuft ein Projekt, um auf Grundlage der bisherigen Erfahrungen Optimierungspotenziale zu erarbeiten. Dies soll vor allem Oracle dazu dienen, sich auf die Anforderungen der „neuen Zeit" einzustellen und die Anwender bei den ganzen Veränderungen hinsichtlich Cloud auch optimal unterstützen zu können. Je mehr Anwendungen in der Cloud laufen, desto dringlicher ist es, dass Oracle schnellen und verlässlichen Support gewährleistet.

Wie beurteilen Sie die Zukauf-Strategie von Oracle?

Ralf Schmilewski: Dazu kann ich eine kleine Anekdote erzählen: Wir hatten hier schon immer Sun-Hardware im Einsatz. Nach der Übernahme von Sun durch Oracle kamen dann die gleichen Leute, mit denen wir zuvor bei Sun im Gespräch waren, als Oracle-Mitarbeiter auf uns zu. Allein durch den Zukauf von Sun sind wir damit ein noch größerer Oracle-Premium-Kunde geworden. Auch den Maximizer hatten wir schon im Einsatz, bevor er zu Oracle gehörte.

Können Sie sich vorstellen, den kompletten Stack eines Herstellers wie Oracle einzusetzen?

Ralf Schmilewski: Wir begrüßen es schon, wenn wir beispielsweise nur einen einzigen Ansprechpartner für die Hardware und die Datenbank haben. Andererseits ist es natürlich für uns wichtig, die besten Produkte in einer Kategorie einsetzen zu können. Von daher haben wir immer den gesamten Markt im Blick. Wen man eine vernünftige IT macht, achtet man schon darauf, sich nicht von einem einzigen Hersteller abhängig zu machen.

Welche Wünsche haben Sie an Oracle?

Ralf Schmilewski: Ich würde mir wünschen, dass es bei Oracle eine kontinuierliche technologische Weiterentwicklung gibt und dass Oracle dabei auch die Anwender mit einbezieht und mit ihnen viel kommuniziert.

Wie sehen Sie den Stellenwert einer Anwendergruppe wie der DOAG?

Ralf Schmilewski: Die DOAG ist für mich ein Katalysator zwischen den Anwendern und Oracle. Der Verein fordert Dinge von Oracle ein und gibt uns Feedback zurück. Darüber hinaus schätze ich die Veranstaltungen der DOAG sehr. Ich habe dort schon viel von den Erfahrungen anderer Anwender profitiert und bin immer gerne bereit, beispielsweise auf der Jahreskonferenz, auch meine gemachten Erfahrungen weiterzugeben.

Zur Person: **Ralf Schmilewski**

Ralf Schmilewski ist Leiter der Anwendungsentwicklung ERP bei myToys und seit dem Jahr 2000 im Unternehmen tätig. Er hat die Entwicklung der MyToys Group GROUP vom Startup zum führenden Online-Händler für Family-Shopping in Europa hautnah miterlebt und mitgestaltet. In dieser Zeit hat er immer wieder Erfahrungen mit diversen innovativen Oracle-Technologien gesammelt, angefangen bei der Infrastruktur bis hin zur Anwendung. Es ist ihm schon immer eine Herzensangelegenheit gewesen, seine Mannschaft agil zu organisieren. Er ist ständig mit seinem Team auf der Suche nach Möglichkeiten, die eigene Arbeit weiter zu optimieren.

ORACLE Platinum Partner

LOGICALIS
Business and technology working as one

Datenbank goes Cloud

- Lizenzoptimierung für die Cloud
- Oracle Public / Hybrid Cloud Integration
- Oracle Cloud at Customer Lösungen
- Managed Services für Private Cloud Lösungen

Treffen Sie uns auf der DOAG Konferenz + Ausstellung

3.Stock, Stand 322
Gegenüber Oracle

Tuning der Oracle-Datenbank – Entwicklungen und Trends

Tobias Deml, ORACLE Deutschland B.V. & Co. KG

Die Tuning-Tätigkeit zählt neben Installation und Patching zu den Kernaufgaben eines Oracle-Datenbank-Administrators. Die Ausprägung des Tätigkeitsbereichs hat sich über die Jahre entwickelt und verändert. Ziel dieses Artikels ist es, zunächst auf die Entwicklung der Oracle-Datenbank einzugehen und anschließend aktuelle Trends aufzuzeigen.

Über die Zeit hinweg wurden der Oracle-Datenbank viele Funktionalitäten zur Performance-Optimierung hinzugefügt. Einige dieser Features verbessern die Analyse-Möglichkeiten, während andere dem Datenbank-Administrator lästige Aufgaben abnehmen können. In heutiger Zeit sind die meisten dieser Funktionalitäten gar nicht mehr wegzudenken. Nachfolgend ein Überblick über die historische Entwicklung der Oracle-Datenbank anhand ausgewählter Tuning-Funktionalitäten *(siehe Abbildung 1)*.

Oracle Database 9i

Im Jahre 2001 wurde die Oracle-Datenbank 9i veröffentlicht. In dieser Version wurden essenzielle Funktionalitäten zur Performance Optimierung implementiert, die bis heute einen wichtigen Bestandteil der Oracle-Datenbank darstellen. Zu diesen Features zählen unter anderem:

- Automatic Query Rewrite
- Automatic Undo Management

Die damalige Einführung von Automatic Query Rewrite ermöglicht es, Zugriffspfade für die Ausführung eines SQL-Statements zu analysieren und zu verändern. Beispielsweise konnte dadurch eine Query, die mehrere Tabellen abfragt, auf eine Materialized View umgeleitet und

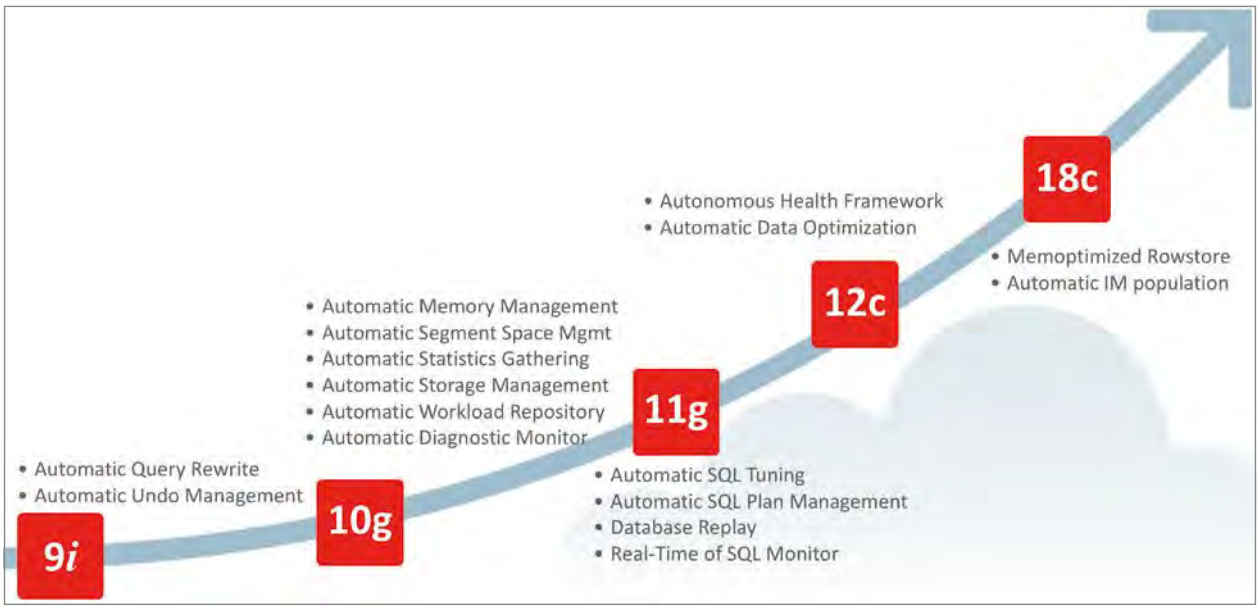


Abbildung 1: Historische Entwicklung der Tuning-Funktionalitäten

dadurch die Abfragegeschwindigkeit eklatant gesteigert werden. Diese Funktionalität wurde in den nachfolgenden Versionen erweitert und verbessert.

Vor Einführung des Automatic Undo Management mit der Oracle-Datenbank 9i war die manuelle Verwaltung der Undo-Daten durch den Datenbank-Administrator notwendig. Dabei mussten verschiedene Initialisierungsparameter bedarfsgerecht gesetzt und deren Einstellungen regelmäßig geprüft werden. Durch das Automatic Undo Management ist hingegen lediglich die Bestimmung des Umfangs des Platzes für Rollback-Segmente und deren Verwahrzeit durch den Datenbank-Administrator erforderlich. Alles Weitere bewerkstelligt die Datenbank automatisch. Die Organisation der Rollback-Segmente ist besonders für die Insert-Performance einer Oracle-Datenbank von großer Bedeutung.

Oracle Database 10g

Mit der Datenbank-Version 10g wurde eine Vielzahl neuer Funktionalitäten eingeführt, die mit der Datenbank-Performance in enger Verbindung stehen.

Die Optimierungen können in drei Teilbereiche zusammengefasst werden: Verbesserungen der Datenablage, der Sammlung und Auswertung von Statistiken sowie Vereinfachung der Speicherverwaltung. Im Detail wurden folgende Funktionalitäten eingeführt:

- Automatic Segment Management
- Automatic Storage Management
- Automatic Statistics Gathering
- Automatic Workload Repository
- Automatic Database Diagnostic Monitor
- Automatic Memory Management

Vor der Einführung des Automatic Segment Management war die Verwaltung der Daten innerhalb eines Segments anhand genauer Definition sogenannter „Storage-Parameter" notwendig. Die bedarfsgerechte Bestimmung der Parameter stellte dabei eine sehr zeitintensive Aufgabe dar. Mit der Version 10g gab es die Möglichkeit, dies von der Datenbank selbst bestimmen zulassen und somit eklatant Zeit einzusparen.

Eine weitere Neuerung der Version 10g, die auf die Optimierung der Datenhaltung innerhalb der Datenbank abzielt, war das Automatic Storage Management (ASM). Die Verwaltung von Filesystemen kann besonders bei großen Datenbanken unübersichtlich werden. Außerdem können Filesysteme, die über das Betriebssystem als Mountpoint angeschlossen sind, sich ebenfalls zu einem potenziellen Problemherd für Performance-Engpässe einer Oracle-Datenbank entwickeln.

Mit Einführung von ASM wurden viele der genannten Probleme adressiert und gelöst. So werden RAW-Devices ohne jegliches Filesystem für die Ablage der Datafiles der Datenbank verwendet. Diese Anbindung eliminiert viele Schichten, die für eine unzufriedenstellende Datenbank-Performance verantwortlich sein können. Außerdem hat der Einsatz von ASM noch einen administrativen Vorteil, denn diese Devices können lediglich durch die Datenbank verwendet werden und sind somit von Eingriffen aus dem Betriebssystem geschützt.

Die Neuerungen Automatic Statistics Gathering, Automatic Workload Repository und Automatic Diagnostic Monitor befassen sich generell mit Verbesserungen der Erhebung und Auswertung von Performance-relevanten Metadaten. Durch die Einführung dieser Funktionalitäten bietet die Oracle-Datenbank eine automatische Analyse der Objekte in der Datenbank. Diese statistischen Daten sind besonders für den Optimizer von Bedeutung, um fundierte Entscheidungen hinsichtlich effizienter Ausführungspläne zu treffen. Diese und weitere Informationen sind im Automatic Workload Repository (AWR) abgelegt, wo sie per SQL-Abfragen oder durch vordefinierte Berichte ausgewertet werden können. Außerdem wurde mit dieser Version noch der Automatic Database Diagnostic Monitor (ADDM) eingeführt. Er gibt anhand von statistischen Werten Empfehlungen, um die Performance der Datenbank zu verbessern.

Neben den Verbesserungen im Umfeld der Datenablage und der Handha-

bung von Datenbank-Statistiken wurde mit der Version 10g das Automatic Memory Management eingeführt, um die Verwaltung des Hauptspeichers, der der Datenbank-Instanz zugeordnet wurde, zu vereinfachen. Vor der Einführung dieses Features mussten anhand einer Vielzahl von Parametern die verschiedenen Memory-Bereiche und Pools der Instanz festgelegt werden. Diese definierten Werte waren statisch und konnten bei Änderung des Lastprofils eine nicht optimale Datenbank-Performance hervorrufen. Diese Problematiken wurden mit der Einführung von Automatic Memory Management adressiert und gelöst. Ab diesem Zeitpunkt konnte man mit den Parametern „SGA_TARGET“ und „PGA_AGGREGATE_TARGET“ lediglich grobe Bereiche festlegen. Diese wurden dann von der Datenbank automatisch und Last-abhängig in die notwenigen Pools unterteilt. Die Verteilung dieser Pools kann die Oracle-Datenbank je nach Situation anpassen, um gegebenenfalls auf veränderte Last-Situationen zu reagieren.

Oracle Database 11g

Im Jahre 2007 wurde mit der Version 11g erneut eine Reihe von Verbesserungen im Umfeld der Performance-Analyse und des Tunings veröffentlicht. Diese Neuerungen konzentrieren sich hierbei im Speziellen auf den Bereich der Auswertung des Workloads und einzelner SQL-Abfragen. Im Detail wurden mit der Version 11g der Oracle-Datenbank folgende Features im Bereich des Performance-Tunings eingeführt:

- Automatic SQL Tuning Advisor
- Automatic SQL Plan Management
- Database Replay
- Real-Time SQL Monitor

Der Automatic SQL Tuning Advisor analysiert ausgeführte SQL-Statements und deren ermittelte Ausführungspläne. Anschließend werden die Ausführungsschritte auf Optimierungspotenzial untersucht. Falls Möglichkeiten zur Optimierung dieses SQL gefunden wurden, schlägt der Automatic SQL Tuning Advisor sie dem Benutzer vor. Diese Optimierungen reichen vom manuellen Erstellen von Statistiken über die Erstellung von Materialized Views bis hin zur Empfehlung der Anlage eines Index.

Mit Einführung von Automatic SQL Plan Management hat der Benutzer die Möglichkeit, über sogenannte „SQL Plan Baselines“ explizit Einfluss auf den Ausführungsplan einer Abfrage zu nehmen. Dies nimmt dem Query Optimizer die Flexibilität, beispielsweise auf Segment-Wachstum zu reagieren, im Problemfall kann sie jedoch von großem Nutzen sein.

Das Automatic Workload Capture/Replay wird ebenfalls unter dem Feature-Namen „Oracle Database Replay“ geführt. Hierbei handelt es sich um die Möglichkeit, den Datenbank-Workload inklusive aller ausgeführten Statements aufzuzeichnen. Anschließend kann diese Aufzeichnung auf einer zweiten Datenbank abgespielt werden, um gegebenenfalls Unterschiede in Performance oder Ergebnissen ausfindig zu machen.

Mit der Oracle Datenbank 11g wurde auch der SQL Monitor verbessert. Ab dieser Version ist es damit möglich, Queries zu analysieren, auch wenn diese noch nicht abgeschlossen wurden. Somit können SQL-Lang-Läufer noch während ihrer Ausführungszeit beurteilt werden und somit schneller Rückschlüsse getroffen werden.

Oracle Database 12c

Im Juli 2013 wurde schließlich das neue Major Release 12c der Datenbank veröffentlicht. Neben der Einführung der neuen Multitenant-Architektur kam eine Vielzahl neuer Funktionalitäten heraus, darunter ebenfalls Features, die in den Bereich des Performance-Tunings fallen. Diese Neuerungen waren:

- Autonomous Health Framework
- Automatic Data Optimization

Das Autonomous Health Framework befasst sich mit der Sammlung und Analyse verschiedenster Protokoll-Informationen aus dem Umfeld des Real Application Cluster (RAC). Ziel ist es, den Zustand des Clusters zu überwachen und proaktiv über gesammelte Informationen sich anbahnende Probleme zu erkennen. Alle die analysierten Informationen werden zentral im Grid Infrastructure Management Repository (GIMR) gesammelt und für historische Auswertungen vorgehalten.

Mit steigenden Datenvolumina wächst auch die Nachfrage nach intelligentem Datenmanagement. Ein solches Management überwacht die Nutzung der Daten, um sie auf dem optimalen Storage-Typus zu platzieren.

Bei hoher Nutzung (heiß) geht es auf den schnellen Storage, bei niedriger Nutzung (kalt) auf den günstigen. Diese intelligente Ablage spart Kosten, ohne Kompromisse bei den Abfragezeiten der heißen Daten einzugehen. Diese Funktionalität wurde mit der 12c-Neuerung Automatic Data Optimization direkt in die Oracle-Datenbank eingebettet.

Oracle Database 18c

Mit der Datenbank-Version 18c wurden weitere Verbesserungen hinsichtlich Performance-Tuning eingeführt. Neben einer Vielzahl kleinerer Verbesserungen wurden ebenfalls Neuerungen im Bereich der Datenverwaltung im System-Memory veröffentlicht. Auf folgende neue Features der Version 18c wird genauer eingegangen:

- Memoptimized Rowstore
- Automatic In-Memory Population

Der Memoptimized Rowstore ist ein neuer Pool der System Global Area (SGA), der zur Optimierung von Zugriffszeiten auf Tabellen gedacht ist. Daten können in Granularität einer Tabelle in diesen Bereich geladen werden. Diese befindlichen Daten werden ebenfalls mit einem Hash-Index versehen, um die Zugriffe zusätzlich zu beschleunigen. Die Verwaltung und Ablage der Daten ist für einen Benutzer, der über einen Client angebunden ist, vollkommen transparent. Somit bedarf

die Einführung dieser Funktionalität keinerlei Änderung von Applikationen.

Im Juli 2014 wurde mit der Version 12.1.0.2 die Funktionalität Oracle In-Memory Database eingeführt. Damit konnten analytische Abfragen mithilfe des Column Store enorm verbessert werden. Bis einschließlich Version 12.2.0.1 mussten die Daten, die in diesen Store geladen werden sollten, manuell bestimmt werden. In der Version 18c wurde die Automatic In-Memory Population implementiert, die diese Tätigkeit komplett abnimmt. Durch die Kombination der In-Memory Database und der Automatic-Data-Optimization-Funktionalität kann diese Möglichkeit realisiert werden. Damit werden die Zugriffe der einzelnen Daten protokolliert, um die Daten in heiße und kalte Bereich zu unterteilen. Die stark nachgefragten Bereiche (heiß) werden anschließend in den In-Memory Column Store geladen, bis dessen Kapazität erreicht wurde.

Trends und aktuelle Entwicklungen

In den letzten Monaten hat die neue Produktlinie Oracle Autonomous Database Bekanntheit erlangt. Dabei handelt es sich um Datenbank-Services in der Oracle-Cloud, die mit einem hohen Automatisierungsgrad ausgestattet sind. Diese Services sind, zum Zeitpunkt der Verfassung dieses Artikels, in zwei verschiedenen Varianten verfügbar:

- Oracle Autonomous Data Warehouse (ADW)
- Oracle Autonomous Transaction Processing (ATP)

Beiden Services unterscheiden sich in ihrer Anpassung an verschiedene Lastprofile, dabei ist ADW auf Data Warehouse und ATP auf OLTP-Last ausgerichtet. Mit Veröffentlichung der Autonomous Database verfolgt Oracle ein Ziel, das sie sich bereits seit Jahrzenten zur Aufgabe gemacht hat, nämlich den Benutzer der Datenbank von Tätigkeiten zu entlasten, wodurch er die Möglichkeit hat, sich Themen zu widmen, die in Zukunft mehr an Bedeutung gewinnen werden. Besonders das Thema „Security" hat mit dem Inkrafttreten der EU-Datenschutz-Grundverordnung in letzter Zeit deutlich an Bedeutung gewonnen.

Diese Services dienen nicht als pauschales Ziel für alle heutigen Oracle-Datenbanken, vielmehr kann man es als neue Möglichkeit betrachten, Anforderungen an Datenhaltungssysteme einfacher abzudecken. Diese Dienste bieten die Stabilität und Innovationskraft einer Oracle-Datenbank, ohne Kompromisse – wie Provisionierung-Aufwände oder Limitierungen bei der Skalierung – in Kauf zu nehmen.

Die Historie der Oracle-Datenbank hat gezeigt, dass über die Zeit viele Automatisierungen eingeführt wurden, um Datenbank-Administratoren zu entlasten. Diese Richtung wird mit der Oracle Autonomous Database weiterverfolgt und weiterentwickelt.

Weitere Informationen und Links

- Deutschsprachiger Oracle-Blog für die Database 18c: *https://blogs.oracle.com/coretec/datenbank-18c*
- Dokumentation Oracle Database 18c: *https://docs.oracle.com/en/database/oracle/oracle-database/18/index.html*
- Dokumentation Oracle Autonomous Data Warehouse: *https://docs.oracle.com/en/cloud/paas/autonomous-data-warehouse-cloud/*
- Dokumentation Oracle Autonomous Transaction Processing: *https://docs.oracle.com/en/cloud/paas/atp-cloud/index.html*

Tobias Deml
tobias.deml@oracle.com

Oracle-Performance-Analyse mit frei verfügbaren Tools

Uwe Küchler, OPITZ CONSULTING Deutschland GmbH

Nicht immer stehen einem Datenbank-Administrator GUI-basierte und oftmals sehr teure Tools zur Verfügung, um ein Performance-Problem in der Datenbank zu analysieren. Doch auch mit den vorhandenen Bordmitteln der Datenbank und kostenfrei verfügbaren Tools lässt sich eine gute Diagnose stellen. Dieser Artikel stellt eine Auswahl solcher Werkzeuge vor und legt einen Schwerpunkt auf die Möglichkeiten, die in der Oracle Standard Edition ohne Diagnostics Pack zur Verfügung stehen.

Die Betrachtung der hier vorgestellten Werkzeuge gliedert sich in zwei Anwendungsbereiche:

- Analyse akuter Performance-Probleme
  - Zugriff nur per SSH möglich?
  - Zugriff nur per SQL möglich?
- Historische Analyse ohne AWR und ASH

Wer über die Oracle Enterprise Edition mit Diagnostics Pack und Enterprise Manager verfügt, ist für beide Aufgabenstellungen gut versorgt. Eine grafische Darstellung der Lastdaten, über die ein Drill-Down zu den Verbrauchern von System-Ressourcen gemacht werden kann, erleichtert dem DBA die Arbeit. Was aber, wenn kein Diagnostics Pack lizenziert ist, beispielsweise in der Standard Edition? Dann kommt auch der Einsatz des Enterprise Manager nicht infrage. Es müssen also Alternativen her – möglichst, ohne erst manuell Abfragen unter SQL*Plus absetzen zu müssen. Im Rahmen dieses Artikels kann nur eine sehr kleine Auswahl an verfügbaren Tools vorgestellt werden. Die Wahl fiel auf Tools, die keine Installation von Drittherstellern benötigen, stellt jedoch ansonsten keine Wertung dar.

Kein (X-)Windows? Kein Remote Desktop? Kein Problem!

Aus Sicherheitsgründen sind Datenbank-Server häufig nur per SSH erreichbar. Es kann also nur im Textmodus gearbeitet und keine grafische Oberfläche einge-

setzt werden. Entwicklern oder externen Beratern wird oft sogar der Shell-Zugriff zum Server verwehrt. Dann bleiben nur noch SQL-Abfragen, um sich Informationen über den Zustand der Datenbank zu besorgen. Zunächst der etwas weniger restriktive Fall, bei dem ein SSH-Zugriff erlaubt und bei dem auch eigene Software verwendet werden kann.

AMON – top für die Oracle-Datenbank

AMON wurde von Andrej Simon, einem Mitarbeiter des Oracle-Support-Teams, entwickelt und wird immer noch aktuell gehalten; es ist auf Oracle 10.1 bis 12.2 sowie auf allen Unix- und Linux-Varianten lauffähig. Wer in der Regel über SSH (etwa mittels PuTTY) auf Datenbank-Servern arbeitet, bekommt mit AMON ein Werkzeug an die Hand, das – ähnlich wie das Unix-Tool „top“ auf OS-Ebene – die Last einer Datenbank-Instanz darstellt.

AMON macht nichts anderes, als in regelmäßigen, konfigurierbaren Abständen Performance-Metriken aus dem Data Dictionary der Datenbank auszulesen und die Differenzen zwischen den Intervallen sortiert darzustellen. Im Unterschied zu „top“ sind die Möglichkeiten des Drill-Down allerdings deutlich höher. Ferner zeigt AMON auch diverse Informationen zur Konfiguration der Datenbank an.

AMON muss nicht installiert werden; ein Entpacken des Archivs reicht aus. Es benötigt dann nur noch folgende Umgebungsvariablen:

- ORACLE_HOME
- ORACLE_SID
- LD_LIBRARY_PATH

AMON kann sich damit nach Aufruf sofort an der Datenbank anmelden, was üblicherweise als „/ as sysdba“ erfolgt, allerdings konfigurierbar ist. Damit lässt es sich gut mit personalisierten Accounts oder in Multitenant-Umgebungen einsetzen. Zur Erleichterung des Aufrufs lässt sich mit „alias amon='~/scripts/zk/amon/11.2/amon64_ol5_11r2 -u system -p oracle'“ ein Alias einrichten.

Die folgenden Screenshots zeigen einen typischen Workflow bei der Diagnose eines Performance-Problems. Nehmen wir einmal an, es würde eine auffällig

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
Move page forward(f), page back(b)
== General database information. ==

Database character set: AL32UTF8        Force full DB cache: NO
Database log mode     : NOARCHIVELOG    Database role : PRIMARY
Force logging         : NO              Flashback mode: NO
Online redo logs: 3 x 50.0MB
Datafiles: 3.16GB  Tempfiles: 197.0MB

General database initialization parameters.
  cpu_count: 2  db_block_size: 8192  processes: 300

Events and hidden database initialization parameters (V$SYSTEM_PARAMETER).
  Name                    Value
  _ash_size               24M
  _catalog_foreign_restore FALSE

Database components (DBA_REGISTRY).
  Component                          Version     Status
  Oracle Real Application Clusters   12.1.0.2.0  OPTION OFF
  JServer JAVA Virtual Machine       12.1.0.2.0  VALID
  OLAP Analytic Workspace            12.1.0.2.0  VALID
  Oracle Application Express         5.0.3.00.03 VALID
  Oracle Database Catalog Views      12.1.0.2.0  VALID
```

Abbildung 1: Startbildschirm von AMON

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
Order by time(1), instance ID(2)
== System time model statistics (V$SYS_TIME_MODEL). ==

Avg. DB time(sec.) per sec. (1, 5, 15 min. ago): 0.24, 0.10, 0.34
Avg. DB wait time ratio (1, 5, 15 min. ago): 8.1%, 15.7%, 14.6%
Avg. DB CPU time ratio (1, 5, 15 min. ago): 91.9%, 84.3%, 85.4%

Time    Statistic
(Sec.)  Name
10.03   DB time
10.03   sql execute elapsed time
9.63    DB CPU
2.69    PL/SQL execution elapsed time
0.75    parse time elapsed
0.40    hard parse elapsed time
0.11    repeated bind elapsed time
```

Abbildung 2: Time Model Statistics

```
$ export LD_LIBRARY_PATH=$ORACLE_HOME/lib:/lib:/usr/lib
$ amon
```

Listing 1

hohe CPU-Last auf dem Server beobachtet und wir möchten feststellen, ob das an der Datenbank liegt und falls ja, woran genau. Starten wir zunächst AMON *(siehe Listing1 und Abbildung 1)*.

AMON empfängt einen mit allgemeinen Informationen zur Datenbank, Hinweisen zu außergewöhnlichen Parametern und gesetzten Events sowie den installierten Komponenten. Das Programm wird über Tastenbefehle gesteuert. Diese können über die Hilfeseiten mit „h“ (alphabetisch) und „hh“ (gruppiert) aufgerufen werden. Dort findet man die Gruppe „Time Model Statistics“ und ruft die erste Seite mit „t“ auf *(siehe Abbildung 2)*.

Nach dem ersten Intervall von zehn Sekunden erscheinen die ersten Werte. Die „Avg CPU Time Ratio“ von 91,9 Prozent zeigt, dass die Instanz vor allem CPU-lastig läuft. Der untere Abschnitt legt mit zehn Sekunden Datenbank-Time in einem Zehn-Sekunden-Intervall nahe, dass hier durchschnittlich eine logische CPU voll ausgelastet läuft. Diese Last wird überwiegend durch SQL-Ausführungen verursacht. Parsing von SQL, das auch sehr CPU-intensiv werden kann, spielt hier mit 0,75 Sekunden nur eine untergeordnete Rolle. Es gilt nun herauszufinden, welche Sessions die momentane Last verursachen. Dazu klickt man ein zweites Mal auf „t“, um die Sessions, sor-

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
Press z(zoom) to get a session details snapshot
== Sess. time model stat. (V$SESS_TIME_MODEL). ==

Avg. DB time(sec.) per sec. (1, 5, 15 min. ago): 1.01, 0.27, 0.41
Avg. DB wait time ratio (1, 5, 15 min. ago): 6.4%, 11.6%, 12.2%
Avg. DB CPU time ratio (1, 5, 15 min. ago): 93.6%, 88.4%, 87.8%

SID,     Con SPID  User   S Program          DB Time DB CPU  BG Time
Ser#     ID                                  (sec.)  (sec.)  (sec.)
40,20750 3   10443 SYSTEM A oracle@vbgeneric 4.00    3.90    0.00
288,7380 3   10465 SYSTEM A oracle@vbgeneric 0.11    0.09    0.00
51,5425  3   6105  SYS    I oracle@vbgeneric 0.00    0.00    0.00
29,24658 3   5798  SYS    I oracle@vbgeneric 0.00    0.00    0.00
```

Abbildung 3: Sessions, sortiert nach Datenbank-Time

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
View session information(1), SQL statement(2), SQL plan(3)
                                    ew. ==
                                Page: 1/3
SID: 43   Ser: 17530  SPID: 9976  Status: ACTIVE  Cur. Time: 13:04:06

Logon time  : 08-apr-18 12:59:58
Session type: USER  Cont-Name: ORCL (Cont-ID: 3)
DB user name: SYSTEM
OS user name: oracle
SOSID       : 9976  (SPID: 9976  STID: 9976)
Program     : oracle@vbgeneric
Machine     : vbgeneric
Service name: orcl
Client Info :
Module: SQL*Plus
Action:

Event: latch: shared pool
P1,P2,P3: 1611722208, 453, 0
Seconds in wait: 0  Wait time: -1  State: WAITED SHORT TIME
Wait class: Concurrency
```

Abbildung 4: Session-Details

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
View session information(1), SQL statement(2), SQL plan(3)
== Current SQL statement (V$SQL). - Dynamic view. ==

SID: 40   Ser: 20750  SPID: 10443  Status: ACTIVE

SQL-ID: 82hxvr8kxuzjq  Child number: 0  Plan hash value: 0
SQL Text:
BEGIN dbms_stats.gather_database_stats; END;

Captured bind variables for this SQL cursor (V$SQL_BIND_CAPTURE):

--- No bind data captured.
```

Abbildung 5: Session SQL

tiert nach Datenbank-Time, aufzulisten *(siehe Abbildung 3)*.

In diesem einfachen Beispiel ist genau eine Session mit auffällig hoher DB-Time aufgelistet (schwarz hinterlegt). Für Details zu einer Session kann man in der Liste mit „j“ und „k“ navigieren und dann mit „z“ die Detailseiten aufrufen *(siehe Abbildung 4)*.

Auf dieser Seite stehen zunächst allgemeine Informationen über die Session sowie das zum Zeitpunkt des Snapshots aktive Wait-Event. Die Informationen füllen mehr als eine Bildschirmseite und lassen sich mit „f“ (forward) und „b“ (back) durchblättern. Zur Orientierung lohnt sich ein Blick in die Kopfzeile (schwarz hinterlegt), in der zyklisch Tastenbefehle angezeigt sind, mit denen man von hier aus weitere Details anzeigen kann. In diesem Fall ist es die Taste „2“, die Details zum aktuell ausgeführten SQL in dieser Session liefert *(siehe Abbildung 5)*.

Als Beispiel-Szenario für diesen Artikel wurde in einer separaten SQL*Plus-Session die Berechnung von Datenbank-Statistiken gestartet, um Last zu erzeugen. Damit haben wir an diesem Punkt identifiziert, wer und was die Last auf unserem Server erzeugt. In anderen Fällen könnte man nun noch tiefer forschen und sich beispielsweise den Ausführungsplan für das SQL ansehen (Taste „3“) oder gleich alle Detailinformationen in eine Textdatei schreiben lassen (Taste „0“), um diese dann in Ruhe zu untersuchen oder weiterzuleiten.

Extended SQL Trace und orasrp

Diese Werkzeuge bewegen sich an der Grenze zwischen „nur SSH-Zugriff“ und „nur SQL-Zugriff“. Ein sogenanntes „Extended SQL Trace“ für ein Statement, eine Session oder auch die ganze Datenbank (Vorsicht: gewaltige Datenmengen) können direkt aus SQL heraus ausgelöst werden. Die Herausforderung besteht allerdings darin, die auf dem Datenbank-Server erzeugten Trace-Files auszuwerten oder auf den eigenen Client zu übertragen. Auch der von Oracle mitgelieferte Trace File Profiler „tkprof“ ist oft nur auf dem Server verfügbar. Er ist zwar Teil der Vollinstallation des Oracle-Clients, doch unter Anwendern ist diese immer weniger verbreitet.

Auch das hier vorgestellte „orasrp“ kann Client-seitig laufen (Linux, MacOS, Windows). Es gibt jedoch auch einen interessanten Anwendungsfall für einen Server-seitigen Betrieb: Ein besonderes Feature ist nämlich eine Proxy-Komponente, über die das Tool dann vom Client ferngesteuert werden kann. Wer den Performance-Tuning-Kurs von Oracle besucht hat, wird mit Extended SQL Trace bereits vertraut sein. Für alle anderen hier das Wichtigste dazu in Kürze:

- Wenn die im Data Dictionary vorhandenen Informationen nicht ausreichen, um das Lastverhalten bestimmter SQL-Statements zu untersuchen, kann ein Tracing mit anschließendem Profiling eingesetzt werden.
- Gegenüber dem normalen SQL Trace bietet das Extended SQL Trace zusätzlich noch die Option, jedes einzelne Wait-Event nachzuverfolgen, das während des Tracing auftritt. Damit lassen

```
-- Extended SQL Trace ------------------------------------------------------
-- An-/Abschalten in der eigenen Session
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

-- Auffinden des Tracefiles
SELECT value
FROM   v$diag_info
WHERE  name = 'Default Trace File';

-- An-/Abschalten für eine andere Session
-- (SID und serial# müssen bekannt sein)
SQL> EXEC DBMS_SYSTEM.set_sql_trace_in_session(sid=>123, serial#=>1234, sql_trace=>TRUE);
SQL> EXEC DBMS_SYSTEM.set_sql_trace_in_session(sid=>123, serial#=>1234, sql_trace=>FALSE);

-- An-/Abschalten für eine andere Session
-- (si: Session-ID, se: Serial#, le: Level, 0=aus)
SQL> EXEC DBMS_SYSTEM.set_ev(si=>123, se=>1234, ev=>10046, le=>12, nm=>' ');
SQL> EXEC DBMS_SYSTEM.set_ev(si=>123, se=>1234, ev=>10046, le=>0, nm=>' ');

-- Auffinden des Tracefiles
SELECT p.tracefile
FROM   v$session s
       JOIN v$process p ON s.paddr = p.addr
WHERE  s.sid = 123;
```

Listing 2

sich auch die kleinsten Details einer Session protokollieren.

- Die beim Tracing erzeugten Protokolle sind meist sehr groß und schwer leserlich. Nur in seltenen Fällen wird es nötig sein, den exakten Ablauf in einer Session anhand der Rohdaten nachzuvollziehen.
- In allen anderen Fällen wird ein Profiler eingesetzt, der die Rohdaten gruppiert, sortiert und menschenlesbar aufbereitet. Die Sortierung unterstützt dabei meist das Top-Down-Prinzip, sodass die Top-Ressourcen und Top-SQLs ganz oben sichtbar sind.
- Im Verlaufe eines Tuning-Zyklus kommt ein Tracing wiederholt zum Einsatz, um Vorher-Nachher-Vergleiche anzustellen.

„tkprof", das „Bordmittel" von Oracle, hat allerdings zwei entscheidende Nachteile:

- Es kann mit den Zusatz-Informationen des Extended SQL Trace nichts anfangen.
- Es extrahiert die SQL-Ausführungspläne nicht aus dem Trace-File, sondern erzeugt sie zur Laufzeit. Die so generierten Ausführungspläne können daher von denen im Trace-File abweichen.

Besser geeignet sind hier zwei neuere Tools, die auch mit den erweiterten Informationen umgehen können. Eines davon ist der Trace Analyzer (TRCA) von Oracle, der unter der MOS Doc ID 224270.1 erhältlich ist. Das zweite Tool, das hier infrage kommt, nennt sich „orasrp". Es muss gegenüber dem TRCA keine Objekte in der Datenbank installieren und erfordert auch keine Verbindung zur Datenbank. Das macht es besonders für Ad-hoc- und Remote-Untersuchungen praktikabel.

„orasrp" wurde vom russischen Datenbank-Administrator Egor Starostin entwickelt. Das letzte Update der Software stammt vom April 2013, es ist also deutlich älter als das Release von Oracle 12.2. Allerdings können Trace-Files von Oracle 12.2 damit problemlos verarbeitet werden. *Listing 2* zeigt die Erzeugung eines Extended SQL Trace:

Nachdem das Trace File erzeugt und der Dateiname bekannt ist, kann „orasrp" mit „$ orasrp --google-charts tracefile.trc parsedfile.html" seine Arbeit beginnen. Die Anweisung „--google-charts" ist optional und erzeugt zusätzlichen Code für die Generierung einer Tortengrafik mithilfe von Google Charts. Nach ein paar Sekunden ist die Ausgabe in die HTML-Datei, die in der Kommandozeile benannt wird, fertig und die Datei kann in einem beliebigen Browser geöffnet werden *(siehe Abbildung 6)*.

Das Beispiel stammt aus einer realen Produktionsumgebung. Diese wurde zwar kurz zuvor von Oracle 11.2 nach 12.1 migriert, doch hatten sich danach die Laufzeiten verschiedener Batch-Jobs um mehr als den Faktor zwei verschlechtert – obwohl eine Verbesserung zu erwarten war, weil im Zuge der Migration sogar auf eine leistungsfähigere Hardware gewechselt wurde. Da ein erster Vergleich der SQL-Ausführungspläne keine Unterschiede zwischen dem alten und dem neuen System zeigte, fiel die Entscheidung, ein Extended SQL Trace speziell für die Batch-Session durchzuführen.

Im „Session Flat Profile" fällt auf, dass „db file sequential read" den größten Teil der Last ausmacht (also Random I/O von einzelnen Datenblöcken) und die maximale Wartezeit für dieses Event mit 1,65 Sekunden viel zu groß ist. Es geht bei diesem Event um einzelne Block-Zugriffe; wenn dabei auf eine Festplatte zugegriffen werden muss, dann sind bei heutigen Festplatten Zugriffszeiten von vier bis acht Millisekunden üblich (siehe *Abbildung 6*, gelbe Hervorhebung). Da „orasrp" die einzelnen Abschnitte untereinander verlinkt, springt man mit einem Klick auf das Event „db file sequen-

tial read“ zu den „Top 5 Statements per Event“ *(siehe Abbildung 7)*.

Hier fällt auf, dass das Top-SQL Nr. 1 bereits für 70 Prozent der Events verantwortlich ist. Darüber hinaus taucht bei diesem SQL wieder die lange, maximale Wartezeit von 1,65 Sekunden auf. Es gilt jetzt, dieses SQL näher zu betrachten; dazu ein Klick auf dessen Hash-Wert in der Tabelle *(siehe Abbildung 8)*.

Der Screenshot zeigt neben dem Ausführungsplan und den kumulativen Statistiken, wie sie auch von „tkprof“ her bekannt sind, ein Wait-Event-Profil, spezifisch für dieses SQL. Auch hier trägt das

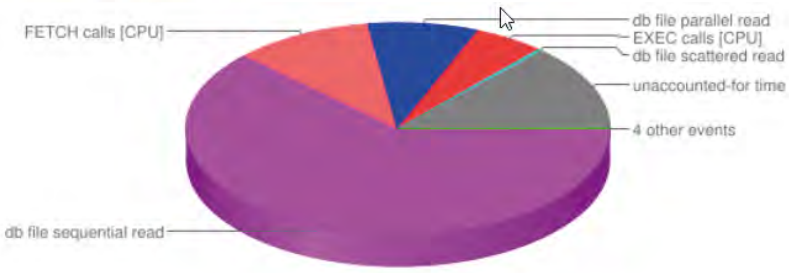
Session Summary

Instance Name: xdb
Node Name: oraxdb01
Oracle Release: 12.1.0.2.0
Session ID: 12.56325
Total SQL Statements: 37 (23 user statements, 14 internal statements)
Total Transactions: 4 (0 rollback(s), 1 read-only)

Report Shortcuts

Session Flat Profile
Top 5 Statements per Event
Session Call Graph
Events Histograms
Statements

Session Flat Profile Pie Chart



Session Flat Profile

Event Name	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
db file sequential read	84.2%	717.3377s	106,623	0.0067s	0.0001s	1.6540s
FETCH calls [CPU]	14.4%	122.5760s	186,707	0.0007s	0.0000s	0.6675s
db file parallel read	11.4%	96.6988s	3,605	0.0268s	0.0004s	0.8373s
EXEC calls [CPU]	7.3%	61.9997s	220,844	0.0003s	0.0000s	33.2023s
db file scattered read	0.4%	3.6598s	265	0.0138s	0.0003s	0.2412s
log file switch completion	0.0%	0.0798s	1	0.0798s	0.0798s	0.0798s
control file sequential read	0.0%	0.0012s	7	0.0002s	0.0001s	0.0003s
log file sync	0.0%	0.0003s	1	0.0003s	0.0003s	0.0003s
PARSE calls [CPU]	0.0%	0.0000s	35	0.0000s	0.0000s	0.0000s
unaccounted-for time	-17.7%	-150.4595s				
Total	**100.0%**	**851.8937s**				

Abbildung 6: Lastprofil der Session

Top 5 Statements per Event

db file sequential read

SQL Hash Value	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
3804044889	70.0%	502.0518s	48,650	0.0103s	0.0001s	1.6540s
1980990944	18.0%	129.3922s	30,672	0.0042s	0.0001s	0.2315s
1428548520	11.1%	79.3675s	24,182	0.0033s	0.0001s	0.4565s
2303779568	0.8%	6.0044s	2,642	0.0023s	0.0001s	0.1868s
3629108195	0.0%	0.2941s	28	0.0105s	0.0001s	0.0592s
3 others	0.0%	0.2276s	449	0.0005s	0.0001s	0.1164s
Total	**100.0%**	**717.3377s**	**106,623**	**0.0067s**	**0.0001s**	**1.6540s**

Abbildung 7: Top-SQL pro Event

SQL Statements

SQL Hash Value: 3804044889 SQL Id: czv8xxzjbu7kt uid: 73 depth: 2 optimizer mode: ALL_ROWS

Statement Text

```
SELECT *
FROM PERS_HISTORIE
WHERE PSN_ID = :B3 AND VORDAT = 'J' AND TO_CHAR(GUELTIG_AB,'YYYYMM') = :B2 ||
  :B1 AND ( UPPER(HIST_FELD) = 'BEITRAG' OR UPPER(HIST_FELD) = 'MTGLART' OR
  UPPER(HIST_FELD) = 'FKEITTG' OR UPPER(HIST_FELD) = 'ZPANFMN' OR
  UPPER(HIST_FELD) = 'REGSPN' OR UPPER(HIST_FELD) = 'ZPDAUER' OR
  UPPER(HIST_FELD) = 'ZAHLART' )
ORDER BY HIST_FELD, AEND_DAT
```

Statement Cumulative Statistics

Call	Cache Misses	Count	- Seconds -		Physical Reads	- Logical Reads -		Rows
			CPU	Elapsed		Consistent	Current	
Parse	0	0	0.0000s	0.0000s	0	0	0	0
Exec	0	24,085	0.8500s	1.3822s	0	0	0	0
Fetch		24,121	87.6810s	605.1096s	58,291	283,917	0	36
Total	**0**	**48,206**	**88.5310s**	**606.4918s**	**58,291**	**283,917**	**0**	**36**
Per Fch	0.0	2.0	0.0037s	0.0251s	2.4	11.8	0.0	0.0
Per Row	0.0	1,339.1	2.4592s	16.8470s	1,619.2	7,886.6	0.0	1.0

Statement Flat Profile

Event Name	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
db file sequential read	72.7%	502.0518s	48,650	0.0103s	0.0001s	1.6540s
db file parallel read	14.0%	96.6988s	3,605	0.0268s	0.0004s	0.8373s
FETCH calls [CPU]	12.7%	87.6810s	24,121	0.0036s	0.0000s	0.6675s
db file scattered read	0.5%	3.6598s	265	0.0138s	0.0003s	0.2412s
EXEC calls [CPU]	0.1%	0.8500s	24,085	0.0000s	0.0000s	0.0059s
Total	**100.0%**	**690.9415s**				

Abbildung 8: SQL-Details (Ausschnitt)

Event „db file sequential read" mit 72 Prozent am meisten zur Laufzeit bei. Auffällig ist auch die doch ziemlich hohe durchschnittliche Wartezeit mit mehr als zehn Millisekunden.

Die in diesem Beispiel gesammelten Informationen wurden im Rahmen eines Support Request mit Oracle Support besprochen. In diesem Fall kam heraus, dass es sich hier tatsächlich um ein Problem beim Filesystem-I/O handelt.

Historische Analyse ohne AWR

Auch für eine historische Analyse gibt es verschiedene Angebote, die vor allem auf eine Nachbildung der Active Session History (ASH) setzen. Da diese aber in der Regel eine Installation eigener Objekte in der Datenbank erfordern und dies oftmals als potenzielles Sicherheitsrisiko angesehen wird, soll hier auf ein herstellereigenes Tool von Oracle eingegangen werden: Statspack.

Oracle aktualisiert Statspack nach wie vor mit jedem Release, einschließlich Version 18c. Allerdings tauchten in den letzten Releases immer wieder Fehler auf, die nicht im Rahmen von Patch-Sets behoben wurden. Die zwei wesentlichen Bugs sind:

- Die Klassifizierung der Wait-Events wird in Statspack separat vom Data Dictionary vorgehalten, aber oft nicht aktualisiert. Dadurch erscheinen weniger relevante Events, sogenannte „Idle Events", in den Top-Events plötzlich ganz oben und verschleiern damit die Sicht auf die wirklich relevanten Events.
- Die Tabelle für historisierte SQL-Ausführungspläne ist nicht mehr identisch mit der Tabelle „PLAN_TABLE" im Data Dictionary. Es fehlt die Spalte „TIMESTAMP"; das stört allerdings nur, wenn man das Package „DBMS_XPLAN" mit einem mit Statspack historisierten Plan verwenden möchte.

Auf GitHub gibt es im „SQL-Zauberkasten" einen Skript-Satz, mit dem sowohl die manuellen Installationsschritte als auch der Workaround für die Idle Events automatisiert durchgeführt werden können *(siehe „https://github.com/Rendanic/SQL-Zauberkasten/tree/master/sql/statspack/spcreate")*. Für die fehlende Spalte gibt das Skript bei Bedarf ein SQL-Statement aus. Die Installation vereinfacht sich daher zu folgenden Schritten:

- Tablespace anlegen
- Das Skript „statspack_configuration.sql" auf die gewünschten Werte für Passwort, Tablespaces, Vorhaltedauer, Snapshot-Intervall und Detailtiefe anpassen
- Das Skript „statspack_create.sql" ausführen

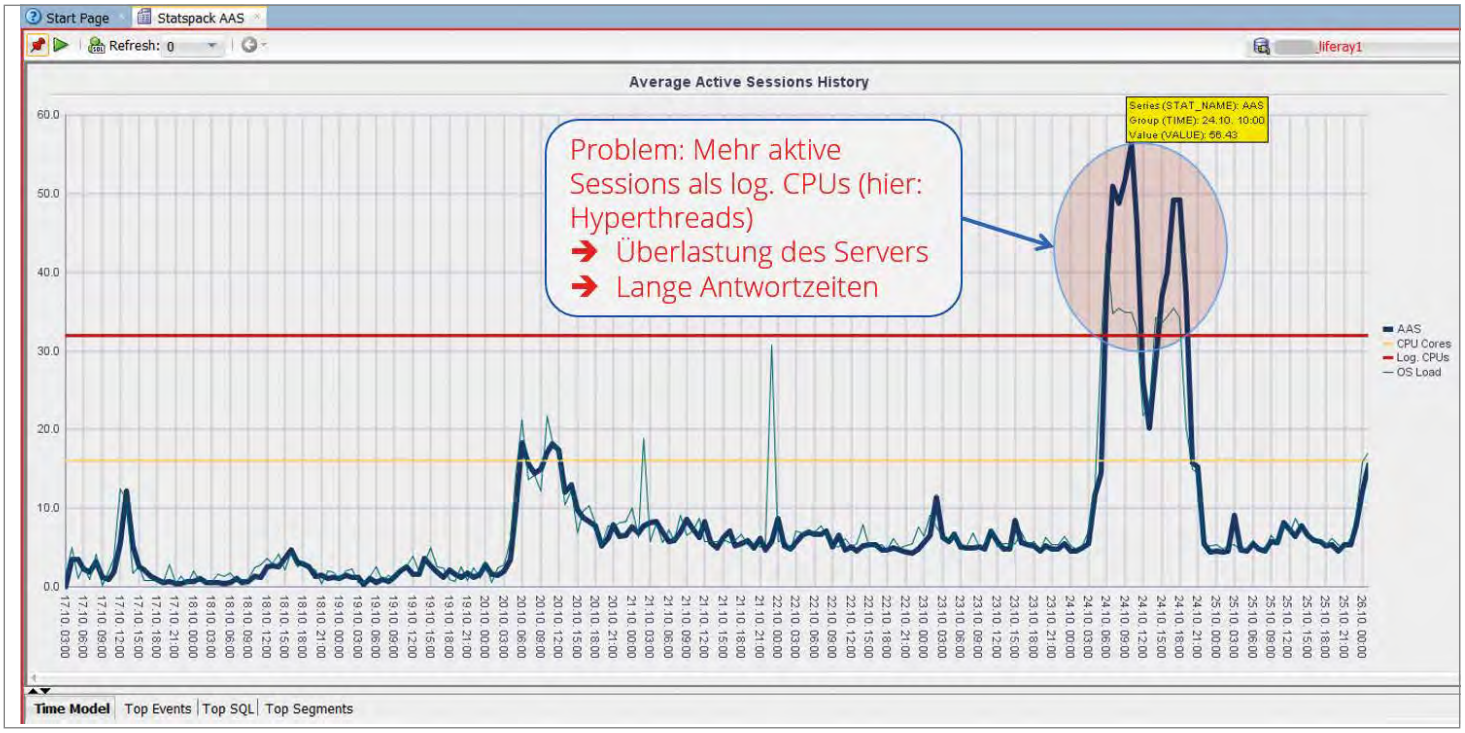
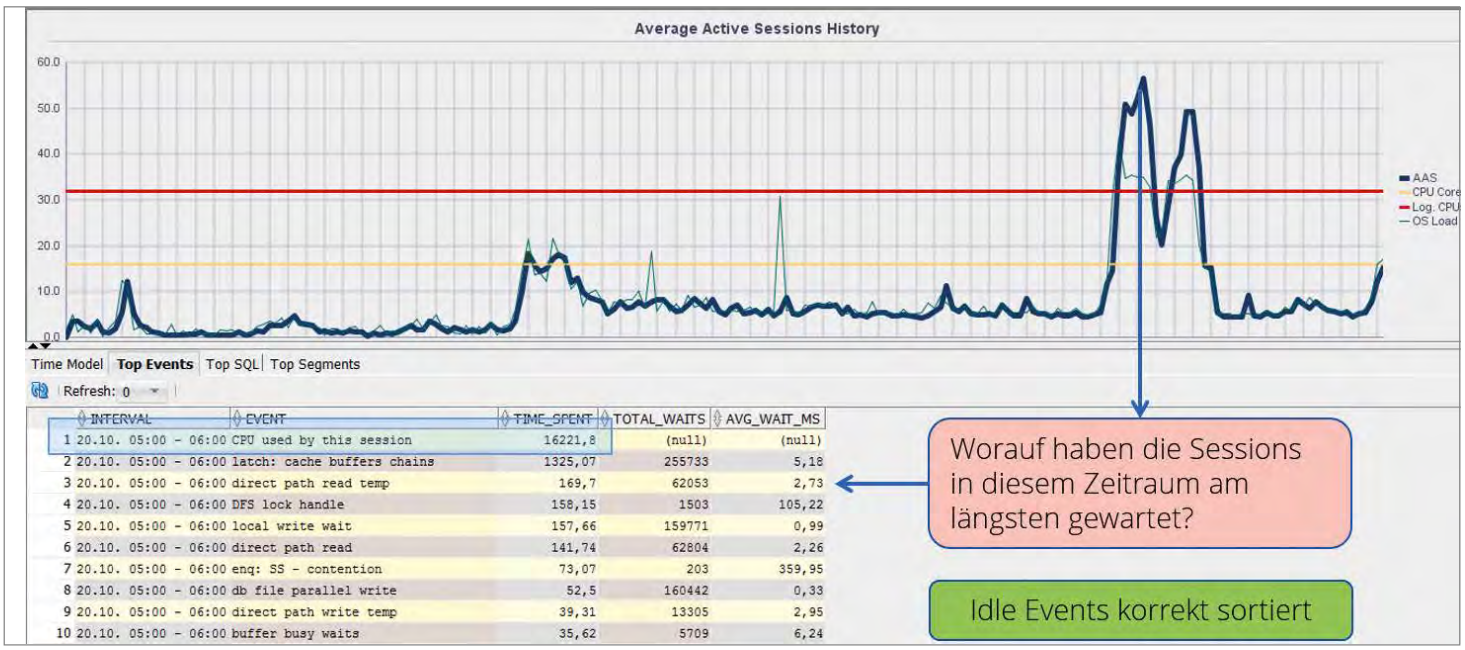


Abbildung 9: Average Active Sessions (AAS)

Nach der Ausführung sind die Jobs sofort aktiv und erstellen im gewünschten Rhythmus Snapshots von Performance-Metriken und ausgeführtem SQL. Die Auswertung der historisierten Daten kann, sobald mindestens zwei Snapshots vorliegen, mithilfe des SQL-Skripts „spreport.sql" erfolgen, das unter „$ORACLE_HOME/rdbms/admin" vorliegt. Dort steht in der Datei „spdoc.txt" auch eine kurze Dokumentation zur Verfügung.

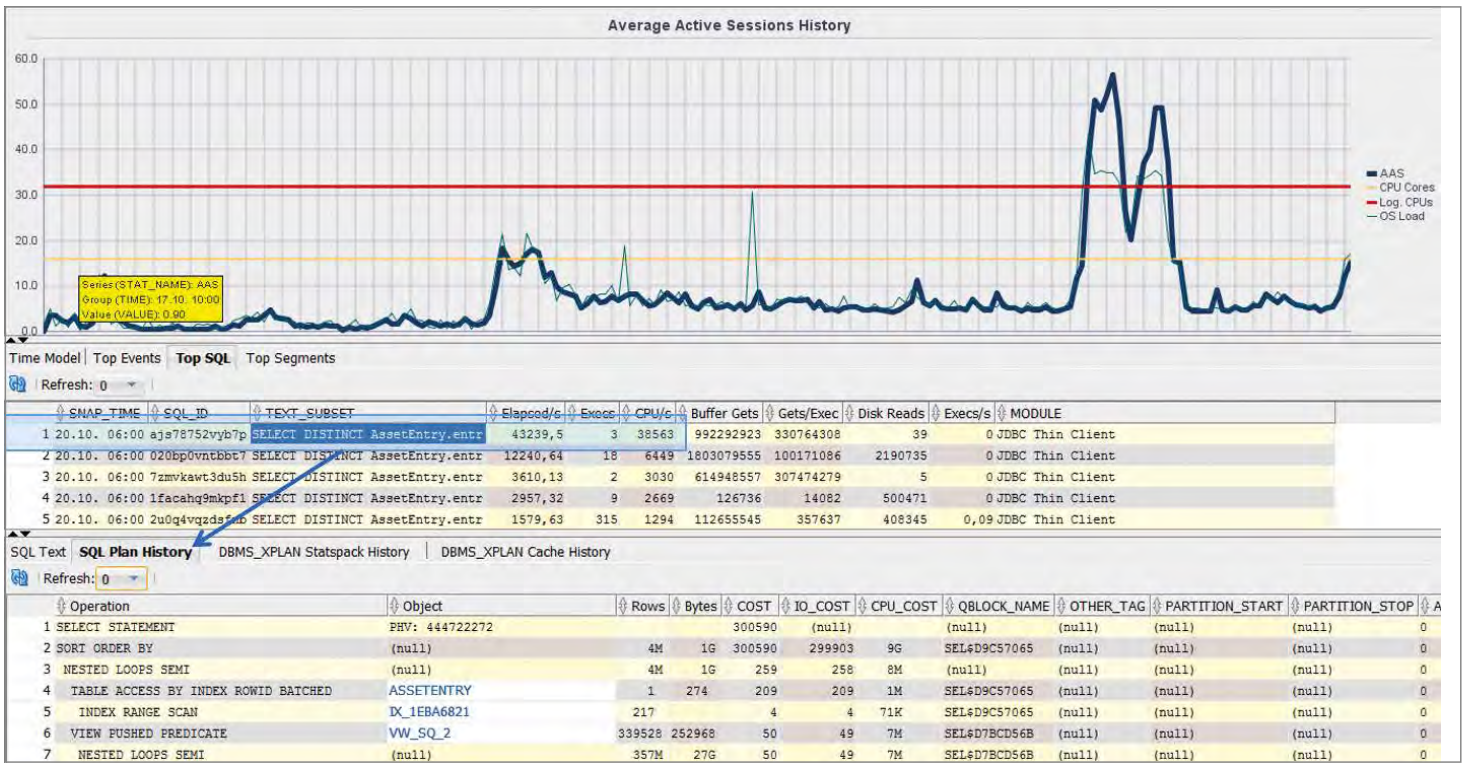
Wer mehr visuell veranlagt ist, bekommt mit dem Oracle SQL Developer ein weiteres kostenfreies Werkzeug an die Hand, mit dem sich die in den Statspack-Tabellen liegenden Daten grafisch aufbereiten lassen. Dazu gibt es, ebenfalls auf GitHub, fertige Reports zum Herunterladen und Importieren. *Abbildung 9* zeigt, wie so ein Report aussehen kann, der von einem produktiven System stammt.

Der erste Teil zeigt einen Graphen über die Historie von durchschnittlich aktiven Sessions im Zeitraum zwischen den einzelnen Snapshots. Falls verfügbar, wird die Last auf dem Server als dünne Linie dargestellt. Insbesondere bei mehreren Datenbank-Instanzen auf einem Server ist das ein wichtiger Vergleichswert, da die untersuchte Datenbank vielleicht gar nicht für die Serverlast



	INTERVAL	EVENT	TIME_SPENT	TOTAL_WAITS	AVG_WAIT_MS
1	20.10. 05:00 - 06:00	CPU used by this session	16221,8	(null)	(null)
2	20.10. 05:00 - 06:00	latch: cache buffers chains	1325,07	255733	5,18
3	20.10. 05:00 - 06:00	direct path read temp	169,7	62053	2,73
4	20.10. 05:00 - 06:00	DFS lock handle	158,15	1503	105,22
5	20.10. 05:00 - 06:00	local write wait	157,66	159771	0,99
6	20.10. 05:00 - 06:00	direct path read	141,74	62804	2,26
7	20.10. 05:00 - 06:00	enq: SS - contention	73,07	203	359,95
8	20.10. 05:00 - 06:00	db file parallel write	52,5	160442	0,33
9	20.10. 05:00 - 06:00	direct path write temp	39,31	13305	2,95
10	20.10. 05:00 - 06:00	buffer busy waits	35,62	5709	6,24

Abbildung 10: Top-Wait-Events



	SNAP_TIME	SQL_ID	TEXT_SUBSET	Elapsed/s	Execs	CPU/s	Buffer Gets	Gets/Exec	Disk Reads	Execs/s	MODULE
1	20.10. 06:00	ajs78752vyb7p	SELECT DISTINCT AssetEntry.entr	43239,5	3	38563	992292923	330764308	39	0	JDBC Thin Client
2	20.10. 06:00	020bp0vntbbt7	SELECT DISTINCT AssetEntry.entr	12240,64	18	6449	1803079555	100171086	2190735	0	JDBC Thin Client
3	20.10. 06:00	7zmvkawt3du5h	SELECT DISTINCT AssetEntry.entr	3610,13	2	3030	614948557	307474279	5	0	JDBC Thin Client
4	20.10. 06:00	1facahq9mkpf1	SELECT DISTINCT AssetEntry.entr	2957,32	9	2669	126736	14082	500471	0	JDBC Thin Client
5	20.10. 06:00	2u0q4vqzdafdb	SELECT DISTINCT AssetEntry.entr	1579,63	315	1294	112655545	357637	408345	0,09	JDBC Thin Client

	Operation	Object	Rows	Bytes	COST	IO_COST	CPU_COST	QBLOCK_NAME	OTHER_TAG	PARTITION_START	PARTITION_STOP	A
1	SELECT STATEMENT	PHV: 444722272			300590	(null)		(null)	(null)	(null)	(null)	0
2	SORT ORDER BY	(null)	4M	1G	300590	299903	9G	SEL$D9C57065	(null)	(null)	(null)	0
3	NESTED LOOPS SEMI	(null)	4M	1G	259	258	8M	(null)	(null)	(null)	(null)	0
4	TABLE ACCESS BY INDEX ROWID BATCHED	ASSETENTRY	1	274	209	209	1M	SEL$D9C57065	(null)	(null)	(null)	0
5	INDEX RANGE SCAN	IX_1EBA6821	217		4	4	71K	SEL$D9C57065	(null)	(null)	(null)	0
6	VIEW PUSHED PREDICATE	VW_SQ_2	339528	252968	50	49	7M	SEL$D7BCD56B	(null)	(null)	(null)	0
7	NESTED LOOPS SEMI	(null)	357M	27G	50	49	7M	SEL$D7BCD56B	(null)	(null)	(null)	0

Abbildung 11: Top-SQL mit historisierten Ausführungsplänen

verantwortlich ist. Im hervorgehobenen Zeitraum korreliert die Serverlast jedoch mit den AAS. Beide Metriken überschreiten die Anzahl der logischen CPUs (auf diesem System sind das Hyperthreads) deutlich; hier liegt offenbar eine Überlastung vor. Beim Klick auf einen Punkt im Graphen wird das erste Detailfenster befüllt *(siehe Abbildung 10)*.

Unter dem Reiter „Top Events" finden sich alle zwischen dem gewählten und dem vorherigen Snapshot aufgetretenen Wait-Events. Das SQL hinter dem Report berücksichtigt bereits die oben erwähnte Problematik mit den Idle Events und sortiert diese alle nach unten. Will man nun herausfinden, welches das Top-SQL im betrachteten Zeitraum war, wechselt man zum entsprechenden Reiter und bekommt daraufhin eine zusätzliche Detailstufe angezeigt *(siehe Abbildung 11)*.

Das Top-SQL wird zunächst nach Gesamtlaufzeit absteigend sortiert. Eine Sortierung nach anderen Spalten ist im SQL Developer durch Anklicken möglich.

Wenn ein SQL angewählt wird (durch einen Klick irgendwo innerhalb der gewünschten Zeile), werden die Details dazu im unteren Teilfenster angezeigt. In den historisierten Ausführungsplänen ist es darüber hinaus möglich, Einträge in der Spalte „Object" (die blau auf weiß hervorgehobenen Namen von Tabellen, Views und Indizes) anzuklicken und damit die Detailfenster zu diesen Objekten zu öffnen.

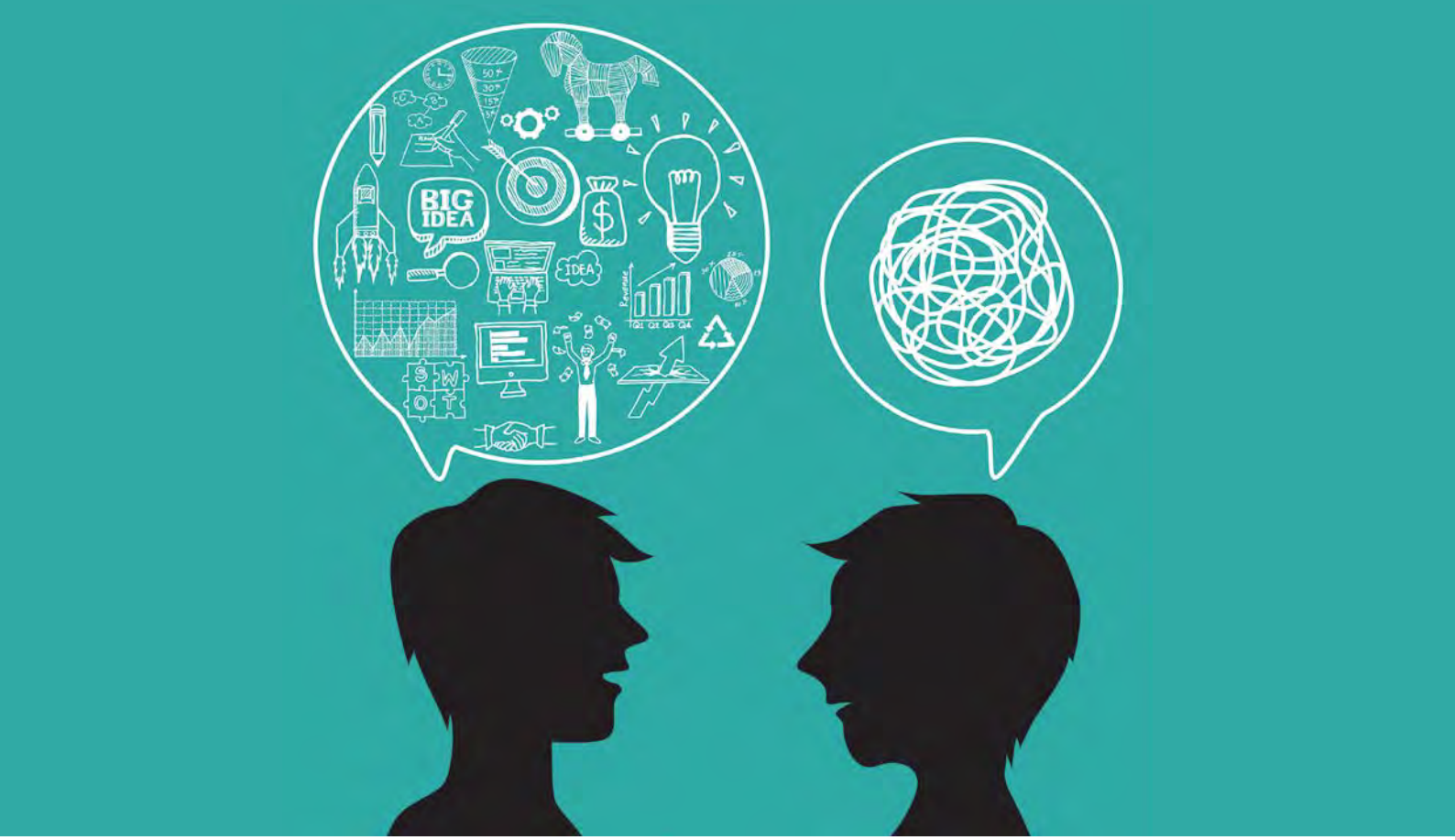
Fazit

Wenn es an die Diagnose von Performance-Problemen geht, kann mit Bordmitteln der Oracle Standard Edition und hilfreichen, kostenfreien Tools von Drittanbietern oder von Oracle Support auch ohne teure Optionen eine zügige Analyse durchgeführt werden.

Referenzen

- AMON: *https://sites.google.com/site/freetoolamon*
- Erzeugen und Auswerten von Extended SQL Trace: *https://oracle-base.com/articles/misc/sql-trace-10046-trcsess-and-tkprof*
- „orasrp"-Anleitung und Download: *http://oracledba.ru/orasrp*
- SQL_TRACE/Event 10046 Trace File Analyzer (trca): MOS Doc ID 224270.1
- Hilfsskripte zur Installation von Statspack: *https://github.com/Rendanic/SQL-Zauberkasten/tree/master/sql/statspack/spcreate*
- Performance-Reports für den SQL Developer: *https://github.com/oraculix/sql-developer-tools/tree/master/reports*, hier vorgestellt: „Statspack_AAS_3Level.xml"

Uwe Küchler
uwe.kuechler@opitz-consulting.com



Die Leistung von Oracle-Datenbanken verstehen

Jérôme Dubar, dbi services sa

Es gibt zahlreiche Quellen zur Feineinstellung und Leistungssteigerung von Oracle-Datenbanken. Das Thema ist umfangreich; man kann Tage damit verbringen, zu lernen und zu verstehen, wie eine Verbesserung erreicht werden kann. Wenn allerdings Performance-Probleme auftreten, muss schnell und effizient gehandelt werden. Dieser Artikel beschreibt einen strukturierten Ansatz, um zu verstehen, warum eine Oracle-Datenbank nicht die erwartete Leistungsfähigkeit bringt, und erläutert die Ausarbeitung eines Aktionsplans zur Leistungsverbesserung.

Die schnellste Komponente im Datenbank-Server ist der Prozessor: Mit hoher Geschwindigkeit, mehreren Kernen und ultraschnellem Cache können Prozessoren heutzutage große Datenmengen in Kürze verarbeiten. In Sachen „Geschwindigkeit beim Zugriff auf größere Datenmengen" ist der Arbeitsspeicher (RAM) das Maß aller Dinge. Wenn die Datenbank vollständig im RAM vorgehalten werden könnte, wäre sie pfeilschnell.

Oracle kann die Datenbank vollständig im Arbeitsspeicher vorhalten. Dazu sind allerdings kostspielige Optionen („Times Ten" oder „In-Memory") erforderlich, die für bestimmte Zwecke entwickelt und optimiert wurden. Bei den meisten Oracle-Datenbanken sind die Datenbank-Dateien allerdings auf Laufwerken gespeichert und der RAM fungiert nur als Datencache.

Daten-Dateien lassen sich auf lokalen Laufwerken jeder Art speichern: Festplatten (HDD), Solid State Disks (SSD) und NVMe SSD (SSD mit direktem Anschluss an den PCIe-Bus). Diese Laufwerke können in einem Speicher-Netzwerk (Storage Area Network, SAN) oder einem netzgebundenen Speicher (Network Attached Storage, NAS) liegen. Alle diese Technologien liegen jedoch (weit) hinter der Geschwindigkeit von RAM-Speichern *(siehe Abbildung 1)*.

Die meisten Aktivitäten der Datenbank bestehen im Auslesen von Daten

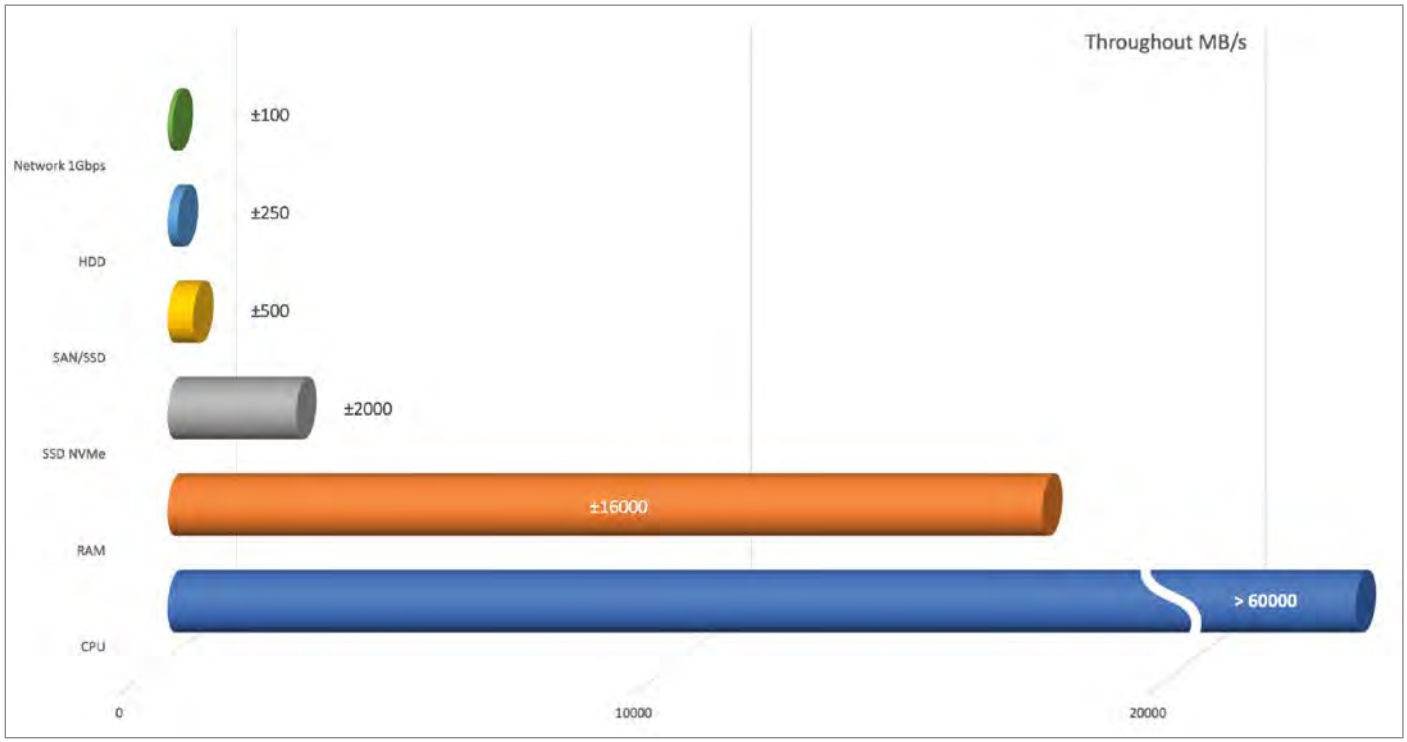


Abbildung 1: Die schnellsten Bauteile eines Systems sind Prozessor und RAM

(„SELECT“). Datenblöcke zur Ausführung eines Statements liest Oracle erst vom Laufwerk und legt sie dann in einem Speicherbereich namens „System Global Area“ (SGA) ab. Wenn man dann später wieder auf diese gleichen Datenblöcke zugreift, liest Oracle sie direkt aus der SGA, also schneller. Die Datenlese-Geschwindigkeit sollte sehr hoch sein, wenn genügend SGA für die häufigsten Statements zur Verfügung steht und die Instanz schon seit längerer Zeit läuft – eine neu gestartete Instanz weist eine leere SGA auf.

Eine andere Aufgabe der Datenbank besteht im Schreiben von Datenblöcken auf das Laufwerk in die Datendateien („INSERT“/„UPDATE“/„DELETE“). Da Oracle jedoch darauf optimiert ist, in kritischen Momenten unkritische Tätigkeiten später zu erledigen, werden geänderte Blöcke erst nur in die SGA geschrieben; das Schreiben in die Datei erfolgt auf asynchrone Weise zu einem späteren Zeitpunkt. In der Tat schreibt Oracle geänderte Blöcke in die Datendateien, wenn es aus Wiederherstellungsgründen nötig ist. Es gibt also keine Garantie, dass Änderungen einer Transaktion in die Datendateien geschrieben werden, wenn man keinen manuellen Checkpoint erstellt oder die Datenbank ordnungsgemäß herunterfährt.

Der einzige synchrone Prozess beim Einfügen/Aktualisieren/Löschen von Daten ist das fortlaufende Schreiben von Datenänderungen in Redolog-Dateien. Diese garantieren die Wiederherstellung nach einem Absturz – sie werden kontinuierlich auf dem Laufwerk und spätestens bei einem COMMIT aktualisiert. Die Geschwindigkeit von Redologs ist kritisch für Datenbanken, da diese quasi ein Nadelöhr darstellen. Deshalb empfiehlt Oracle die Verwendung dedizierter und schneller Laufwerke für diese Dateien. Datendateien können auf langsameren Laufwerken liegen.

Einen Leistungsbericht über die Datenbank erstellen

Der erste Schritt zur Leistungsanalyse der Datenbank ist die Erstellung eines Leistungsberichts. Damit lassen sich am leichtesten alle Leistungsdaten (selbst solche, die nie verwendet werden) in einer einzigen Datei sammeln und verdichten. Dieser Leistungsbericht wird später analysiert.

Leistungsberichte können mit integrierten Oracle-Tools (Automatic Workload Repository (AWR) oder Statspack) erstellt werden. Zusätzliche Werkzeuge sind nicht notwendig. AWR erfordert die Enterprise Edition und das Diagnostics-Pack. Sofern beide lizenziert sind, ist AWR direkt einsatzbereit.

Wer das Diagnostics-Pack nicht lizenziert hat oder seine Datenbank mit der Standard Edition betreibt, muss stattdessen Statspack verwenden. Es ist kostenlos, allerdings nicht vorinstalliert, und kann direkt durch Aufruf eines Skripts installiert werden *(siehe Listing 1)*. Zu beachten ist, dass Statspack bei 12c Opti-

```
sqlplus / as sysdba
@?/rdbms/admin/spcreate sqlplus / as sysdba
```

Listing 1

```
sqlplus / as sysdba
@?/rdbms/admin/awrrpt
```

Listing 2

```
sqlplus / as sysdba
@?/rdbms/admin/spreport
```

Listing 3

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
ORCL	4248975166	orcl	1	23-Juil.-11 21:07	11.2.0.1.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
HQSBIS02	Microsoft Windows x86 64-bit	12	6	1	31.99

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	5701	01-Août -11 10:00:24	91	2.6
End Snap:	5869	08-Août -11 10:00:32	111	3.4
Elapsed:		10,080.13 (mins)		
DB Time:		24,661.13 (mins)		

Abbildung 2: Die angezeigt DB-Zeit

mierungen erfordert *(siehe „https://blog.dbi-services.com/statspack-idle-events“)*.

Die zwei Leistungstools (niemals beide für die gleiche Datenbank verwenden) funktionieren auf gleiche Art: In festen Intervallen (voreingestellt jede Stunde) nimmt das Tool einen sogenannten „Snapshot“ der aktuellen Leistungsdaten auf und schreibt diese Daten in dedizierte Tabellen. Ein Leistungsbericht ist die konsolidierte Form dieser Daten in einem Zeitfenster zwischen zwei Snapshots. Mit AWR erfolgt die Erstellung eines Leistungsberichts wie in *Listing 2*, mit Statspack wie in *Listing 3*.

Je nach Tool sind drei bis fünf Fragen zu beantworten:

- Dateityp der Ausgabe, hauptsächlich HTML oder Text (Statspack: nur Text)
- Anzahl der Tage des Snapshot-Verlaufs zur Auswahl (Statspack zeigt alle verfügbaren Snapshots an)
- Nummer des Snapshots für den Startzeitpunkt
- Nummer des Snapshots für den Endzeitpunkt
- Dateiname der Berichtsdatei

Die Leistung einer Datenbank beziehungsweise die Anforderung an die Datenbank schwankt je nach Tag, Tageszeit, Auslastung, Größe der Datenmengen etc. Es empfiehlt sich, mit der Erstellung eines Leistungsberichts zur Spitzenzeit der Datenbank-Anforderungen (üblicherweise die Spitzenzeit der Applikationsnutzung) zu beginnen. Ist die Spitzenzeit nicht bekannt, besteht ein weiterer Ansatz darin, die Durchschnittsleistung der Datenbank über einen längeren Zeitraum zu untersuchen, etwa während einer ganzen Woche, und ruhig mehrere Leistungsberichte in verschiedenen Zeiträumen zu erstellen. Dies erhöht das Verständnis über das Verhalten der Datenbank. Anmerkung: Zur Analyse eines Leistungsberichts sollte der Zeitraum allerdings üblicherweise maximal eine Stunde betragen, da sonst temporäre Spitzen im Durchschnitt untergehen.

Kurz gesagt: Die Oracle-Tools AWR und Statspack stehen zur Erstellung von Leistungsberichten bereit; es ist keine Fremdhersteller-Software notwendig. Leistungsberichte sind lesbare Dateien mit absoluten und aggregierten Daten für ein bestimmtes Zeitfenster. Es empfiehlt sich, Berichte für mehrere Zeitfenster zu erstellen.

Die DB-Zeit der Datenbank prüfen

Die DB-Zeit stellt diejenige Zeit dar, die im System von der Datenbank durch Benutzer beziehungsweise Applikationen zwischen dem Start- und dem Ende-Snapshot verbraucht wurde. Dies ist eine wichtige Zahl zur Bestimmung der Auslastung der Datenbank. Eine hohe DB-Zeit bedeutet eine hohe Auslastung: eventuell viele Nutzer, die durch die Applikation viele Statements ausführen – oder eine nicht optimierte Datenbank.

Tatsächlich lässt sich ohne weitere Angaben nicht sagen, ob eine bestimmte DB-Zeit wirklich hoch ist oder nicht – das hängt hauptsächlich vom System ab. Falls diese aus einem großen dedizierten Server mit Dutzenden Prozessoren besteht, stellt eine vermeintlich hohe DB-Zeit wahrscheinlich keine große Last dar im Vergleich zu einem Server mit einem einzigen Kern, der die gleiche Datenbank mit den gleichen Aktivitäten ausführt. Die DB-Zeit sollte stets mit der während der verstrichenen Zeit der nach Leistungsbericht verfügbaren Prozessorzeit verglichen werden, wobei zu beachten ist, dass Prozessorzeit nicht gleich verstrichene Zeit ist. Die Formel lautet: „Verfügbare Prozessorzeit = verstrichene Zeit x Anzahl der Prozessorkerne im Server“.

Wenn beispielsweise die DB-Zeit bei einer verstrichenen Zeit von einer Stunde zwanzig Minuten beträgt, ist dies eine recht große DB-Zeit für den Server mit einem einzigen Kern, denn die Datenbank verbraucht ein Drittel der verfügbaren Prozessorzeit. Wenn der Server hingegen über zwei Prozessoren mit je 18 Kernen verfügt, stehen 36 Stunden Prozessorzeit zur Verfügung, und zwanzig Minuten sind nur etwa ein Prozent der Server-Rechenleistung. Die DB-Zeit ist recht leicht im AWR- oder Statspack-Bericht nachzuprüfen, sie steht am Anfang des Berichts *(siehe Abbildung 2)*.

In diesem Beispiel besitzt der Server einen Single-Socket-Prozessor mit sechs Kernen (die Angaben lassen sich natürlich auch direkt im System prüfen). Man sollte sich der Anzahl der Prozessoren (CPUs, im Beispiel 12) bewusst sein. Tatsächlich ist dies die Anzahl der Threads, und ein Thread ist nicht einem dedizierten Kern gleichzusetzen. Stehen zwei Threads pro Kern zur Verfügung (etwa bei Hyper-Threading mit Intel-Prozessoren), kann ein Kern theoretisch zwei Instruktionen-Ströme vom Betriebssystem gleichzeitig ausführen. Tatsächlich ergibt sich jedoch mit zwei Threads pro Kern nur ein Leistungsgewinn von 1,2 bis 1,3 im Vergleich zu einem Kern ohne Hyper-Threading. Dieses Mehr von zwanzig bis dreißig Prozent ist also nur als kleine Leistungsverstärkung und nicht als Verdoppelung der zur Verfügung stehenden Prozessor-Leistung anzusehen.

Die gesamte auf dem Server verfügbare Prozessorzeit beträgt im Beispiel 6 x 10.080 Minuten (1 Woche) = 60.480 Minuten. Die DB-Zeit ist recht hoch: Angenommen sie ist reine CPU-Zeit, so stellen die 24.661 Minuten mehr als vierzig Prozent der Rechenleistung des Systems dar, die durchschnittlich von dieser Datenbank beansprucht wird. Wenn man bedenkt, dass die Aktivitäten während des Tages schwanken, ist es wahrscheinlich, dass dies bei dieser Da-

tenbank während Spitzenzeiten zu Kapazitäts-Engpässen führt.

Was geschieht, wenn Ihre Datenbank versucht, mehr Prozessor-Ressourcen zu verbrauchen, als im System verfügbar ist? Nun ja, überhaupt nichts. Alles läuft weiter, doch die Datenbank gibt die Ergebnisse an die Nutzer oder die Anwendung später zurück. Selbst wenn das System durch die Datenbank stark ausgelastet ist, geht alles weiter. Oracle-Datenbanken laufen auf stark ausgelasteten Systemen recht stabil, doch die Antwortzeiten sind sehr lang.

Es ist zu bedenken, dass auf dem System mehr als eine Datenbank laufen kann und eventuell andere Programme die Server-Ressourcen anteilig nutzen. Bei Systemen mit mehreren Oracle-Datenbanken sind die DB-Zeiten aller Datenbanken zu addieren, wobei man die Leistungsberichte für jede Datenbank im gleichen Zeitraum ausführt, um zu sehen, ob sich die gesamte DB-Zeit mit dem System vereinbaren lässt.

Kurz gesagt: Im Leistungsbericht ist der CPU-Teil der Datenbank-DB-Zeit zu prüfen und mit der im System verfügbaren Prozessorzeit zu vergleichen. Sofern es nur wenige Prozent sind, ist die Datenbank-Aktivität gering, sodass keine Leistungsprobleme aufgrund von Prozessor-Engpässen auftreten sollten. Sollte die Prozessor-DB-Zeit recht groß sein, geht es zur Leistungsprofil-Analyse. Sofern auf dem Server mehrere Datenbanken laufen, sind zur Bestimmung der Gesamtlast des Systems die DB-Zeiten aller Datenbanken zu addieren.

Das Leistungsprofil der Datenbank untersuchen

Die DB-Zeit ist einfach die Gesamtzeit, die die Vordergrund-Prozesse der Datenbank im System beanspruchen. Diese Zeit kann mit leistungsfähigen und weniger leistungsfähigen Aufgaben verwendet werden. Im AWR- oder Statspack-Bericht stehen die sogenannten „Top X Timed Events“ (Name hängt von der Version ab). In der Tat finden Sie dort, wo Ihre Datenbank Zeit verbraucht, und genauer, wie die DB-Zeit aufgeteilt wird *(siehe Abbildung 3)*.

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		1,651,293		62.8	
db file sequential read	300,139,356	657,763	2	25.0	User I/O
PX Deq Credit: send blkd	2,751,058	243,931	89	9.3	Other
db file scattered read	136,888,796	48,752	0	1.9	User I/O
log file sync	9,174,074	13,958	2	.5	Commit

Abbildung 3: Die Top-5-Timed-Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
db file scattered read	57,378,623	182,501	3	28.6	User I/O
db file sequential read	40,890,463	164,466	4	25.8	User I/O
CPU time		91,136		14.3	
RMAN backup & recovery I/O	1,063,783	21,432	20	3.4	System I/O
control file sequential read	2,834,991	18,036	6	2.8	System I/O

Abbildung 4: Die Top-5-Timed-Events einer anderen Datenbank

In der Einführung wurde gesagt, dass Prozessor und Arbeitsspeicher die schnellsten Komponenten auf dem Server sind. Bei der Datenbank in *Abbildung 3* werden mehr als 62 Prozent der DB-Zeit als Prozessorzeit benötigt (Prozessor- und Arbeitsspeicher-Zugriffszeit). Das sind recht gute Nachrichten. Andere Events haben eine Wait-Klasse, die anzeigt, dass die Datenbank auf etwas wartet. Tatsächlich wartet alles, was DB-, jedoch keine Prozessor-Zeit („Service Time“) ist. Hier besteht die Haupt-Wait-Klasse aus Nutzer-IOs, hauptsächlich beim Lesen aus Datenbank-Dateien (Datendateien). Der „db file sequential read“, also das sequentielle Lesen einzelner Blöcke, gilt (meistens) als „gutes Lesen“, da direkt auf die einzelnen Blöcke auf dem Laufwerk zugegriffen wird, in denen die zu lesenden Daten erwartet werden, was als indiziertes Lesen zu verstehen sind. Auf der anderen Seite sind Lesevorgänge („db file scattered read“) über ganze Laufwerks-Bereiche „schlechtes Lesen“, da sie üblicherweise aus vielen gelesenen Blöcken wenige Daten filtern. Dies kommt beispielsweise bei vollständigen Tabellenscans vor; Oracle muss hier eine ganze Tabelle einlesen, um (wahrscheinlich) eine begrenzte Anzahl von Zeilen zurückzugeben.

Es ist bekannt, dass nicht alles im Datenbank-Cache (Teil der SGA) gespeichert werden kann, insbesondere wenn die Datenbank groß ist, sodass gelegentliches Lesen von Datendateien normal ist. Aber in diesem Beispiel sind 25 Prozent Lesevorgänge einzelner Blöcke ein

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~
                                                           Avg   %Total
                                                          wait    Call
Event                                        Waits    Time (s)    (ms)    Time
---------------------------------------- ---------- ---------- ------ ------
log file switch (checkpoint incomplete)         301        315   1048   70.7
CPU time                                                   116          26.0
db file async I/O submit                        315          5     16    1.2
log file parallel write                       1,771          4      2     .9
control file parallel write                   2,631          2      1     .4
          -------------------------------------------------------------
```

Abbildung 5: Die Top-5-Timed-Events einer weiteren Datenbank

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
286,377	282,928	1	286376.56	10.89	dsv0v7u028vqm		DECLARE job BINARY_INTEGER := ...
284,731	281,336	0		10.83	8ru089sd1frx4		INSERT /*+ BYPASS_RECURSIVE_CH...
253,330	252,482	11	23030.02	9.64	487c0771ad7wz	SQL*Plus	BEGIN p_vad_delai_stk(1000); E...
238,339	237,948	847,982	0.28	9.06	d3fqcs9588q8r	SQL*Plus	SELECT DATLIV, SUM (QTERAL) Q...
230,375	227,205	33,377	6.90	8.76	q2wzmv1ckp6cp		SELECT SUM(EVL.QTECDE-LEAST(EV...
177,448	174,443	4	44361.91	6.75	bzwu8tavqm0rk	SQL*Plus	BEGIN PKG_PFN_OPTIMRAO.PFN_OPT...
175,122	173,965	5	35024.39	6.66	9m0jt4zczyyk7	SQL*Plus	CREATE TABLE PFN_OPTIMRAO_PRO_...
150,800	8,340	6	25133.37	5.74	820aw69dqttq9	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...
106,144	105,859	1,799,504,320	0.00	4.04	bathukprjfwau		SELECT SUM(LEAST(EVP.QTECDE, E...
89,887	89,455	5,497,174	0.02	3.42	q8f9qctabbj2c		SELECT GREATEST(MAX(v.P12), M...
86,820	86,056	48	1808.74	3.30	2qs1bth05bhnz		SELECT decode(count(1), 1, m...
51,361	51,321	1,886,352,880	0.00	1.95	2zw0cnjfcrzk5		SELECT MIN(REC.NUMEVE) FROM EV...
50,565	2,064	187	270.40	1.92	8zwqshkraujah	SQL*Plus	BEGIN P_PFN_VTIC_NONAME; END; ...
48,040	1,974	182	263.96	1.83	4tzd11tdbr17f	SQL*Plus	SELECT EVE.CODSOC, EVE.ACHVTE...
40,487	972	1	40487.25	1.54	d6624jt44yz65	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...
32,801	9	1	32801.34	1.25	1jp2a66xbumkx	TOAD background query session	select numeve, count(1) from ...
28,869	2,271	7	4124.08	1.10	9k2814ab24khs	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...

Abbildung 6: SQL ordered by elapsed time

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
307,446,396	445	690,890.78	21.90	4252.36	11185.38	af3g17f9fscpk	SQL*Plus	BEGIN p_pfn_maj_codeta_er(1); ...
286,208,661	425	673,432.14	20.39	3964.13	10319.55	aqxc3tbt7hjbb	SQL*Plus	SELECT MSK.* FROM MSK, UT_SPL...
65,048,020	10	6,504,802.00	4.63	1380.24	4258.35	fm6qpks4k24fm	SQL*Plus	BEGIN p_vad_fin_fact(1000, 'FI...
65,041,544	10	6,504,154.40	4.63	1372.98	4232.72	2w6yq16h5cn6k	SQL*Plus	SELECT EVP.CODSOC, EVP.ACHVTE...
40,011,973	6	6,668,662.17	2.85	8339.57	150800.22	820aw69dqttq9	bdd_serveur.exe@FEJ20040 (TNS V1-V3)	select codsoc, z40l_1, z40l_...
34,298,285	9	3,810,920.56	2.44	1799.39	4637.62	79j290sm1qxp6	SQL*Plus	BEGIN p_vad_decisi_liv(1000, '...
33,542,189	5	6,708,437.80	2.39	2065.05	4380.30	8ptj1yrphkmth	SQL*Plus	Begin p_pfn_export_upd_datmo...
28,599,963	6	4,766,660.50	2.04	1085.52	2368.88	71ur4w5tdb1s7	wireportserver.exe	SELECT TIE_MAG.SIGTIE, TIE_...
28,261,882	7	4,037,411.71	2.01	625.52	1934.34	cbudavn2n1f1v	SQL*Plus	BEGIN p_vad_reservation(1000, ...
24,405,594	8	3,050,699.25	1.74	2154.38	18535.51	c94zqvqmzrtqj		SELECT MevMsk.CODSOC, ...
22,259,295	187	119,033.66	1.59	2064.42	50565.50	8zwqshkraujah	SQL*Plus	BEGIN P_PFN_VTIC_NONAME; END; ...
21,299,663	182	117,031.12	1.52	1973.84	48040.24	4tzd11tdbr17f	SQL*Plus	SELECT EVE.CODSOC, EVE.ACHVTE...
19,014,844	10	1,901,484.40	1.35	404.42	1608.05	3wcah505p9xcm	SQL*Plus	BEGIN p_fej_csk_differes(1000)...
19,001,065	10	1,900,106.50	1.35	383.81	1570.02	qtnum77ba11fz	SQL*Plus	SELECT EVL.CODPRO, SUM (QTECD...
18,803,749	710	26,484.15	1.34	1040.21	4582.49	cyjq8qypxtuaf		SELECT DISTINCT NUMEVE FROM EV...
18,679,963	904	20,663.68	1.33	1005.85	4618.35	qpqu7q5zq8p2u		UPDATE EVT JEvt SET DATMOD=:1,...
18,456,776	9	2,050,752.89	1.31	462.70	1502.79	130mwwtxqx2dq	SQL*Plus	BEGIN p_vad_maj_stcde(1000, 'M...
18,166,529	9	2,018,503.22	1.29	559.05	5116.02	06pmwypsjzv0r	SQL*Plus	SELECT EVE.CODSOC, EVE.ACHVTE...
16,316,638	9	1,812,959.78	1.16	512.02	1259.92	5f90z6s8m8htb	SQL*Plus	UPDATE EVT SET CODZN18 = ' ', ...
16,105,086	9	1,789,454.00	1.15	379.51	1157.53	8zf3mz5ryf8y6	SQL*Plus	UPDATE EVT SET CODZN14 = 'N', ...
15,256,508	7	2,179,501.14	1.09	341.61	1058.24	801hhhpdpz426	SQL*Plus	SELECT EVP.CODSOC, EVP.ACHVTE...
14,351,606	6	2,391,934.33	1.02	2316.23	13080.74	6q5n7kkzhxubu	SQL*Plus	begin w_fej_maj_adr_cmp(1); e...

Abbildung 7: SQL ordered by reads

recht großer Teil der Datenbank-Aktivität. Wie lassen sich diese Lesevorgänge auf einfache Weise verringern? Wahrscheinlich ist die SGA-Größe zu erhöhen. Wenn die SGA größer ist, steigt die Wahrscheinlichkeit, Blöcke aus dem Cache-Teil der SGA zu lesen und physische Lesevorgänge auf ein Laufwerk zu verhindern. Man sollte sich jedoch dessen bewusst sein, dass eine Vergrößerung der SGA nur möglich ist, wenn das System über genügend RAM verfügt. Heutzutage ist dies kaum mehr ein Problem, da Arbeitsspeicher günstig ist und leicht aufgerüstet werden kann.

Abbildung 4 zeigt das Leistungsprofil einer anderen Datenbank. Während die vorherige Datenbank kein so schlechtes Leistungsprofil hatte, ist dieses hier wirklich dürftig. Die aktive Arbeit der Datenbank (CPU- beziehungsweise Service-Time) macht nur 14 Prozent der gesamten DB-Zeit aus. Den Großteil der Zeit wird auf physische IOs gewartet. Oracle muss Blöcke vom Laufwerk lesen, und Laufwerke können diese Daten natürlich nur gemäß ihrer Leistungscharakteristik liefern. Dies könnte ein selbstzerstörerisches Verhalten sein: Das Warten auf Blöcke auf dem Laufwerk verbraucht Prozessor-Zeit, ohne dass irgendetwas berechnet wird. Dies kann die verfügbaren Rechenzyklen für reine Prozessoren senken. Wenn dann die Redologs auf dem gleichen Speicher wie die Datendateien liegen (falls das Design einer Oracle-Datenbank nicht beachtet wurde), könnte die Leistung sogar noch schlechter ausfallen.

Diese Datenbank benötigt zwei Feineinstellungs-Vorgänge. Zunächst könnte eine Erhöhung des SGA die IO-Vorgänge verringern. Die Feineinstellung (Ausführungsplan, Zugriffsstrukturen) für weniger SQL-Statements könnte die Notwendigkeit von Full-Table-Scans („schlechter

```
SELECT ticb.sigtie, cpr.datmod, cpr.utimod
  FROM soc1.mev a, soc1.ticb, soc1.mev b,
       soc1.cpr, soc1.tbl ttu786
  WHERE b.codsoc=:p_codsoc and b.codent='CPR' and b.segment=' '
       and ttu786.codtbl='786' and ttu786.cletbl='WEB'
       and cpr.codsoc=b.codsoc_phy and cpr.datmod>=ttu786.lir
       and a.codsoc=:p_codsoc and a.codent='TIE'
       and a.segment='CLI'
       and ticb.codsoc=a.codsoc_phy and ticb.typtie='CLI'
       and ticb.typcpr='PRV' and ticb.codcb=cpr.codcb;
```

Listing 4

Lesevorgänge“) reduzieren; man müsste also die SQL-Statements ausmachen, die die meisten Ressourcen verbrauchen (siehe nächstes Kapitel).

Abbildung 5 zeigt ein letztes Beispiel mit einer weiteren Datenbank. Hier liegt ein anderes Problem vor: Die Datenbank kann den Wechsel auf die nächste Redolog-Datei nicht schnell genug vornehmen, 70 Prozent der DB-Zeit sind dafür notwendig. Hier gilt es, die Konfiguration der Redologs zu untersuchen; vielleicht sind sie zu klein oder es gibt nicht genug Gruppen?

Es gibt viele andere Events, die Wartezeiten verursachen können. Wenn ein Wait-Event beträchtlich ist, untersucht man, worauf er sich bezieht.

Kurz gesagt: Die DB-Zeit sollte hauptsächlich aus Prozessor-Zeit bestehen, der Zielwert liegt bei 70 Prozent und mehr, weil Prozessor-Zeit tatsächliche DB-Verarbeitung bedeutet und nicht Wartezeit. DB-Datei-Events („Wait-Class User I/O“) sind Zugriffe auf Datendateien. Die „db file scattered read“-Lesevorgänge sind nicht optimierte, „db file sequential read“ dagegen optimierte. Die „db file sequential read“-Lesevorgänge lassen sich durch Erhöhung der SGA-Einstellungen verringern. Die richtige Konfiguration der Redologs ist ebenfalls wichtig für die Leistung. Es gibt viele andere mögliche Wait-Events.

Die Haupt-SQL-Statements untersuchen

Es gibt wahrscheinlich Tausende von Statements, die täglich an den Datenbanken ausgeführt werden, aber darum braucht man sich keine Sorgen zu machen. Oracle führt die meisten dieser Statements so schnell aus, wie es sie verarbeiten kann. Von den meisten davon wird nie die Rede sein. Sehr häufig hängen die Leistungsprobleme mit sehr wenigen Statements zusammen; die Optimierung dieser zwei oder drei Statements kann die DB-Zeit drastisch senken und die Gesamtleistung verbessern.

Die Haupt-SQL-Statements stehen weiter unten im AWR- oder Statspack-Bericht. Man findet dort mehrere Tabellen, die nach Merkmalen sortiert sind. Zuerst fokussiert man sich auf die SQL-Statements, die nach verstrichener Zeit sortiert sind. *Abbildung 6* zeigt ein Beispiel.

Diese Tabelle ist recht interessant, da hier die am längsten dauernden Statements aufgeführt sind, unabhängig davon, ob es sich um einmal ausgeführte große Statements oder um kleine Statements handelt, die Tausende Male ausgeführt werden. Der Prozentsatz der gesamten DB-Zeit gibt den Teil der DB-Zeit an, für die das Statement verantwortlich ist. Nach Meinung des Autors sollte ein einziges Statement nicht mehr als zehn Prozent der gesamten DB-Zeit verbrauchen. Sollte mehr der Fall sein, müsste das Statement überprüft oder neu geschrieben werden.

In der als Beispiel genannten Datenbank scheint die DB-Zeit zwischen den Statements ausgewogen verteilt zu sein. Man sollte sich dessen bewusst sein, dass nicht alle Statements während der Snapshots erfasst werden. *Abbildung 7* zeigt eine andere Datenbank. Sie fokussiert auf Statements, die nach (physikalischen) Lesevorgängen sortiert sind, also Laufwerk-Lesevorgänge.

Zunächst einmal gibt es ein großes Statement, das zwanzig Prozent der gesamten Lesevorgänge durchführt (zweite Zeile). Dieses Statement ist höchstwahrscheinlich Teil des PL/SQL-Blocks der ersten Zeile. Der Fokus liegt also auf dem Statement in Zeile 2. Die Spalte für physische Lesevorgänge ist hier von Interesse: Darin werden die Zahl oder Blöcke von Laufwerk-Lesevorgängen angezeigt. Angenommen, die Datenbank hat eine Blockgröße von 8 KB, was heutzutage Standard ist: Das identifizierte Statement

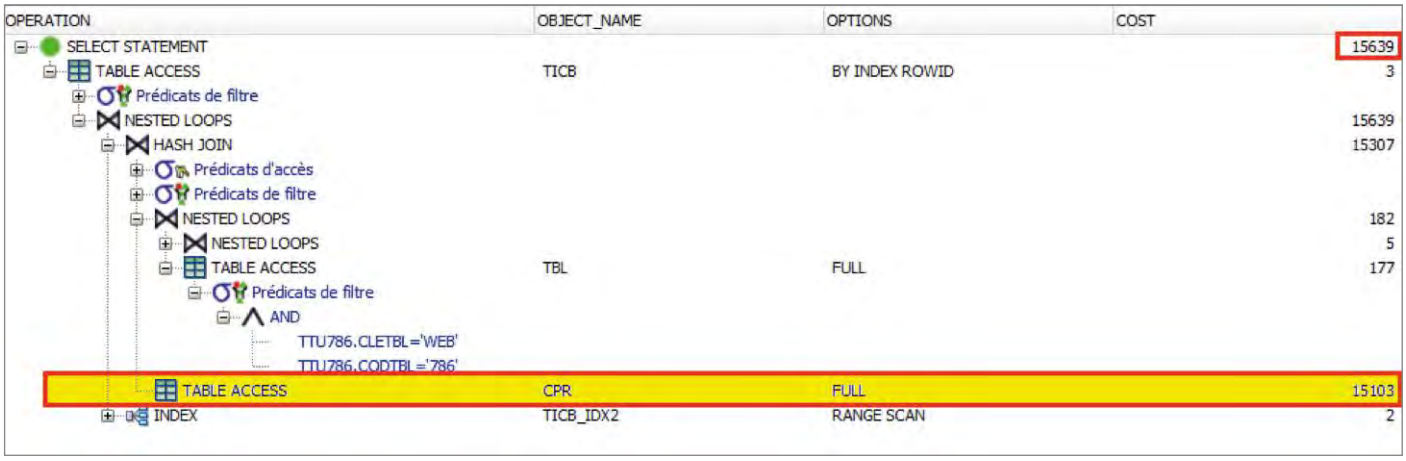


Abbildung 8: Ausführungsplan

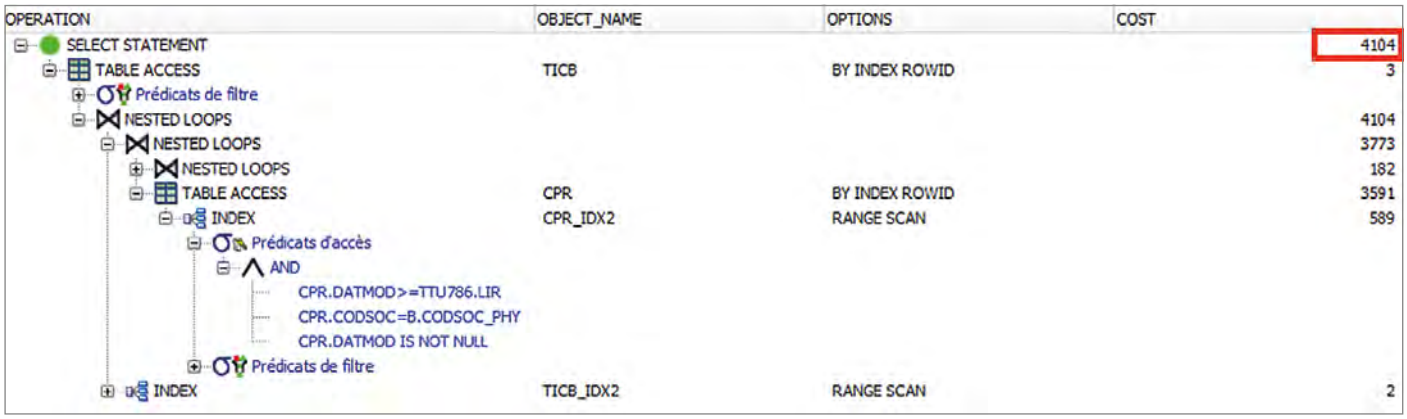


Abbildung 9: Neuer Ausführungsplan

```
select index_name, count(*) from dba_ind_columns
  where table_name = 'CPR'
      and column_name in ('CODSOC', 'DATMOD')
      group by index_name having count(*)>=2
```

Listing 5

liest hier mehr als 2 TB Daten während des Zeitraums aus.

Kurz gesagt: Einzelne SQL-Statements sollten im Normalfall nicht mehr als zehn Prozent der DB-Zeit benötigen. Man sollte zuerst Statements mit der längsten Ausführungsdauer untersuchen. Die Verbesserung der ersten zwei oder drei Statements in der sortierten Liste kann die DB-Zeit verringern und die Gesamtleistung der Datenbank erhöhen.

Indizierung von Statements untersuchen

DBAs haben in der Regel keinen Zugriff auf den in die Anwendung eingebetteten SQL-Code. Wahrscheinlich möchten sie auch nicht Hunderte SQL-Zeilen unter die Lupe nehmen. Manchmal besteht bereits die Verbesserung eines Statements lediglich in der neuen Indizierung zur Vermeidung verteilter Full-Table-Scans. *Listing 4* zeigt ein Beispiel aus der Praxis. Dieses Statement ist für zwanzig Prozent der DB-Zeit einer Datenbank verantwortlich.

Auf den ersten Blick ist dieses Statement nicht sehr komplex. Ein DBA kann den Ausführungsplan prüfen (etwa mit einem SQL-Entwickler), um zu sehen, wie dieses Statement von Oracle ausgeführt wird *(siehe Abbildung 8)*.

Die (geschätzten) Ausführungskosten dieses Statements liegen bei 15.639 zu lesenden Blöcken. Die Kosten entstehen vornehmlich durch einen einzigen Vorgang, der für beinahe alle erforderlichen Lesevorgänge verantwortlich ist: einen vollständigen Tabellen-Zugriff. Tatsächlich müssen hier alle Zeilen dieser Tabelle ausgelesen werden. Bei einem Blick auf das SQL-Statement ist zu erkennen, dass es in dieser Tabelle Filter für die Spalten „codsoc" und „datmod" gibt. Es besteht also wahrscheinlich kein Grund dafür, die gesamte Tabelle auszulesen. Vielleicht existiert kein Index auf diesen Spalten? Dies lässt sich wie in *Listing 5* überprüfen. Sollte es keinen Index geben, ist leicht verständlich, warum ein vollständiger Tabellenscan erforderlich ist.

Eine einfache Optimierung könnte in der Erstellung eines Index zu diesen beiden Spalten bestehen: „create index soc1.CPR_IDX2 on SOC1.CPR (codsoc,datmod) tablespace IDX;". *Abbildung 9* zeigt den neuen Ausführungsplan.

Die Zahl der zur Ausführung dieses Statements notwendige Blöcke ist jetzt vier Mal weniger; von zwanzig Prozent der DB-Zeit verbraucht dieses Statement nur noch fünf Prozent. Anders gesagt: Dieser einzige Index hat die DB-Zeit der Datenbank um fünfzehn Prozent verringert.

Anmerkung: Es ist zu berücksichtigen, dass ein Index einen erhöhten Ressourcen-Verbrauch beim Ändern der indizierten Spalten und mehr Redo-Daten verursacht. Außerdem kann dieser neue Index die Ausführung anderer Statements beeinflussen.

Kurz gesagt: Die Indizierung ist Teil der Arbeit des Datenbank-Administrators. Gute Indizes können die Notwendigkeit vollständiger Tabellenscans und IO-Waits verringern. Dies kann auch die Gesamtleistung der Datenbank erhöhen. Die DB-Zeit der Datenbank ist nach Erstellung neuer Indizes unbedingt zu prüfen.

Fazit

Für den Autor funktioniert dieser Ansatz bei neunzig Prozent der Performance-Probleme, die er beheben musste. Manchmal kann die Feineinstellung jedoch weitaus komplexer ausfallen. Um die Erfahrung und das Know-how zu verbessern, erstellt man einfach mehrere Leistungsberichte zu mehreren Datenbanken, selbst bei Datenbanken ohne Performance-Probleme.

Jérôme Dubar
jerome.dubar@dbi-services.com

PL/SQL & SQL – was nicht messbar ist, kann man nicht lenken

Jonas Gassenmeyer, syntegris information solutions GmbH

Beschwerden über langsame Applikationen sind in der Regel undankbar, wenn unklar ist, ob eher das Netzwerk oder die Datenbank selbst die Ursache für Performance-Probleme darstellen. Um solche Unklarheiten zu vermeiden, bietet die Oracle-Datenbank hervorragende Bordmittel, um Laufzeiten und Kenngrößen in PL/SQL und SQL zu beziffern. Der Artikel zeigt grundlegende Funktionsweisen des Hierarchical Profiler und des (Session-)Tracing auf, um den Einstieg ins SQL- und PL/SQL-Performance-Tuning zu finden.

Geschäftsanwendungen, die auf einer Oracle-Datenbank beruhen, sind in der Regel dadurch gekennzeichnet, dass mehr als nur ein simpler SQL*Plus-Client mit einfachen SQL-Statements zum Lesen und Schreiben von Daten involviert ist. PL/SQL-Packages können den Zugriff auf Daten kapseln (APIs); üblicherweise läuft die eigentliche Anwendung auf einem Application Server und darin laufende andere Programmiersprachen (PHP, Python etc.) können weitere rechenintensive Schritte enthalten. Schlussendlich muss eine Oberfläche an den Benutzer ausgeliefert werden, etwa über den Browser (HTTPS).

Unabhängig von der System-Architektur kann sich die Suche nach Ursachen für Performance-Probleme schwierig gestalten. Netzwerk-, Applikations- und Datenbank-Spezialisten sollten grundsätzlich eng miteinander arbeiten, um die genauen Ursachen möglichst schnell einzugrenzen. Datenbank-Entwicklern und -Administratoren stehen hervorragende Möglichkeiten zur Verfügung, um PL/SQL und SQL messbar zu machen und somit festzustellen, ob die Ursache in der Datenbank zu finden ist. Die Mess-Ergebnisse lassen sich dann objektiv zur Bewertung des Problems heranziehen.

Es gibt Unmengen von Werkzeugen, die mehr oder weniger ähnliche Messungen ermöglichen und Ergebnisse aufbereiten. Einige sind kostenpflichtig, andere wiederum können komplett quelloffen (Open Source) und kostenlos bezogen werden. Es kann allerdings sein, dass vor der Nutzung eine Installation erforderlich ist, was voraussetzt, dass Performance-Messungen bereits vor einem eigentlichen Problem antizipiert wurden.

Das Gute am Hierarchical Profiler (HPROF) und am Tracing ist, dass diese Tools mit der Installation einer jeden Oracle-Datenbank (lizenzunabhängig) bereitstehen. Angenommen, es gibt zwei Muster von Performance-Beschwerden:

- Immer, wenn ich xy mache, wird es langsam (häufig alle Sessions)
- Warum war xy langsam (häufig spezifische Session)?

Dazu sei verdeutlicht, dass nur wiederholbare Performance-Probleme (der erste Typ) mittels HPROF und Tracing analysiert werden können, weil eine Wiederholung der

Aktion notwendig ist. Das ist wahrscheinlich auch der Grund, weshalb „v$active_session_history" (beziehungsweise „dba_hist_active_sess_history") (ASH) so gerne zur Analyse herangezogen werden. Wie das „history" im Namen schon vermuten lässt, ermöglichen diese Objekte in einer Enterprise Edition (plus Diagnostics und Tuning Pack) eine lizenzkostenpflichtige Möglichkeit, auch vergangenheitsbezogene Analysen (der zweite Typ) durchzuführen.

Was aber kann man sich nun unter dem Hierarchical Profiler und Tracing vorstellen? In beiden Fällen wird bei der Durchführung einer Performance-Messung eine Datei mit der Dateiendung „.trc" (Trace File) auf dem Datenbank-Server erzeugt. Außerdem erfolgt die Messung über einen Start- und Stoppbefehl: Ist der involvierte Code (PL/SQL oder Session) identifiziert, so wird zu Beginn ein Marker gesetzt, dass die Messung beginnen soll. Dann wird der Code ausgeführt und abschließend ein weiteres Kommando zum Stoppen abgesetzt. Fragen, wie zum Beispiel vom Client-Rechner auf die Datei auf dem Server zugegriffen werden kann, wie dessen Inhalt zu verstehen beziehungsweise besser aufzubereiten ist und welche Voraussetzungen überhaupt erfüllt sein müssen, damit das Trace File auch tatsächlich geschrieben werden kann, sind im zweiten Teil des Artikels erklärt.

Messungen mit dem Hierarchical Profiler (PL/SQL)

Ist der zu messende PL/SQL-Code identifiziert, lässt sich der Hierarchical Profiler (HPROF) durch ein „SYS.DBMS"-Package („dbms_hprof") starten und stoppen. Dabei ist keine Re-Kompilierung des eigenen Codes notwendig. Dennoch sind einige vorbereitende Maßnahmen unabdinglich. *Listing 1* zeigt, dass zunächst über ein Directory-Objekt ein Zugriff auf ein Verzeichnis auf dem Datenbank-Server erfolgen muss. Außerdem benötigt das Schema natürlich entsprechende Rechte, das „dbms_hprof"-Package auszuführen. Dann kann es aber schon mit der Messung losgehen.

Listing 2 zeigt, dass das eben angelegte Directory als Parameter in der „start_profiling"-Prozedur mitgegeben wird. Zusätzlich vergeben wir einen Namen, den das spätere Trace File trägt. Vorsicht: Wenn ein File mit dem gleichen Namen existiert, wird Oracle diese Datei ohne Vorwarnung überschreiben. Alle darauffolgenden PL/SQL-Aufrufe werden gemessen, bis das Profiling durch die Prozedur „stop_profiling" beendet ist. Im Beispiel wird die Prozedur „lauf_forrest_lauf" aufgerufen, es wären jedoch auch mehrere Aufrufe möglich.

Obwohl die Doku *(siehe „https://goo.gl/imjkkM")* gute Erklärungen liefert, ist die rohe „.trc"-Datei gewöhnungsbedürftig. Man kann hier bereits von oben nach un-

```
create directory hp_dir as '/tmp/plshprof/results';
grant read, write on directory hp_dir to demo_user;
grant execute on dbms_hprof to demo_user;
```

Listing 1

```
dbms_hprof.start_profiling ('HP_DIR','forrest.trc);
lauf_forrest_lauf;
dbms_hprof.stop_profiling;
```

Listing 2

```
dbms_hprof.analyze(
   location    => 'HP_DIR'
, filename     => 'forrest.trc'
, run_comment  => 'Demo'
);
```

Listing 3

```
with prep as(
   select
       fi.module || '.' || fi.function n
    , pci.subtree_elapsed_time pci_subtree_elapsed_time
   , fi.subtree_elapsed_time fi_subtree_elapsed_time
   , pci.function_elapsed_time pci_function_elapsed_time
   , fi.function_elapsed_time fi_function_elapsed_time
   , pci.calls --wie viele Aufrufe der Routine
   , fi.sql_text --neu in 12.2
   from dbmshp_function_info fi
   left join dbmshp_parent_child_info pci on fi.symbolid = pci.childsymid
and fi.runid = pci.runid
   )
select
   --sorgt fuer Einrueckungen bei Subroutinenaufrufen
       rpad(' ',level, '.') ||level||') '|| n aufruf
   , calls
   --Microsekunden -> Sekunden
   , pci_subtree_elapsed_time/1000000 sek_inkl_subprog
   , fi_subtree_elapsed_time/1000000 sek_inkl_subprog_summiert
   , pci_function_elapsed_time/1000000 sek_atomar
   , fi_function_elapsed_time/1000000 sek_atomar_summiert
   , sql_text
from prep
--hier wird die Hierarchie aufbereitet
connect by parentsymid = prior symbolid
start with parentsymid is null
```

Listing 4

```
select sql_id , sql_text from dbmshp_function_info;
```

Listing 5

ten gehen und Zeiten („P#X“) ermitteln, die einzelne Subroutinen benötigt haben. Die eigentliche Stärke wird allerdings erst erkennbar, wenn die Rohdaten aufbereitet sind. Der Name „hierarchical“ kommt nicht von ungefähr, denn in den Messdaten ist auch festgehalten, wie häufig und vor allem von welchen darüber liegenden Routinen eine Subroutine aufgerufen wurde. Eine solche hierarchische Darstellung vereinfacht die Suche nach dem „dicken Fisch“ – also jenem Unterprogramm, das einen Zeitfresser darstellt.

Messdaten des HPROF auswerten

Um eine solche Voraggregierung und Hierarchisierung der Messdaten zu vereinfachen, kann der Nutzer das Kommandozeilen-Tool „plshprof” verwenden. Dieses ist in jedem Full-Client oder im Datenbank-Server (Datenbank-Installation) vorhanden. Damit wird eine HTML-Seite erzeugt, die in einem vordefinierten Bericht Auskunft über die Laufzeiten gibt *(weitere Infos siehe „https://goo.gl/7k69Bx“)*.

Das setzt nicht nur voraus, dass man mit den vordefinierten Berichten zufrieden ist, sondern auch, dass man direkten Zugriff auf die „.trc“-Datei hat, die ja auf dem Datenbank-Server abgelegt wurde. Ist das nicht der Fall, bleibt man am besten in der Session und ruft eine weitere Prozedur im bereits bekannten Package auf *(siehe Listing 3)*. Das bekannte Directory und die in „start_profiling” definierte „.trc“-Datei sind erneut als Parameter für den Aufruf erforderlich.

Auch dieser Aufruf erfordert eine Vorbereitung, denn es muss ein Skript ausgeführt werden („$ORACLE_HOME/rdbms/admin/dbmshptab.sql“). Im Hintergrund passiert nicht wirklich etwas Sensationelles: Es werden drei Tabellen (und eine Sequenz) im Schema erstellt, die eine Analyse der Messwerte auf SQL-Basis ermöglichen. Hier gab es eine nennenswerte Neuerung in 18c. Zukünftig bleiben dem Entwickler lästige Suchen in Installationsordnern nach dem Skript erspart: Das PL/SQL-Package stellt einen Prozeduraufruf („dbms_hprof.create_tables“) bereit, mit dem das Setup erzeugt werden kann. Vor allem sollte man einen genauen Blick auf die Tabellen „dbmshp_function_info” und „dbmshp_parent_child_info” sowie die Spalten „subtree_elapsed_time” und „function_elapsed_time” werfen. Hier sind die Laufzeiten für eine Routine in Millisekunden aufgeführt.

Wie der Name „Parent Child“ schon erwarten lässt, enthält „dbmshp_parent_child_info” die Aufrufhierarchie. Mit einer geschickten „connect by“-Klausel in SQL *(siehe Listing 4)* lässt sich dann eine Darstellung ermitteln, die zum einen eine isolierte Betrachtung erlaubt, wie viele Zeit in einer Subroutine verbracht wurde („function_elapsed_time”), und zum anderen zeigt, wie viel Zeit inklusive aller bis dahin ausgeführten Routinen („subtree_elapsed_time”) benötigt wurde.

Es soll noch betont sein, dass ab der Datenbank-Version 12.2 auch die „sql_id” eines innerhalb der PL/SQL-Routine ausgeführten SQL-Statements mit angezeigt werden kann *(siehe Listing 5)*. Das leitet dann auch wunderbar zum Session-Tracing über. Denn was bringt dem Oracle-Spezialisten die Info, dass ein SQL langsam läuft, ohne weitere Details darüber zu erhalten, wofür sich der Optimizer bei der Statement-Ausführung entschieden hat? Solche im HPROF fehlenden Details lassen sich durch Tracing ermitteln.

Messungen mittels Session-Tracing (SQL)

Genau wie HPROF wird das Tracing (auch als „SQL Trace“ oder „event 10046“ bekannt) über ein „SYS.DBMS“-Package („dbms_monitor”) ein- und ausgeschaltet und auch hier wird eine „.trc“-Datei auf dem Server abgelegt. Das waren aber auch schon die Gemeinsamkeiten. Nicht nur, dass bereits ein Standard-Directory („select value from v$diag_info where name= ‘Default Trace File’”) existiert – im Gegensatz zum HPROF wird beim Tracing eine ganze Session (nicht nur PL/SQL) gemessen.

Da unter Umständen sehr viele Aktivitäten/Befehlsanweisungen in einer Session ablaufen können, ist es ratsam, vor der Aktivierung einen genauen Plan zu skizzieren, was und wie lange etwas zu sehen sein soll, um dem Performance-Problem auf den Zahn zu fühlen. Was hier in einem schnellen Satz abgehandelt ist, stellt sich in der Praxis häufig als sehr viel

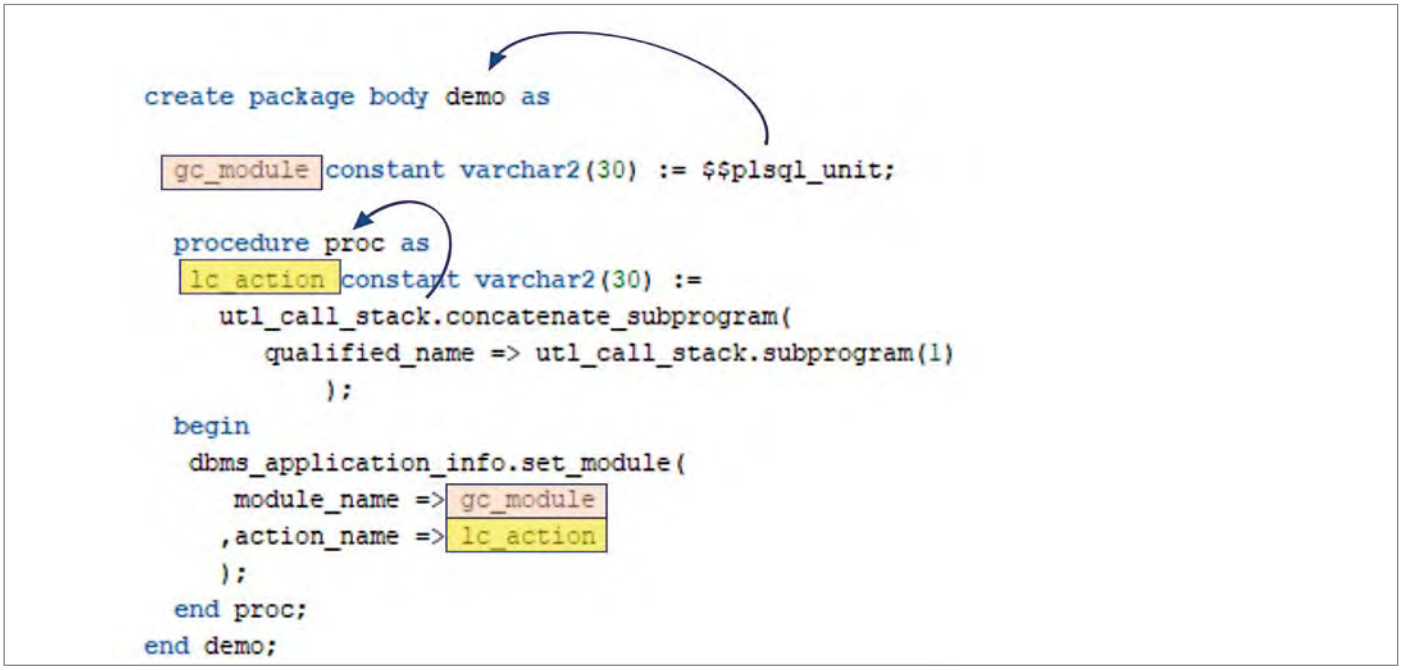


Abbildung 1: So könnte eine Code-Instrumentalisierung aussehen

anspruchsvoller dar. Irgendwie muss erreicht werden, dass die Session identifiziert werden kann, sobald sie auf einer Instanz aktiv wird. Deshalb passt für das Einschalten des Tracing der Spruch „Viele Wege führen nach Rom" wahrscheinlich am besten; denn „dbms_monitor" erlaubt es zum einen, in der selbst aktivierten Session (eine einzige Verbindung zur Datenbank involviert) das Tracing jederzeit zu aktivieren oder zu deaktivieren. Das setzt allerdings voraus, dass der Code irgendwie abgewandelt und eventuell neu kompiliert werden kann, was nicht immer der Fall sein wird. Deswegen können zum anderen verschiedene Kriterien (Modul und/oder Action und/oder Client Identifier oder die Session-ID selbst) zur Identifikation der Session herangezogen werden. Das ist dann gut, wenn man in einer zweiten, unabhängigen Session ein Tracing für die eigentliche Applikationssession steuern möchte. Am besten klappt das, wenn die Anwendung vernünftig instrumentalisiert ist. *Abbildung 1* zeigt, was darunter zu verstehen ist.

Das Package „dbms_application_info" in geschickter Kombination mit „utl_call_stack" (Version 12) und dem Compilerflag „$$plsql_unit" kann wahre Wunder bewirken und lässt sich fast universell in ihren Code kopieren. Unter der Voraussetzung, dass die PL/SQL-Routinen sprechend benannt sind und wiedergeben, was der Code gerade tut, enthält ein unabhängiger Betrachter diese Details ebenfalls. *Listing 6* zeigt, dass „v$session" fast immer ein guter Startpunkt ist, um sich diese Informationen zusammenzureimen; das impliziert auch, dass mehrfache Testausführungen notwendig sein können, bis mit der eigentlichen Messung begonnen wird.

Wer Vollzugriff auf seinen Code hat, sollte am besten schnell mal nachschauen, ob dort schon Session-Informationen an die Oracle-Datenbank herangetragen werden (Code-Instrumentierung). Wenn nicht, sollte man das am besten schleunigst nachpflegen. Wer allerdings nicht die Möglichkeit hat, den problematischen Code zu ändern und so zu instrumentalisieren, dass eine Session fürs Tracing einwandfrei identifiziert werden kann, für den ist Kreativität gefragt. David Kurtz zeigt zum Beispiel unter *„https://goo.gl/*

```
select module, action, client_info from v$session;

MODULE               ACTION          CLIENT_INFO
-----------------    ------------    --------------
JDBC Thin Client     do_something    sqlclient_v1
```

Listing 6

```
dbms_monitor.serv_mod_act_trace_enable(
  service_name=>'orcl',
  module_name=>'api',
, action_name=>'running');
```

Listing 7

```
  select payload
  from v$diag_trace_file_contents
  where trace_filename = 'X'
```

Listing 8

```
tkprof session.trc out.txt pdbtrace=demo/pwd@orcl
```

Listing 9

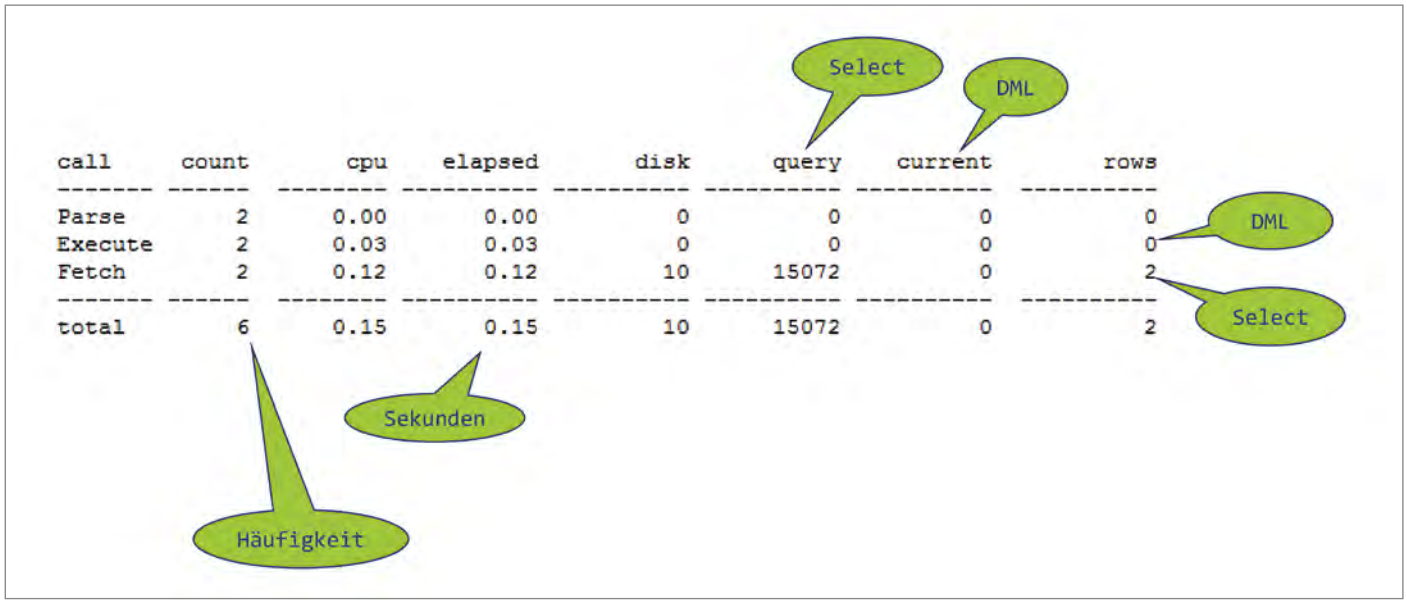


Abbildung 2: Tabellen-Darstellung im „tkprof"-Output

QAW47D" (Folie 27) einen Weg mittels Trigger, um PeopleSoft zu „markieren".

Nachdem einige Möglichkeiten aufgezeigt wurden, wie man Sessions identifizieren kann, zum Abschluss noch, wie die eigentliche Messung durchgeführt wird. *Listing 7* zeigt, wie die Messung in einer von der Applikation unabhängigen Session gestartet werden kann.

Sobald die eigentliche Session der Applikation mit dem entsprechenden Modul „api" und der Action „running" aktiv wird, tut Oracle sein Übriges. Hat man genug Messdaten geschrieben, wird die Prozedur „serv_mod_act_trace_disable" aufgerufen, um die Messung zu stoppen. Oracle hat dann ein „.trc"-File im vordefinierten Trace-Verzeichnis (siehe oben) erstellt. Da Oracle standardmäßig einen Namen für die Datei vergibt, kann es bei dessen Suche Schwierigkeiten geben. Wenn ein „alter session"-Befehl möglich ist, kann man mit „set tracefile_identifier" ein Suffix vergeben, um das Auffinden zu erleichtern. Ab Oracle Version 12.2 wird auch kein Zugriff mehr auf den Server benötigt. Stattdessen lässt sich der Inhalt der „.trc"-Dateien über den Client abfragen *(siehe Listing 8)*.

Messdaten des Tracing auswerten

Für eine Aggregation der Messdaten stellt Oracle ebenfalls ein Kommandozeilen-Tool bereit: „tkprof" sollte definitiv genutzt werden, denn ein Trace File im Rohformat kann zum Zeitfresser werden. Vor 18c benötigt „tkprof" eine lokale „.trc"-Datei; in zukünftigen Versionen kann man mit dem „pdbtrace"-Parameter dafür sorgen, dass sich der Aggregator die Rohdaten selbst über eine angegebene Datenbank-Verbindung ermittelt *(siehe Listing 9)*. Voraussetzung ist wieder, dass man den Namen (im Beispiel „session.trc") der Trace-Datei kennt.

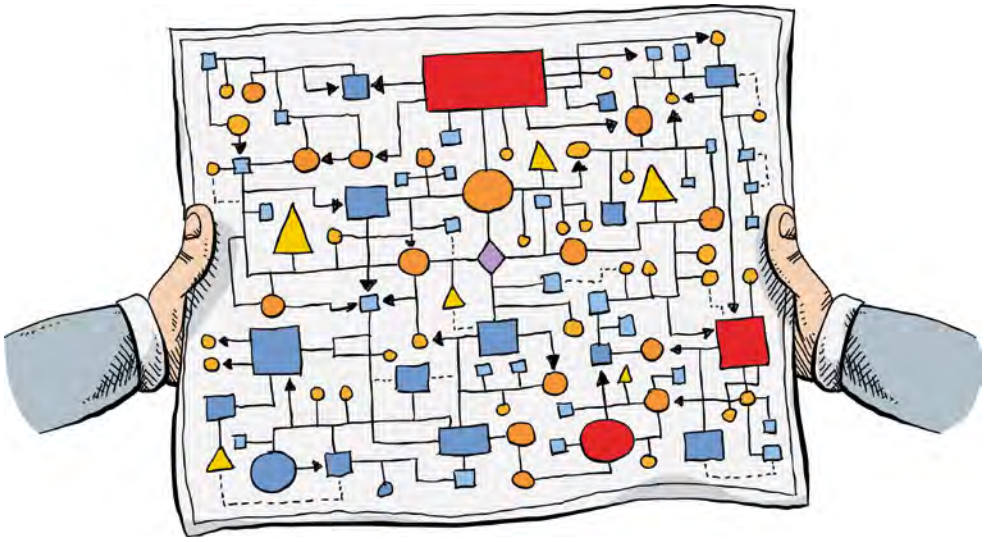
Weitere Parameter für „tkprof" erläutert die Dokumentation genauer *(siehe „https://goo.gl/3QXe5s")*. Typischerweise fällt in der Output-Datei auf, dass dort Tabellen enthalten sind, die für ein ausgeführtes Statement die Randbedingungen zusammenfassen; *Abbildung 2* zeigt eine solche Tabelle.

Die Laufzeit ist in drei Schritte unterteilt: Es wird dargestellt, welche Ressourcen (I/O, CPU etc.) für den semantischen und syntaktischen Check („Parse"), das Ausführen („Execute") und die letztendliche Datenermittlung („Fetch") verwendet wurden. Nennenswert ist noch, dass beim Blick in die Spalten zwischen DML- und Select-Ausführungen unterschieden werden muss. Eine vollständige Analyse würde den Rahmen des Artikels sprengen, abschließend sei jedoch noch erwähnt, dass sich die Untersuchung nicht ausschließlich auf die Zeit der Ausführung konzentrieren sollte. Möglich ist auch, dass ein Statement auf die Ausführung warten musste. Solche Wait-Events lassen sich im Tracing ebenfalls ermitteln, da die „dbms_monitor"-Prozeduren und „tkprof" Parameter bereitstellen. Es empfiehlt sich, zu Beginn einfach ein wenig mit diesen und anderen Parametern herumzuspielen und die Unterschiede in den jeweils erstellten Output-Files zu vergleichen. Das vermittelt eine genaue Idee davon, was aus den Messdaten alles herausgelesen werden kann – es ist eine ganze Menge.

Fazit

Der Artikel vermittelt einen Eindruck, wie PL/SQL und SQL mit Hierarchical Profilers und Session-Tracing messbar gemacht werden können. Die Code-Beispiele haben gezeigt, wie man Messungen durchführt, und es wurden Ansätze der Interpretation erklärt. Jederzeit lässt sich das Thema vertiefen. Dazu kann man Anfragen, Anregungen oder Feedback jederzeit per Mail an den Autor schicken.

Jonas Gassenmeyer
jonas.gassenmeyer@syntegris.de



Warum kompliziert, wenn es auch einfach geht: (SQL-) Funktionen in der Oracle-Datenbank

Ulrike Schwinn, ORACLE Deutschland B.V. & Co. KG

Programmiersprachen kommen und gehen. Nur wenige Sprachen, die heute in Gebrauch sind, haben Wurzeln, die wie bei SQL über Jahrzehnte zurückreichen. Der SQL-Umfang wächst mit jedem Oracle-Datenbank-Release. Die Neuerungen erweitern dabei nicht nur die Funktionalität, sondern helfen auch, die Programmierung zu erleichtern und dabei sogar die Performance zu erhöhen.

Viele Anfragen – darunter auch komplexe – lassen sich mittlerweile mit einfachen SQL-Mitteln lösen. Man benötigt keine anderen Programmiersprachen oder gar zusätzlichen Code. Ein wichtiges Ziel muss dabei immer sein, dass die Performance nicht unter der zunehmenden Funktionalität leidet, sondern ganz im Gegenteil die Abfragen performant erfolgen. Grund genug, in den folgenden Abschnitten ein paar interessante und vielleicht auch noch unbekannte SQL-Funktionen aus unterschiedlichen Datenbank-Releases vorzustellen, die genau diese Ziele verfolgen.

SQL-Konstrukte, die jeder kennen sollte

Ein einfaches, häufig verwendetes Abfragekonstrukt, mit dem jeder Datenbank-Entwickler irgendwann konfrontiert wird, sind die Top-N-Abfragen und das Blättern in Applikationen. Lange Zeit waren das Arbeiten mit „ROWNUM“ und die geschickte Nutzung einer Subquery dazu erforderlich, wie das Beispiel in *Listing 1* mit dem altbekannten Demoschema „SH“ und der Tabelle „PRODUCTS“ zeigt. Die Aufgabe besteht darin, die fünf kostengünstigsten Produkte aufzulisten.

```
select prod_id, prod_list_price from
   (select * from sh.products order by prod_list_price)
   where rownum <= 5;
```

Listing 1

```
select prod_id, prod_list_price
from (select prod_id, prod_list_price, rownum AS rnum
   from (SELECT prod_id, prod_List_price
       FROM products
       ORDER BY prod_list_price)
   where rownum <= 10)
where rnum >= 5;
```

Listing 2

```
select prod_id, prod_list_price
from sh.products order by prod_list_price fetch first 5 rows only;
```

Listing 3

```
select prod_id, prod_list_price
from sh.products order by prod_list_price offset 5 rows fetch next 5
rows only;
```

Listing 4

```
select e.ename, e.sal, avg_sal from emp e, (select avg(sal) avg_sal
from emp e1 where e.deptno=e1.deptno);
*
ERROR at line 1:
ORA-00904: "E"."DEPTNO": invalid identifier
```

Listing 5

```
select e.ename, e.sal, avg_sal from scott.emp e,
lateral (select avg(sal) avg_sal from scott.emp e1 where e.deptno=e1.
deptno)
ENAME             SAL     AVG_SAL
---------- ----------  ----------
JAMES              950  1566.66667
TURNER            1500  1566.66667
BLAKE             2850  1566.66667
MARTIN            1250  1566.66667
WARD              1250  1566.66667
ALLEN             1600  1566.66667
FORD              3000        2175
ADAMS             1100        2175
...
```

Listing 6

```
SQL> select * from emp;
  EMPNO ENAME      JOB        MGR     HIREDATE      SAL     COMM    DEPTNO
------- ---------- ---------  ------ --------- ------- ------- --------
   7369 SMITH      CLERK      7902   17-DEC-80     800                20
   7499 ALLEN      SALESMAN   7698   20-FEB-81     1600     300       30

14 rows selected.
SQL> set linesize window
SQL> set feedback on sql_id
SQL> r
  1* select * from emp

  EMPNO ENAME      JOB        MGR     HIREDATE      SAL     COMM    DEPTNO
------- ---------- ---------  ------ --------- ------- ------- --------
   7369 SMITH      CLERK      7902   17-DEC-80     800                20
   7499 ALLEN      SALESMAN   7698   20-FEB-81     1600     300       30
…
14 rows selected.

SQL_ID: a2dk8bdn0ujx7
```

Listing 7

Nun soll um fünf Einträge weitergeblättert werden. Auch das lässt sich mit einer Erweiterung der Abfrage leicht bewerkstelligen. Setzt man zusätzlich noch den Result Cache für die sich wiederholenden Abfragen ein, kann man schnell einen Performance-Vorteil gewinnen *(siehe Listing 2)*.

Die Verwendung des Result Cache lässt sich mit dem Hint „RESULT_CACHE“ oder über (Session- beziehungsweise System-) Parameter einschalten und ist eine einfache und sehr effiziente Methode, das Ergebnis von komplexen Abfragen in einem speziellen Cache vorzuhalten. Sobald sich das Ergebnis ändert, wird der Cache automatisch invalidiert. Daher sollte man überprüfen, für welche Abfragen die Einstellung sinnvoll ist, und nicht unbedingt nach dem Gießkannen-Prinzip den Cache für alle Abfragen reservieren. Diese Art von Abfragen lässt sich ab der Version 12c viel übersichtlicher mit der sogenannten „Row Limiting“- Klausel programmieren. Für die fünf kostengünstigsten Produkte ergibt sich dann der SQL-Code in *Listing 3*. Es werden einfach die Schlüsselworte „fetch first <Anzahl Zeilen> rows only“ angefügt. Für das Blättern der nächsten fünf Zeilen ergibt sich die erweiterte Syntax mit „offset“ *(siehe Listing 4)*.

Eine weitere interessante Funktionalität, die das Programmieren mit SQL vereinfacht, ist die Unterstützung der „LATERAL“-Klausel ab 12c – übrigens eine Funktion aus dem „SQL:2016“-Standard. Es geht darum, Tabellen außerhalb einer Inline-View (Korrelation) zu referenzieren. Ein einfaches Beispiel in 11g R2 zeigt die Problematik: Es sollen die Angestellten, ihre Gehälter und die Durchschnittsgehälter der zugehörigen Abteilung ausgegeben werden. Ein erster Versuch scheitert mit einem Fehler *(siehe Listing 5)*. Diese Aufgabe lässt sich allerdings lösen. Mit 12c und der neuen „LATERAL“-Klausel ist dies ganz einfach mit einer einzigen SQL-Abfrage möglich *(siehe Listing 6)*.

Das Beispiel zeigt einen Join mit zwei Operanden. Der zweite Operand ist eine Lateral-Inline-View, gekennzeichnet durch das Schlüsselwort „LATERAL“, die die Tabelle „SCOTT.emp e“ in der „WHERE-Klausel“ nutzen kann, ohne einen Fehler zu erzeugen.

Monitoring von Ausführungsplänen

Es gibt in der Oracle-Datenbank viele Möglichkeiten (wie Views und Reports), die dabei helfen, die Ausführungspläne der Statements zu monitoren. Lästig ist

dabei häufig die Notwendigkeit, die „SQL_ID“ herauszufinden. Mit SQL Developer oder der neuesten „SQL*Plus“-Version ist dies nun einfach möglich. Mit „SET FEEDBACK ON SQL_ID“ beispielsweise lässt sich in SQL*Plus-18c ganz einfach die „SQL_ID“ des abgelaufenen Statements anzeigen. Kein langes Suchen im „V$SQL“ oder anderen Views ist erforderlich. Auch bei der Ausgabe-Formatierung in SQL*Plus gibt es weitere Erleichterungen: Die Ausgabe lässt sich nun wie bei der grafischen Variante an die Bildschirmgröße anpassen. Dazu muss nur der Wert für „SET LINESIZE“ auf den neuen Wert „WINDOW“ gesetzt werden *(siehe Listing 7)*.

Für die Ausführungspläne der Abfragen gibt es das nützliche PL/SQL-Package „DBMS_XPLAN“, um Berichte über die Durchführung von Abfragen zu erstellen. Einfacher geht es mit dem SQL Developer, der nicht nur die „SQL_ID“ anzeigen kann, sondern auch den Ausführungsplan im eigenen Viewer ausgibt beziehungsweise die Unterschiede von zwei Ausführungsplänen hervorheben kann. Auch Informationen zur „AUTOTRACE“-Funktion sind möglich, sofern die Privilegien diesen Zugriff gestatten. Dabei werden sogar sogenannte „Hotspots“ angezeigt, die man genauer analysieren sollte. In der (aktuellen) Version 18.2 (Stand August 2018) des SQL Developer lassen sich die Ausführungspläne sogar in der „DBMS_XPLAN“-Sichtweise anzeigen *(siehe Abbildung 1)*.

Übrigens ist es in 12.2 auch einfacher, mögliche SQL-Statements in PL/SQL-Pro-

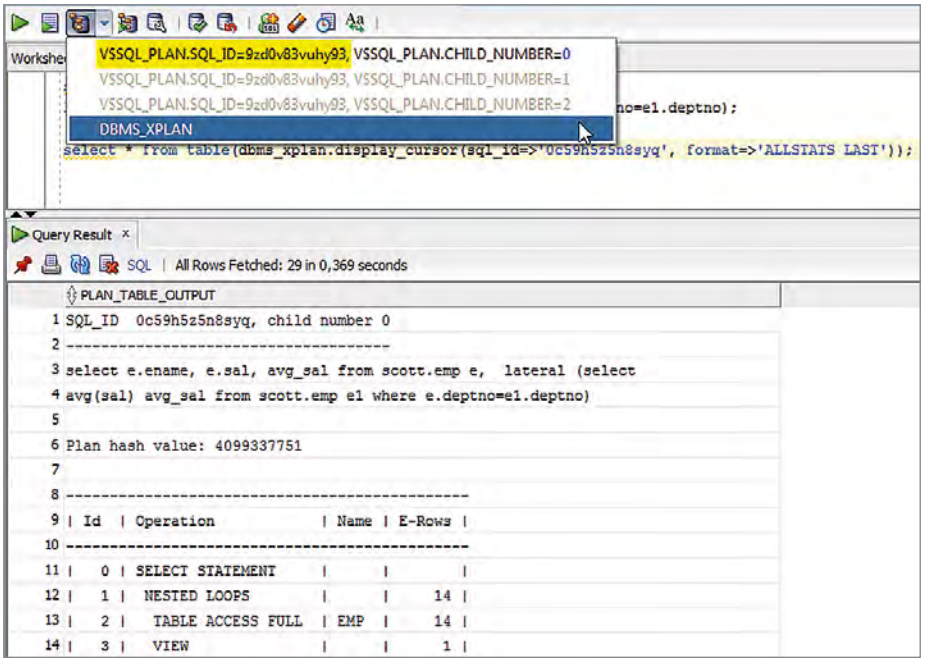


Abbildung 1: „DBMS_XPLAN“-Ausführungsplan im SQL Developer 18.2

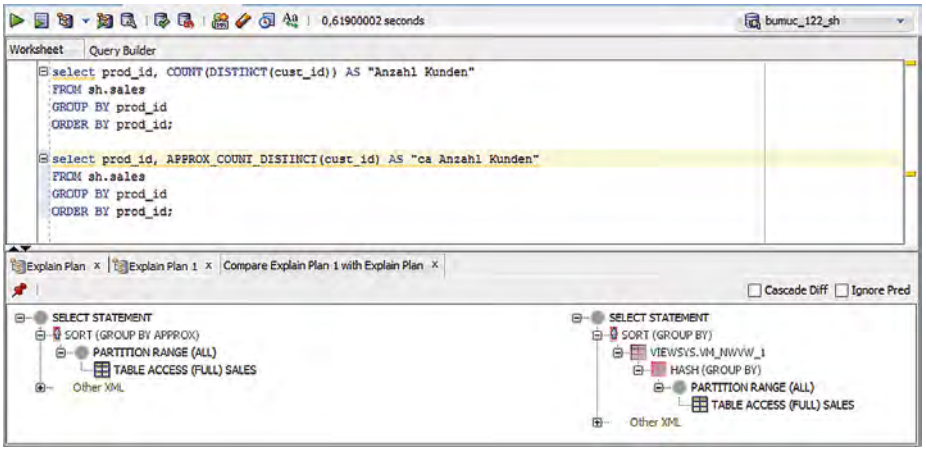


Abbildung 2: Vergleich zweier Ausführungspläne im SQL Developer

```
SQL> CREATE VIEW daily_prod_count AS
  2   SELECT t.calendar_year year,
  3       t.calendar_month_number month,
  4       t.day_number_in_month day,
  5       APPROX_COUNT_DISTINCT_DETAIL(s.prod_id) daily_detail
  6   FROM sh.times t, sh.sales s
  7   WHERE t.time_id = s.time_id
  8   GROUP BY t.calendar_year, t.calendar_month_number, t.day_number_in_month;

View created.

SQL> select * from daily_prod_count where rownum=1;

      YEAR      MONTH        DAY
---------- ---------- ----------
DAILY_DETAIL
--------------------------------------------------------------------------------
      2000          1          1
0D0C04CC0006000000000000000600000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

Listing 8

grammen aufzuspüren. Der PL/SQL Hierachical Profiler, der immer noch eines der wichtigsten Werkzeuge darstellt, um Bottlenecks zu finden und Performance-Tuning in PL/SQL durchzuführen, zeigt jetzt automatisch die zugehörigen SQL-IDs und das Statements (also SQL-Text) an. So ist es einfach möglich, den zugehörigen SQL-Code zu analysieren.

Eine ähnliche Idee steckt auch hinter der PL/Scope-Erweiterung. Diese sammelt während der Kompilierung Informationen über die Verwendung der Identifier und speichert diese automatisch in Data-Dictionary-Tabellen. Aufgezeichnet werden Typen, Namen, Nutzung und Vorkommen der Identifier im PL/SQL-Code. So ist deren einfache Analyse im Code möglich. Der Parameter „PLSCOPE_SETTINGS“ aktiviert die Nutzung und stellt sie ein. In 12.2 lassen sich damit auch SQL-Statements mitloggen. Dazu werden neue Werte wie „IDENTIFIERS:SQL, STATEMENTS:ALL“, „IDENTIFIERS:ALL, STATEMENTS:ALL“ und „IDENTIFIERS:PLSQL, STATEMENTS:NONE“ bereitgestellt. Die neue View „USER_STATEMENTS“ listet dabei die SQL-Statements auf.

SQL für analytische Anfragen

Standard-SQL kann sicherlich viele oder auch die meisten Datenbank-Anfragen beantworten. Wenn es allerdings um Fragen wie zum Beispiel „Wie hoch ist die laufende Summe der Mitarbeitergehälter?“ geht, ist dies mit Basis-SQL-Syntax nicht einfach zu lösen, da Berechnungen Zeile für Zeile durchgeführt werden müssen. Analytische Funktionen (auch „Window-Funktionen“ genannt) fügen SQL-Erweiterungen hinzu, die solche Operationen nicht nur schneller ablaufen lassen, sondern die auch einfacher zu programmieren sind. Im Internet stehen viele Beispiele dazu, etwa im Oracle Magazine oder in den Live-SQL-Tutorials.

In diesem Zusammenhang ist eine neue Kategorie von Funktionen eingeführt worden – die sogenannten „Approximate-Funktionen“, die besonders dann sinnvoll sind, wenn keine exakten Werte benötigt werden und Näherungen akzeptabel sind. Schon seit Release 12.1 gibt es die Funktion „APRROX_COUNT_DISTINCT“, die eine Alternative zu „COUNT(DISTINCT)“ darstellt. Sie verarbeitet große Datenmengen deutlich schneller, wie man bei der Analyse der beiden Ausführungspläne in *Abbildung 2* erkennen kann – mit vernachlässigbarer Abweichung vom exakten Ergebnis.

Weitere Funktionen wie zum Beispiel „APPROX_COUNT_DISTINCT_DETAIL“, „APPROX_COUNT_DISTINCT_AGG“ und „TO_APPROX_COUNT_DISTINCT“ helfen seit 12.2 dabei, einmalig Ressourcen-intensive Näherungszahlberechnungen durchzuführen, die resultierenden Details zu speichern und anschließend effiziente Aggregationen und Abfragen zu diesen Details durchzuführen. „APPROX_COUNT_DISTINCT_DETAIL(expr)“ ist dabei eine Art Helper-Funktion, die Näherungswerte für „expr“ berechnet und einen „BLOB“-Wert, das sogenannte „Detail“, in einem speziellen Format zurückliefert.

Dies können hochgranulare Details sein, wie etwa demografische Zählungen der Städte oder tägliche Verkaufszahlen, die in einer resultierenden Detail-Tabelle oder (materialisierten) View gespeichert werden. Das Beispiel in *Listing 8* berechnet die Produkt-Verkaufszahlen pro Tag mit „APPROX_COUNT_DISTINCT_DETAIL“.

Die von „APPROX_COUNT_DISTINCT_DETAIL“ zurückgegebenen Details können dann als Input für die Funktion „APPROX_COUNT_DISTINCT_AGG“ dienen, um Aggregationen der Details durchzuführen. Dies können Details geringerer Granularität sein, wie etwa demografische Zählungen oder monatliche Verkaufszahlen. Die Funktion „TO_APPROX_COUNT_DISTINCT“ wandelt dann die Ergebnisse aus „APPROX_COUNT_DISTINCT_DETAIL“ oder „APPROX_COUNT_DISTNCT_AGG“ in ein lesbares Format um *(siehe Listing 9)*.

Korrespondierende Funktionen stehen übrigens auch im Zusammenhang mit „PERCENTILE“ und „MEDIAN“ zur Verfügung, also „APPROX_PERCENTILE_DETAIL“, „APPROX_PERCENTILE_DETAIL” und „TO_APPROX_PERCENTILE”. Um den Einsatz auch ohne Änderung des bestehenden Codes zu ermöglichen, sind mit 12.2 zusätzliche Initialisierungsparameter wie „APPROX_FOR_COUNT_DISTINCT“ und „APPROX_FOR_PERCENTILE“ eingeführt worden. Sind diese aktiviert, wird die exakte Funktion einfach durch die korrespondierende „Approximate“-Funktion ersetzt.

Die Idee, weitere Näherungsfunktionen zur Verfügung zu stellen, um die Performance bei Berechnungen speziell im Massendaten-Umfeld zu erhöhen, wurde auch in 18c weiterverfolgt. So gibt es die neuen „Window“-Funktionen „APPROX_RANK“, „APPROX_COUNT“ und „APPROX_SUM“. „APPROX_RANK“ gibt dabei den Rang in einer Gruppe von Werten zurück. Hier ist allerdings eine bestimmte Syntax-Vorgabe erforderlich. Um die Funktionsweise kurz zu demonstrieren, nehmen wir die Tabelle „EMP“ als Beispiel. Es werden jeweils pro Abteilung die Top-10-Jobs ausgegeben. Für jeden Job sind das Gesamtgehalt und das Ranking angegeben *(siehe Listing 10)*.

```
SQL> SELECT year,
  TO_APPROX_COUNT_DISTINCT(APPROX_COUNT_DISTINCT_AGG(daily_detail))
yearly_detail
  FROM daily_prod_count
  GROUP BY year order by year;

      YEAR YEARLY_DETAIL
---------- -------------
      1998            60
      1999            72
      2000            72
      2001            71
```

Listing 9

```
SQL> SELECT deptno, job, APPROX_SUM(sal),
  APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal) DESC) rk
  FROM emp
  GROUP BY deptno, job
  HAVING APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal)
DESC) <= 10;

    DEPTNO       JOB APPROX_SUM(SAL)          RK
---------- --------- --------------- ----------
        10     CLERK            1300          3
        10   MANAGER            2450          2
        10 PRESIDENT            5000          1
        20     CLERK            1900          3
        20   MANAGER            2975          2
        20   ANALYST            6000          1
        30     CLERK             950          3
        30   MANAGER            2850          2
        30  SALESMAN            5600          1
```

Listing 10

```
CREATE PRIVATE TEMPORARY TABLE ORA$PTT_sales_ptt_session
  (time_id       DATE,
  amount_sold    NUMBER(10,2))
  ON COMMIT PRESERVE DEFINITION;
```

Listing 11

Weitere Neuigkeiten in Oracle Database 18c

Lange ersehnt und endlich realisiert ist nun die Möglichkeit, mit privaten temporären Tabellen zu arbeiten. Um eine solche Tabelle (PTT) zu erzeugen, ist eine gewisse Namenskonvention einzuhalten: Der Name beginnt immer mit einem Präfix (Default „ORA$PTT“), der über den Initialisierungsparameter „PRIVATE_TEMP_TABLE_PREFIX“ festgelegt ist und geändert werden kann. Wie bei der globalen temporären Tabelle gibt es auch hier eine Transaktions- und eine Session-spezifische Variante – allerdings mit einigen Unterschieden. Die Transaktions-spezifische Variante, die den Default darstellt oder mit den Schlüsselworten „DROP DEFINITION“ erzeugt wird, speichert Daten nur bis zum Ende einer Transaktion. Die Session-spezifische Variante *(siehe Listing 11)* wird mit „PRESERVE DEFINITION“ angelegt. In beiden Fällen erfolgt dabei kein implizites Commit.

Externe Tabellen (auch External Tables) sind aus der Oracle-Datenbank nicht mehr wegzudenken. Sie sind seit jeher ein geeignetes Mittel, um auf Daten einfach zuzugreifen, die in Flat-Files außerhalb der Datenbank gespeichert sind. Erforderlich sind dazu ein logisches Directory zur Lokalisierung der Daten und ein Access-Treiber wie „ORACLE_LOADER“, „ORACLE_DATAPUMP“, „ORACLE_HIVE“ oder „ORACLE_HDFS“.

In der Datenbank 18c gibt es einige interessante Neuerungen dazu; beispielsweise besteht die Möglichkeit, Inline External Tables zu nutzen. Es wird dabei keine externe Tabelle als persistentes Objekt im Data Dictionary angelegt, sondern einfach das „SELECT“-Kommando um eine neue „inline_external_table“-Syntax in der „FROM“-Klausel erweitert. Dies vereinfacht den Zu-

```
SQL> select * from sales_tab where sales_id=961;

  SALES_ID    PROD_ID    CUST_ID TIME_ID   QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- ---------- --------- ------------- -----------
       961      11160      17450 01-JUN-18            20         800
Execution Plan
----------------------------------------------------------
Plan hash value: 1909132751
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |              |     1 |    28 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID READ OPTIM| SALES_TAB    |     1 |    28 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN READ OPTIM           | SYS_C0022714 |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SALES_ID"=961)
```

Listing 12

griff auf externe Daten und ermöglicht die Entwicklung einfacher und effizienter Datenbank-Anwendungen.

Darüber hinaus ist es mit 18c jetzt auch möglich, External-Table-Informationen in Verbindung mit dem In-Memory Column Store zu verwenden. So lassen sich Daten, die in externen Dateien vorliegen, in diesen Column Store im Hauptspeicher laden, um damit alle Vorteile der In-Memory-Funktionen (wie zum Beispiel Performance-Steigerung beim Zugriff) nutzen zu können. Verwendet man den neuen Autonomous Database Cloud Service, gibt es darüber hinaus eine interessante Erweiterung: Externe Daten, die im Objekt-Storage gespeichert sind, können einfach über eine External Table zur Verfügung gestellt werden.

Bei Oracle Database 18c gibt es einen neuen Memory-Bereich, den Memory-optimierten Rowstore (auch „MemOptimized Rowstore" genannt), nicht zu verwechseln mit dem In-Memory Column Store. Konzipiert für hohe Abfrage-Leistungen (wie Internet-of-Things-Workloads), sorgt er für sehr schnelle und latenzarme Abfragen. Key-Value-Lookups auf Basis von Primärschlüsselwerten (mit Abfragefilter „Spalte = Wert") nutzen direkt einen Memory-Hash-Index bei der Ausführung. Das neue Attribut auf Tabellen-Ebene („MEMOPTIMIZE FOR READ") gibt an, welche der Tabellen mit dem neuen Memory-Hash-Index in den Buffer-Cache gepinnt werden sollen. Die Datenbank erzeugt einen Hash-Index und pflegt ihn automatisch. Das Beispiel in *Listing 12* zeigt eine Ausführung mit dem Memory-Hash-Index.

Das Feature „MemOptimized Rowstore" ist im Moment auf Exadata und dem Oracle Database Cloud Service „Enterprise Edition Extreme Performance" verfügbar. Weitere Informationen dazu stehen im Handbuch „Licensing Information User Manual".

Neue, aussagekräftige Optimizer-Hints wie „OPTIMIZER_IGNORE_HINTS" und „OPTIMIZER_IGNORE_PARALLEL_HINTS", die speziell für das Arbeiten mit der Autonomous Database eingeführt wurden, erleichtern ab 18c das Testen von Queries mit Hints. Der Default ist in den On-Premises-Installationen natürlich „FALSE". Möchte man wissen, wie eine Abfrage ohne Hints ausgeführt wird, können diese Parameter in der Session verändert werden. Mehr Details dazu und zu weiteren 18c-Features stehen auf „blogs.oracle.com/coretec" in der Kategorie „18c".

Fazit

Sich mit SQL-Funktionen auseinanderzusetzen und neue Funktionen zu erlernen kann viele Vorteile bringen. Zum Erlernen eignet sich beispielsweise die Code Library in „Oracle Live SQL:", die kleinen Tutorials erklären häufig neue Features. Auch die Werkzeuge wie SQL Developer oder die neue Linemode-Variante „SQLcl" bieten gerade beim Testen hilfreiche Funktionen.

Ob die Funktion in der Cloud oder in ihrer eigenen Umgebung (soweit verfügbar) getestet wird, spielt dabei übrigens keine Rolle; die Funktionen bleiben die gleichen. Dabei bieten SQL Developer, SQLcl und auch andere Werkzeuge in der Regel einfache Möglichkeiten, sich mit den Oracle-Cloud-Umgebungen zu verbinden.

Man sollte sich allerdings im Klaren darüber sein, was man programmiert hat beziehungsweise welche Performance-Auswirkungen die Änderung am eigenen Code haben kann. Daher sollte vor jedem produktiven Einsatz nicht nur die Überprüfung des Ergebnisses stehen, sondern man sollte, wenn möglich, sich auch ein Bild von der Ausführungs-Performance beziehungsweise vom Ausführungsplan machen.

Handbücher und interessante Links

- SQL Language Reference
- Development Guide
- Licensing Information User Manual
- Oracle Magazine: A Window into the World of Analytic Functions: *https://blogs.oracle.com/oraclemagazine/a-window-into-the-world-of-analytic-functions*
- Oracle Dojos: *tinyurl.com/dojoonline*
- Oracle-Datenbank und Cloud Community Blog: *blogs.oracle.com/coretec*
- Oracle Live SQL: *http://livesql.oracle.com*

Ulrike Schwinn
ulrike.schwinn@oracle.com

… die Schlechten ins Kröpfchen – SQL-Analyse für DBAs

Martin Klier, Performing Databases GmbH

Dieser Artikel zielt auf alle, die bereits erste Erfahrungen mit der Oracle-Datenbank sammeln konnten und sich für die Analyse und Optimierung von Performance-Vorfällen interessieren.

Das Finden, Validieren und gegebenenfalls Verbessern von langsamem oder Ressourcen-intensivem SQL ist Teil der Aufgaben jedes DBA, der seine/ihre Aufgaben ernst nimmt. Es kann schon spannend genug sein, entsprechende Statements zu identifizieren; doch Verbesserungen für etwas vorzuschlagen oder auszuführen, dessen Zweck oft nicht ganz klar ist, ist eine ganz andere Hausnummer. Auf diese Art werden wir jedoch niemals Verbesserungen um Faktoren von einer Million oder Hunderttausend erreichen – das ist ein Privileg der Designer und Anwendungsentwickler. Es gibt allerdings einen generischen Ansatz für uns „Datenbank-Flüsterer", um Statements anzupacken, die Aufruhr verursachen. Damit erreichen wir zwar nur Faktor zwei, fünf oder zehn – mehr ist oft nicht drin. Aber auch das ist immer einen Versuch wert und hilft oft deutlich weiter.

Aufbau der Datenbank

Das relationale Datenbank-Management-System (RDBMS) von Oracle besteht im Wesentlichen aus zwei Teilen: Während alle Komponenten im RAM wie Speicherbereiche und Prozesse zur „Instanz" zählen, bilden die Datafiles, Online-Redo-Logs und Control-Files die eigentliche „Datenbank". Diese Unterscheidung ist sinnvoll, denn die Datenbank stellt das Ergebnis aller Bemühungen des RDBMS dar. Die Instanz ist lediglich das Werkzeug, das zum Ziel führt. *Abbildung 1* aus der Oracle-Dokumentation 12cR1 zeigt das Zusammenwirken von Instanz und Datenbank.

Arbeiten mit dem Massenspeicher

Zur Wiederholung sei kurz und vereinfacht die Handhabung von Daten im Oracle RDBMS dargestellt. Reguläre Lesevorgänge *(siehe Abbildung 2)* erfolgen durch den jeweiligen Server-Prozess („Session") und bringen relevante Blöcke im Ganzen in den Buffer-Cache. Ab hier spricht man nicht mehr vom Block, sondern vom Buffer. Von dort aus grenzt der Server auf Teilmengen/Rows ein.

Verändernde Transaktionen schreiben zunächst auf den Buffer im Cache und erzeugen dort ebenso Undo-Informationen in einem weiteren Buffer *(siehe Abbildung 3)*:

- Für beide Vorgänge werden Redo-Daten im Redo-Log-Buffer eingetragen.
- Alle Buffer des Cache werden nach und nach durch den Komplex aus den Checkpoint- (CKPT) und Database-Writer-Prozessen (DBWn) in die Tablespace Datafiles synchronisiert.
- Der Log-Writer-Prozess (LGWR) schreibt die Inhalte des Redo-Log-Buffer in die Online-Redo-Logs. Sind alle Redo-Logs voll und alle referenzierten Buffer synchronisiert, so werden die ältesten Einträge im Redo-Log wieder überschrieben.
- Die aktuelle System-Change-Number wird im Control-File aktualisiert, ebenso gegebenenfalls neue Datafile-Größen.
- Optional, aber sehr üblich für den „Archivelog-Mode": Die Archiver-Prozesse (ARCn) sorgen für die Duplizierung aller vollständig gefüllten Online-Redo-Logs in die sogenannten „Archived Redo Logs" (kurz: Archivelogs), die für verschiedene Recovery-Szenarien benötigt werden, etwa ein vollständiges Recovery.
- Optional für Flashback-Database, nur in Enterprise Edition: Der Recovery-Writer-Prozess (RVWR) kopiert angefallene Undo-Informationen in die Flashback-Logs. Diese finden Verwendung, um etwa die ganze Datenbank in der Zeit zurückrollen zu können, ohne ein Backup einzuspielen.

Parsing und Optimierung

Die über Sessions eingehenden Anweisungen liegen in hochsprachlicher Form als SQL vor. Diese Abstraktion ist vom DBMS in konkret ausführbare Anweisungen zu übersetzen. Daher wird jedes Statement einem Parsing- und gegebenenfalls auch einem Optimierungsprozess unterzogen.

Nach der relativ unkomplizierten Prüfung der syntaktischen Korrektheit validiert der Parser die Semantik: Stimmen

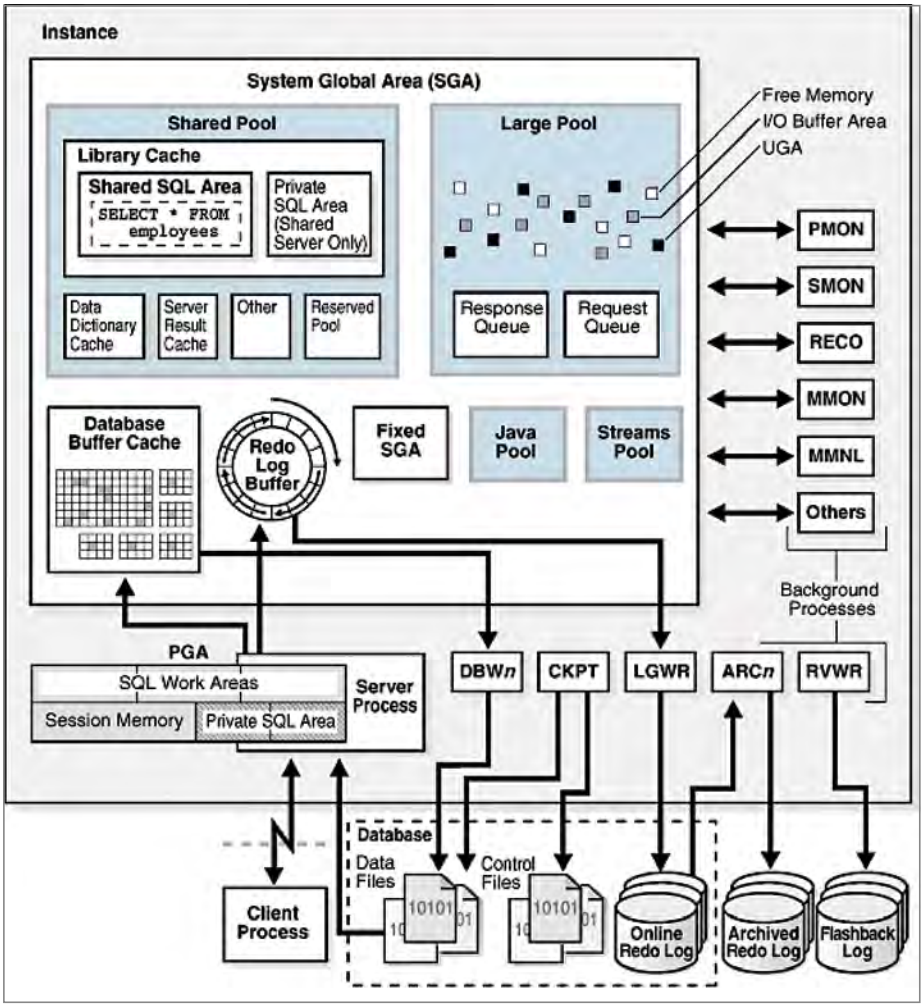


Abbildung 1: Zusammenwirken von Instanz und Datenbank

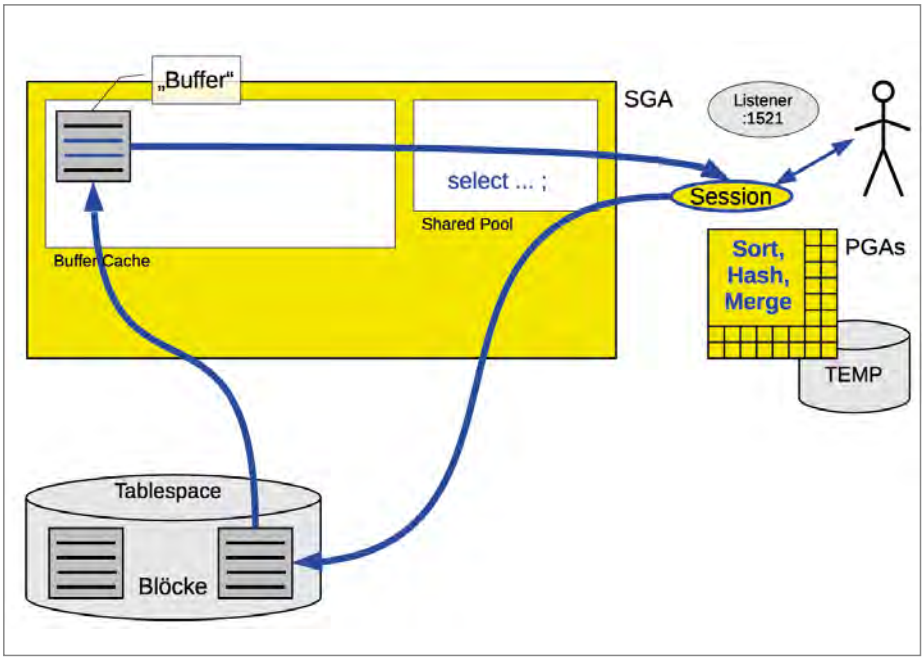


Abbildung 2: Lesevorgang

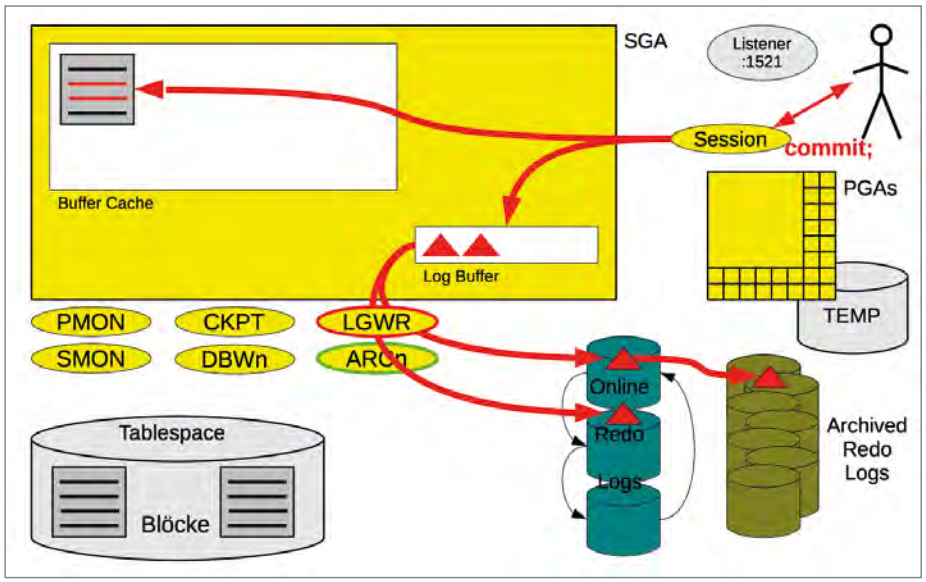


Abbildung 3: Schreibvorgang

Objektnamen, Felder, Datentypen und deren Zusammenhänge überein? Hierzu sind Zugriffe auf das Data Dictionary notwendig – das jedoch in aller Regel gut priorisiert in seinem Cache liegt.

Cursor Sharing

Oracle hat schon vor langer Zeit erkannt, dass der eigentliche SQL-Optimierungsvorgang (also das maschinelle Finden des besten Wegs, alle erforderlichen Daten in Zugriff zu nehmen) viele Ressourcen verbraucht und daher möglichst selten stattfinden sollte. Dieses Feature wird als „Cursor Sharing" bezeichnet, und erlaubt unter Nutzung des Library Cache, bereits erstellte Ausführungspläne wiederzuverwenden. Es entsteht ein sogenannter „Cursor", der durch den Hashwert des SQL-Statements identifiziert und mit diesem als Schlüssel abgelegt wird. Er beinhaltet unter anderem den erstellten Ausführungsplan und diverse Informationen über die Umgebung, in der er entstand. Beim sogenannten „Hard Parse" *(siehe Abbildung 4)* erfolgt der volle Durchgang des Optimizers, beim „Soft Parse" *(siehe Abbildung 5)* wird nach erfolgreichem „Library Cache Check" der vom ersten Lauf hinterlegte Plan zur Ausführung herangezogen.

Voll zum Tragen kommen die Vorteile des Cursor Sharing erst, wenn variable Werte (die zu abweichenden Hashwerten führen und Cursor Sharing unmöglich machen würden) als sogenannte Bind-Variablen übergeben werden. Diese werden beim ersten Execute ausgewertet („Bind Peeking") und fließen in die Optimierung wesentlich mit ein.

Dieses Verfahren hat jedoch einen gravierenden Nachteil: Wir sparen zwar einen Optimierungslauf ein, dieser nimmt jedoch an, dass alle Werte dieser einen Variable die gleiche Selektivität aufweisen werden. Eine Nachprüfung findet nicht statt. Zur Illustration verweise ich auf meinen Blog-Artikel „Oracle Depends on Humidity" aus dem Jahr 2012 *(siehe „http://www.usn-it.de/index.php/2012/08/29/r-i-p-oracle-database-10g-oracle-depends-on-humidity")*.

Adaptive Cursor Sharing

Ab Oracle 11.1 wurde für die nachträgliche Ermittlung der Selektivität einer Bind-Variablen ein Mechanismus eingebaut, der „Adaptive Cursor Sharing" (ACS), also etwa „sich anpassendes Cursor Sharing", genannt wurde. Nach einigen Anlaufschwierigkeiten stand er ab 11.2 zuverlässig zur Verfügung. Dieses ACS basiert auf „Statistics Feedback" (vor 12.1 hieß es noch „Cardinality Feedback"), nämlich der Rückmeldung, ob die aus den Statistiken gewonnenen Kardinalitäten auch den Ergebnissen des letzten Laufs entsprachen. Falls nicht, wird der Plan als „Bind Aware" gekennzeichnet, bei der nächsten Ausführung dieses SQL-Statements die Bind-Variable erneut ausgewertet und ein neuer Ausführungsplan für die ermittelte Selektivitätsklasse der Variablen hinterlegt. Nun ist der Plan

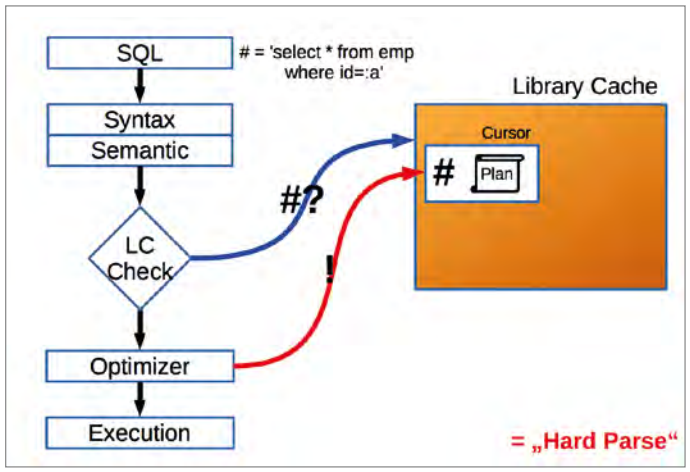


Abbildung 4: Hard Parse

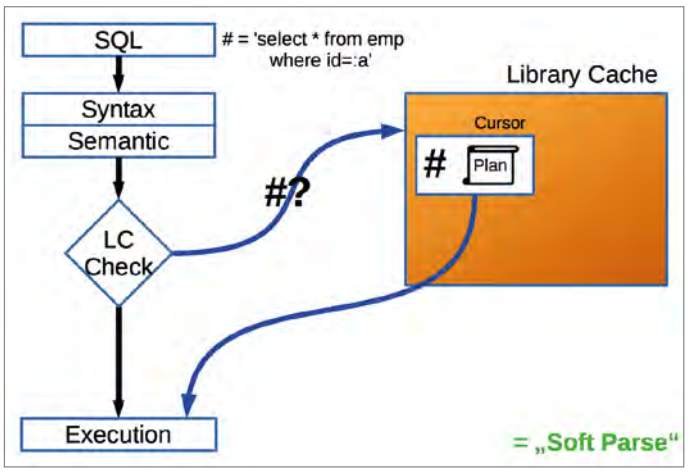


Abbildung 5: Soft Parse

auch „Bind Sensitive“ und bei jeder neuen Ausführung wird immer mindestens die Selektivität der Bind-Variablen hinterfragt („Bind Peeking“). Das ist nun etwas teurer als der klassische „Soft Parse“ und trägt daher den Untergrundnamen „Harder Soft Parse“ *(siehe Abbildung 6)*.

Mein Weg der Analyse

Ich möchte Ihnen nach dieser Einführung in die Grundlagen gerne meinen Weg zeigen, an die Analyse heranzugehen. Das muss nicht jedermanns Geschmack sein und ist sicher auch nicht perfekt. Aber es hat seine Berechtigung und passt zu meiner generellen Denk- und Arbeitsweise.

Das ist ein wichtiger Punkt: Ein Performance-Experte darf und soll seinen eigenen Stil entwickeln, denn einen universell richtigen Weg gibt es nicht. Spielen Sie keine angenommene Rolle, sondern seien Sie Sie selbst, stehen Sie zu ihren Methoden und Ergebnissen. Jedoch würde ich Lernfreude, Ergebnisoffenheit, Faktenwissen und die Fähigkeit, Ergebnisse verständlich darzustellen, als allgemein äußerst nützlich einstufen.

Die Analyse beginnt stets mit der Frage: „Was ist das Problem?” Denn nur wenn der Beschwerdeführer abgeholt und verstanden wird, kann sich überhaupt eine Ursache oder Lösung finden, die dieser auch anerkennt.

Womit man dann beginnt, ist eine Frage der Methodik, Erfahrung und nicht zuletzt von Instinkt. Letzteren haben jeder Rennfahrer, Reiter, Segler oder Flieger: das gute alte „Po-Gefühl“ für sein Arbeitsgerät. Wenn ich nun zum ersten Mal eine bis dato unbekannte Umgebung sattle, ist es das erste Bestreben, diese Verbindung aufzubauen und mit allen Sinnen einzutauchen in die Umweltbedingungen, die dort herrschen. Das klingt vielleicht etwas pathetisch, trifft es aber ziemlich genau.

Instinkt ist jedoch der persönliche Weg. Er fördert im Durchschnitt zwar gepaart mit etwas Erfahrung erheblich die Effizienz und beschleunigt damit den Weg zu einer Lösung. Instinkt hat allerdings in Begründungen, Ergebnissen und Schlussfolgerungen nichts verloren. Entscheidungen haben auf soliden Fakten und Hintergrundwissen zu basieren.

Tools

Meine Liste empfohlener Tools für die Analyse von Oracle-Datenbank-Performance ist erfreulich kurz:

- AWR + ASH oder Statspack
- SQL*Plus + rlwrap oder SQLcl
- SQL Developer (man muss nicht puristischer sein als nötig)
- dbms_xplan.display_cursor()

Dazu kommt dann fallabhängig für schnellen Überblick und effiziente Kommunikation von Zahlen noch die Datenaufbereitung und Visualisierung etwa mit Splunk, R oder im Notfall eine Tabellenkalkulation mit Charting-Funktion.

Der Störenfried

Die Ursache oder die Ursachen zu finden, die wesentlich zur beklagten Situation beitragen, ist selten einfach und hängt stark vom umgebenden Business-Case ab. Unter der Annahme, dass tatsächlich ein oder mehrere SQL-Statements verantwortlich sind, bietet sich die Unterscheidung von zwei Fällen an:

- Ein Statement sättigt eine Ressource so, dass die Gesamtleistung beeinträchtigt wird. In dem Fall führen alle Methoden zur Last-Analyse zum Ziel (wie AWR). Hier ist der DBA zu Hause.
- Ein Einzelvorgang dauert zu lange, ist aber in der Gesamtbelastung zu vernachlässigen. Hier greift nur der Ansatz über die Applikation, über ein auf Debugging oder eine Code-Instrumentierung basierendes Profiling. Erst dann setzt die Datenbank-basierte Fehlersuche effektiv ein, kann jedoch vorher schon über das Ausschlussverfahren nützliche Hinweise zum Gesamtbild beitragen (etwa Auflistung relevanter SQLs mit Laufzeit > x).

In beiden Fällen sind die hier vorgestellten Ansätze in unterschiedlicher Ausprägung nützlich.

AWR-Reports auswerten

Ich setze im Folgenden die Kenntnisse zum Erzeugen eines AWR-Reports für einen bestimmten Zeitraum voraus. Liegt dieser dann vor, so ist zunächst die Relation zwischen Systemleistung und Belastung im betrachteten Zeitraum interessant. Es handelt sich hier um eines von vielen möglichen Szenarien, nur dazu gedacht, den Ansatz zu vermitteln:

- Wie viel Zeit betrachten wir (Elapsed Time)? Etwa 1 h = 60 min = 3.600 s
- Wie viele CPU-Cores standen zur Verfügung? Etwa 12 = 720 CPU-Minuten
- Wie viel Zeit hat die DB als Arbeitszeit aufgezeichnet (DB Time)? Etwa 52 min

Hier scheidet auf den ersten Blick globale CPU-Überlast aus (52 min von 720 min), allerdings entsprechen 60 min den 100 Prozent eines Cores, und hier liegen wir schon dichter daran. „Report Summary“ liefert in Form von User Calls, SQL Executes und Transactions (jeweils Durchschnitte pro Sekunde) einen wichtigen Eindruck von der Last in diesem Zeitraum.

Einordnung

Im „Report Summary“ lohnt sich im weiteren Verlauf eine gewisse Kategorisierung nach IO- und CPU-Anteilen. So ordne ich Logical Reads, Block Changes, Parses und Hard Parses uneingeschränkt der Kategorie „CPU“ zu: Speicherzugriffe sind nun einmal CPU-Arbeit. Physical Read/Write, Read/Write IO Requests, Read/Write IO gehören ebenso klar zum IO-Teil.

Die Top-10-Foreground-Events liefern weitere Relationen: Zu wie viel Prozent (von oben „DB Time“) haben wir CPU genutzt? Etwa 95 Prozent von 52 = 49,5 min. Wie viel Prozent IO? Etwa 9 Prozent. Wie viele Serialisierungen (Latches, Mutextes etc.)? Etwa keine.

Bitte beachten Sie, dass manche Waits CPU-Zeit beinhalten, so zum Beispiel „log file sync“ oder „Mutex waits“. Dadurch addieren sich die meisten Fälle auf mehr als 100 Prozent auf. Ich möchte dazu auf die vorzüglichen Detail-Analysen zum Wait-Interface und seinen Messfehlern hinweisen, die Frits Hoogland veröffentlicht hat.

Fallabhängige Recherche I

Durch die oben beispielhaft verwendeten Zahlen deutet vieles auf hohe Beanspru-

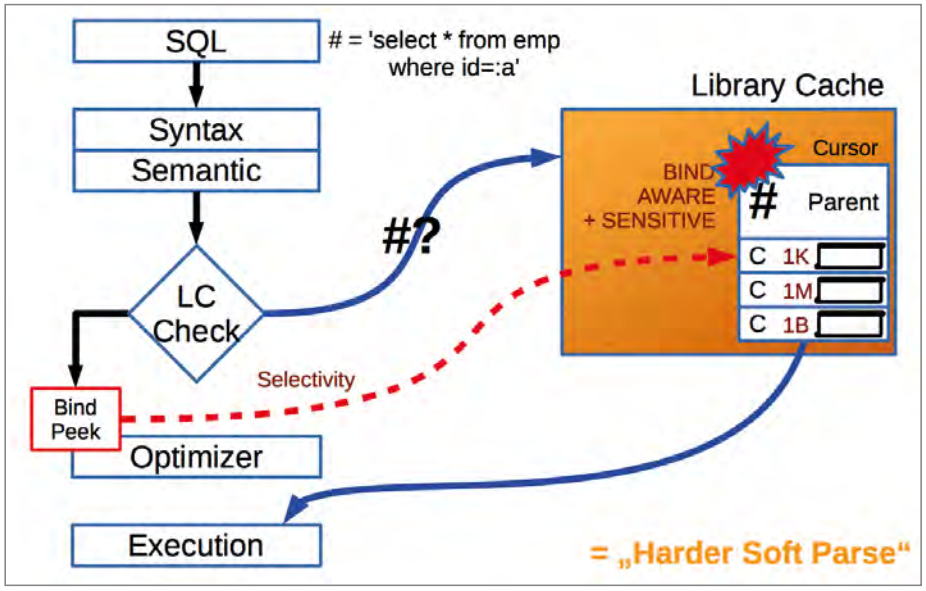
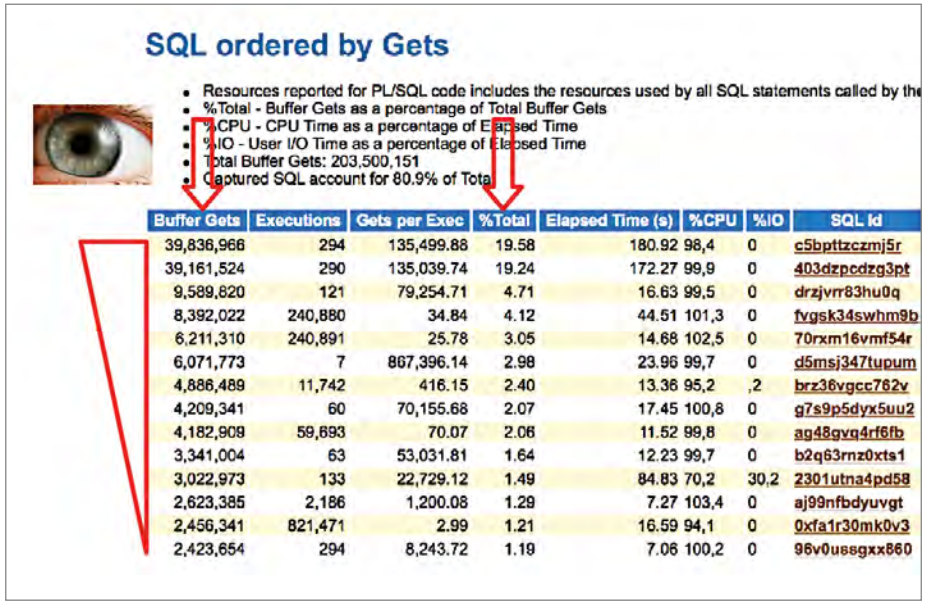


Abbildung 6: Harder Soft Parse

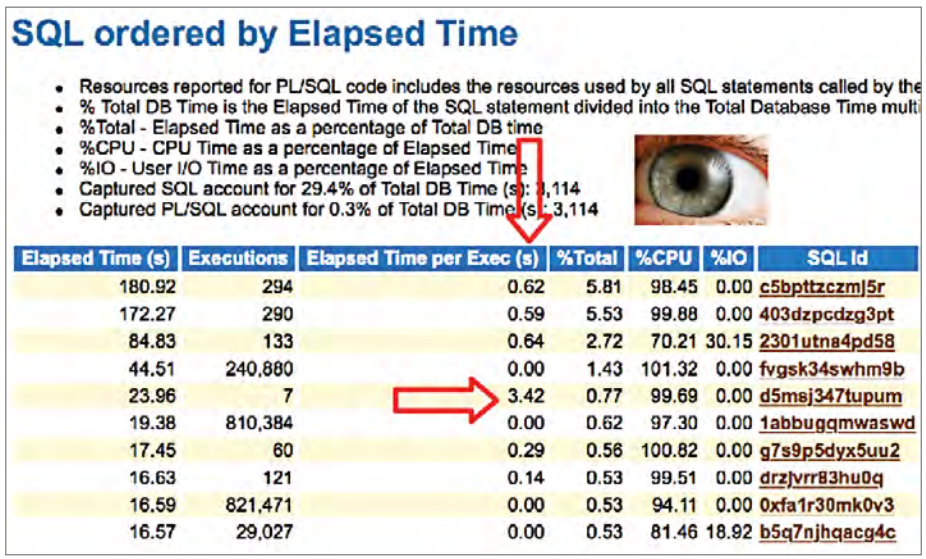


SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the
- %Total - Buffer Gets as a percentage of Total Buffer Gets
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Buffer Gets: 203,500,151
- Captured SQL account for 80.9% of Total

Buffer Gets	Executions	Gets per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id
39,836,966	294	135,499.88	19.58	180.92	98,4	0	c5bpttzczmj5r
39,161,524	290	135,039.74	19.24	172.27	99,9	0	403dzpcdzg3pt
9,589,820	121	79,254.71	4.71	16.63	99,5	0	drzjvrr83hu0q
8,392,022	240,880	34.84	4.12	44.51	101,3	0	fvgsk34swhm9b
6,211,310	240,891	25.78	3.05	14.68	102,5	0	70rxm16vmf54r
6,071,773	7	867,396.14	2.98	23.96	99,7	0	d5msj347tupum
4,886,489	11,742	416.15	2.40	13.36	95,2	,2	brz36vgcc762v
4,209,341	60	70,155.68	2.07	17.45	100,8	0	g7s9p5dyx5uu2
4,182,909	59,693	70.07	2.06	11.52	99,8	0	ag48gvq4rf6fb
3,341,004	63	53,031.81	1.64	12.23	99,7	0	b2q63rnz0xts1
3,022,973	133	22,729.12	1.49	84.83	70,2	30,2	2301utna4pd58
2,623,385	2,186	1,200.08	1.29	7.27	103,4	0	aj99nfbdyuvgt
2,456,341	821,471	2.99	1.21	16.59	94,1	0	0xfa1r30mk0v3
2,423,654	294	8,243.72	1.19	7.06	100,2	0	96v0ussgxx860

Abbildung 7: SQL ordered by Gets



SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multi
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 29.4% of Total DB Time (s): 3,114
- Captured PL/SQL account for 0.3% of Total DB Time (s): 3,114

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id
180.92	294	0.62	5.81	98.45	0.00	c5bpttzczmj5r
172.27	290	0.59	5.53	99.88	0.00	403dzpcdzg3pt
84.83	133	0.64	2.72	70.21	30.15	2301utna4pd58
44.51	240,880	0.00	1.43	101.32	0.00	fvgsk34swhm9b
23.96	7	3.42	0.77	99.69	0.00	d5msj347tupum
19.38	810,384	0.00	0.62	97.30	0.00	1abbugqmwaswd
17.45	60	0.29	0.56	100.82	0.00	g7s9p5dyx5uu2
16.63	121	0.14	0.53	99.51	0.00	drzjvrr83hu0q
16.59	821,471	0.00	0.53	94.11	0.00	0xfa1r30mk0v3
16.57	29,027	0.00	0.53	81.46	18.92	b5q7njhqacg4c

Abbildung 8: SQL ordered by Elapsed Time

chung der CPU ohne Serialisierungsprobleme hin. Damit rücken die Buffer Gets in den Fokus, und SQLs mit vielen Buffer Gets zeigt AWR sehr zuverlässig an *(siehe Abbildung 7)*. Zusammengeführt mit dem Problem („Ist eines der genannten SQLs relevant?“) liefert dieser Ansatz schon einmal eine oder mehrere SQL_IDs, die näher untersucht werden sollten.

Fallabhängige Recherche II

Außerdem sind natürlich auch die Ausführungszeiten interessant – einmal ganz offensichtlich natürlich für den Endanwender, falls die Applikation in hohem Maße von der Antwortzeit eines Einzelstatements abhängt. In jedem Fall sind jedoch das genau die Zeiten, die auch im Profil der Applikation wiederzufinden sind.

Das konkrete Beispiel in *Abbildung 8* zeigt den Abschnitt „SQL ordered by Elapsed Time“. Hier ist ein Statement zu finden, das im Schnitt 3,42 Sekunden für eine Ausführung benötigt und sieben Mal ausgeführt wurde. Sollte dieses interaktiv laufen und die Applikation einen gewissen Echtzeitanspruch haben, kann und wird es zu Beschwerden führen. Auch dafür lohnt es sich, die SQL_ID für die Untersuchung vorzumerken.

Weitere Tipps

Folgende Punkte empfehle ich noch zur weiteren Prüfung, als Joker oder um auch bei nicht ganz alltäglichen Problemen die Analyse erfolgreich abzuschließen:

- Anzahl SQL Executions (denke: Software läuft, so schnell sie kann)
- Shared Memory Nutzung/Version Count (etwa durch Child-Cursor-Probleme)
- Plausibilität der Instanz-Parameter prüfen
- NLS-Einstellungen ansehen und bei der weiteren Analyse berücksichtigen (Index-Nutzung, Sortierung, String-Vergleiche etc.)
- Advanced Queues und (drei Arten von) Jobs prüfen

Ausführungspläne lesen

Ist der „Störenfried“ gefunden, geht es darum, diesen im Detail zu untersuchen.

```
SELECT *
FROM table(
      DBMS_XPLAN.DISPLAY_CURSOR(
          '&&SQL_ID',
          null,
          'COST,IOSTATS,LAST,ADVANCED,ADAPTIVE'
      )
  );
```

Listing 1

Ein wichtiges Werkzeug dafür ist der Ausführungsplan („Execution Plan"). Alle Ausführungspläne im Folgenden wurden nach meiner Lieblingsmethode erzeugt *(siehe Listing 1)*.

Fallabhängige Recherche III

Das Beispiel, das ich bearbeiten möchte, zeigt einen Ausführungsplan mit hohen Kosten (rund 34.000), der im ersten Beispiel zu „Lesen von AWR-Reports" durch sehr viele Buffer-Gets aufgefallen ist. Die Plausibilitätsprüfung hat ebenfalls ergeben, dass das Statement relativ langsam läuft, etwa fünf Sekunden wurden genannt.

Läuft das Datenbank-System auf „STATISTICS_LEVEL=TYPICAL", müssen wir im Execution-Plan leider auf die Auswertung tatsächlicher Buffer-Gets und realer Kardinalität verzichten. Ich würde dem stets den Vorzug geben, durch seine Auswirkungen auf Ausführungszeiten und AWR-Größe ist „STATISTICS_LEVEL" allerdings nicht in allen produktiven Umgebungen veränderlich – außer in Zeiten allerhöchster Not.

Aber auch anhand der Estimated Rows und Estimated Bytes ist mein Ansatz anschaulich zu demonstrieren. Auf der Suche nach dem bedeutendsten Lastverursacher findet sich zunächst einmal der Hash-Join in Zeile 3. Dieser wiederum nährt sich vorrangig aus dem „Table Access by Index Rowid" in Zeile 5 (und nicht, wie man zunächst vielleicht beanstanden möchte, aus dem „Table Access Full" in Zeile 4).

Die Zeilen 5 und 6 lösen beim Zugriff auf die Tabelle „MAILING" und einen ihrer Indizes zwei Prädikate auf. Dies erkennen Sie an den Asterisken (*) vor der Zeilennummer. Im Bereich „Predicate Information" weiter unten (blaue Markierung) finden Sie die Gleichungen, die an der Stelle evaluiert werden *(siehe Abbildung 9)*.

Wir sehen im konkreten Fall die „CLIENTPRJID" gegen die erste Bind-Variable und „FLAGDISPATCHNODEPRJ" im Rahmen einer „NLSSORT()"-Funktion gegen die zweite Bind-Variable. Diese Funktion bewirkt mit dem Parameter „BINARY_CI", dass das angegebene Textfeld Case-Insensitive (CI), also ohne Berücksichtigung von Groß- und Kleinschreibung, durchsucht wird.

Ganz offensichtlich ist jedoch „CLIENTPRJID" allein nicht sehr selektiv, sodass der Großteil der Rohdaten gegen die „NLSSORT()"-Funktion geprüft werden muss. Das bedeutet viele Buffer-Gets aus dem Cache (oder „db file sequential read" von Disk) plus erhebliche CPU-Belastung.

Lösung

Um das beschriebene Problem zu lösen, empfiehlt sich hier zunächst die Prüfung

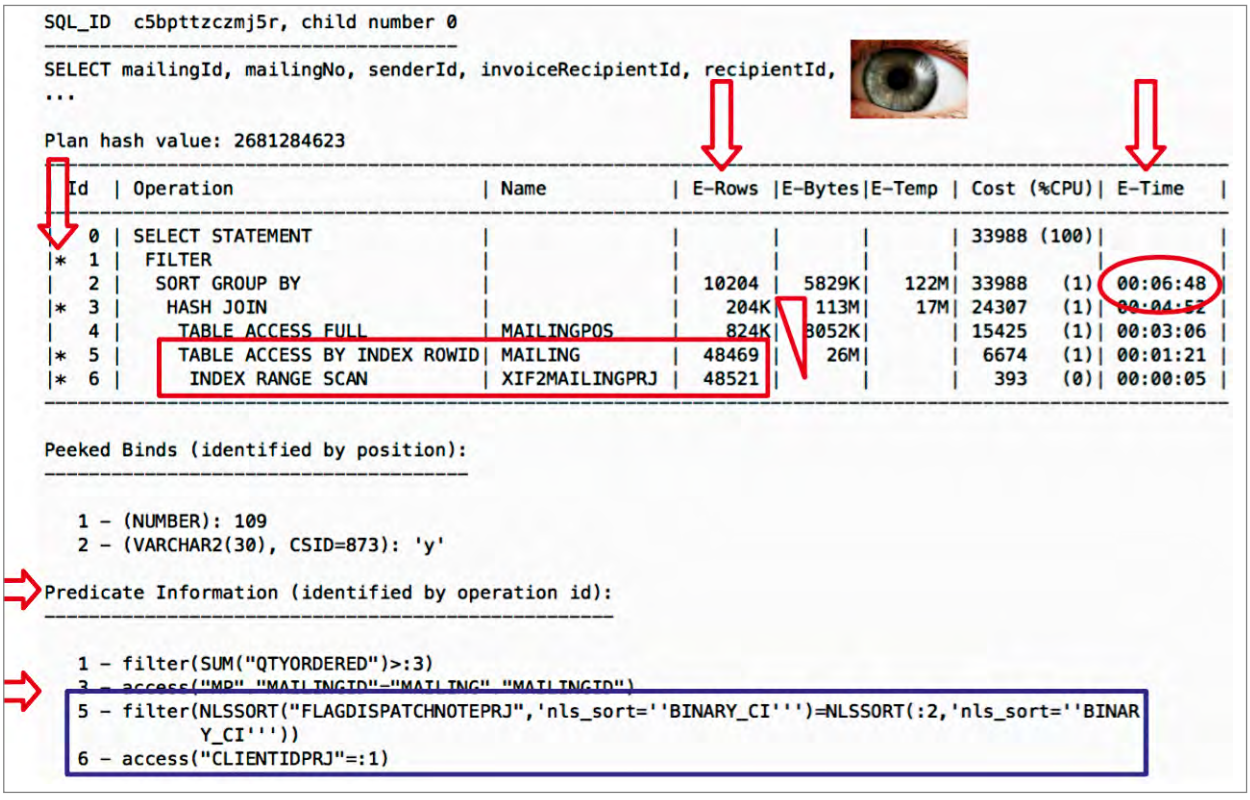


Abbildung 9: Xplan1

der Selektivität verwendeter Bind-Variablen, wie sie „Peeked Binds“ angibt. Doch Vorsicht, hier handelt es sich nur um die Inhalte der beiden Variablen, die zur Erzeugung des Plans geführt haben. Das muss nicht repräsentativ sein und es empfiehlt sich, das System dahingehend weiter zu untersuchen.

Ist man sich jedoch hinreichend sicher, dass „FLAGDISPATCHNODEPRJ“ die bessere und „CLIENTPRJID“ die zweitbeste Einschränkung auf die Tabelle „MAILING“ ist, bietet sich ein mehrspaltiger Function-Based-Index zur Behebung des Problems an *(siehe Listing 2)*.

Danach sollten unbedingt neue Statistiken auf die Tabelle und ihre abhängigen Objekte erzeugt werden – ich empfehle in der Regel, für alle „Skewed“-Columns auch Histogramme anzulegen. Natürlich sollte ein solcher Eingriff in das Schema der Applikation von einem Change-Prozess begleitet werden. Mindestens umfasst dieser die Gegenprüfung darauf, ob sich ein ähnliches Statement nun besser im Sinne der Performance-Entfaltung verhält.

Diese Kalkulation geht hier auf: Beide Prädikate werden nun in Zeile 5 über einen „INDEX RANGE SCAN“ evaluiert und die Kosten fallen auf 8 (von 34.000). Der

```
Create index I_MAILING_TUNING_4 on MAILING (
   NLSSORT("FLAGDISPATCHNOTEPRJ",'nls_sort=''BINARY_CI''''),
   CLIENTIDPRJ
   );
```

Listing 2

Optimizer scheint nun bezirzt – doch wie sieht die Realität aus? Diese lässt sich nur über die SQL-Ausführungsstatistiken ermitteln, und hier konnten die durchschnittlichen Buffer-Gets pro Execution von 135 k auf 1 k reduziert werden. Das ist noch immer nicht brillant, aber schon deutlich besser. Die Ausführungszeit fiel von 6:48 s auf 0:01 s, damit war der Fall für den Kunden erledigt *(siehe Abbildung 10)*.

Die Nachsorge erstreckte sich dann auf die Überprüfung der Planstabilität, wofür später noch Extended Statistics für die Kombination der beiden Spalten angelegt wurden. Damit erkennt der Optimizer deren gemeinsame Kardinalität.

Tipps

Kurz vor Schluss noch einige Ratschläge aus der Praxis für die Praxis:

- Der nächste Crash / die Eskalation / die Krise kommt bestimmt. Sorgen Sie vor, indem Sie Ihre Systeme regelmäßig auf anfällige Statements überprüfen. Statements, die viele Ressourcen (wie Buffer-Gets oder IOs) benötigen, reagieren empfindlich auf deren Verknappung. Die nächste Beschwerde ist dann absehbar. Sorgen Sie vor, schaffen Sie vermeidbare Baustellen schon zu Friedenszeiten aus dem Wege.
- Einer weiß es immer besser: die Kollegen, die Vorgesetzten, ein Berater. Setzen Sie sich in Ruhe (auch wieder zu Friedenszeiten) hin und entwickeln Sie Ihren eigenen Stil. Bereiten Sie Ihre Tools vor, werden Sie damit vertraut. Ob sie selbst gebaut oder gekauft, frei oder kostenpflichtig sind, ist völlig unerheblich: Sie müssen für das passen, was Sie können und erreichen wollen, Sie sollten allerdings

```
SQL_ID  c5bpttzczmj5r, child number 0
-------------------------------------
SELECT mailingId, mailingNo, senderId, invoiceRecipientId, recipientId,
...

Plan hash value: 2016964577

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |        |       |     8 (100)|          |
|*  1 |  FILTER                       |                     |        |       |            |          |
|   2 |   SORT GROUP BY               |                     |      1 |   600 |     8  (13)| 00:00:01 |
|   3 |    NESTED LOOPS               |                     |      1 |   600 |     7   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| MAILING             |      1 |   590 |     4   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN         | I_MAILING_TUNING_4  |      1 |       |     3   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID| MAILINGPOS          |      4 |    40 |     3   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN         | XIF1MAILINGPOS      |      4 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - (NUMBER): 401
   2 - (VARCHAR2(30), CSID=873): 'y'

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(SUM("QTYORDERED")>:3)
   5 - access("MAILING"."SYS_NC00090$"=NLSSORT(:2,'nls_sort=''BINARY_CI''') AND
              "CLIENTIDPRJ"=:1)
   7 - access("MP"."MAILINGID"="MAILING"."MAILINGID")
```

Abbildung 10: Xplan2

vertraut damit sein. Dann können Sie Ihre eigenen Ergebnisse erklären und Ihre Herangehensweise begründen. Fehler zu machen, liegt in der Natur der Sache, mit jedem Fehler werden Sie jedoch auch besser.

- Messen ist immer ein Vergleich. Meine klare Empfehlung ist, alles auf die Sekunde herunterzubrechen. Die Faktoren 3.600 auf die Stunde und 86.400 für einen Tag sehen zunächst sperrig aus, aber sie gehen schnell in Fleisch und Blut über. IOs, FLOPs und hundert andere Referenzwerte der Hardware (und oft auch der Software) nutzen die Sekunde als Basis. Lassen Sie sich davon anstecken, es fördert die Transparenz und die Verständlichkeit.
- Wissen ist Macht, doch nur Beweise sichern die Weltherrschaft. Schön wäre es – aber Fakt ist auch, dass Wissen ohne Beweis allenfalls am Stammtisch oder beim Mittagessen Wert hat. Spätestens beim Budget hört meist der Spaß auf, und wenn in einer Krise schmerzhafte Maßnahmen angesagt sind, möchte jeder Manager mehr in der Hand haben als das erprobte Bauchgefühl seines Datenbank-Experten. Lernen Sie daher, von der Aufnahme eines Falls bis zur Lösung alles zu dokumentieren. Damit sichern Sie wichtige Beweismittel über die erreichte Verbesserung und die getroffenen Entscheidungen. Ob Sie Eintragungen in ein Ticket-System, Textfile, Wiki oder Notizsystem machen, ist unerheblich. Aber eine klar aufgezeichnete Maßnahmenkette lässt sich nicht mehr plausibel widerlegen und ist ab dann mindestens als ein Zweig der Wahrheit zu berücksichtigen. Auch der vielgeschmähte Screenshot hat trotz im Überfluss verfügbarer Bildbearbeitungssoftware noch immer seinen Wert.

Fazit

Das komplexe Thema kennt kein Ende, nur die Aufforderung an alle Leser, sich weiterzubilden und die losen Enden der Geschichte aufzugreifen, als Ansatzpunkt für das nächste Projekt oder für die eigene Fortbildung aus Interesse heraus.

Martin Klier
martin.klier@performing-db.com

Debugging und Tuning von Apex-Anwendungen

Till Albert, MT AG

Ist die Apex-Anwendung zu langsam? Die Gründe dafür können vielseitig sein, doch mit den richtigen Hilfsmitteln lassen sie sich aufspüren und beseitigen. Dieser Artikel stellt die nötigen Werkzeuge vor und gibt einen Ausblick auf das Tuning.

Zunächst einmal sollte man sich einen Überblick über die Performance-Situation der Anwendung verschaffen. Apex bietet dafür verschiedene Statistiken im Workspace an, zum Beispiel für die Anzahl von Seiten-Aufrufen oder die Dauer der Ladezeit. Sie lassen sich nutzen, um besonders langsame Anwendungsseiten ausfindig zu machen. So lässt sich auch einschätzen, mit welcher Priorität bestimmte Seiten hinsichtlich eines Performance-Tunings untersucht werden sollten. Wird eine Seite nur wenige Male pro Jahr aufgerufen, sollte ihr natürlich weniger Aufmerksamkeit gewidmet werden als einer Seite mit täglich Tausenden von Aufrufen.

Möchte man dagegen einen Überblick über die Komplexität der Anwendung anhand des Programm-Codes bekommen, kann sich ein Blick in die Apex-eigenen Views lohnen, aus denen anhand der Meta-Daten entsprechende Informationen entnommen werden können. Sie lassen sich, wie auch die Performance-Statistiken, im Workspace finden. Eine praktische Visualisierung der Komplexität der Anwendung anhand eben dieser Views bietet der Apex Visualizer *[1]* von Oliver Lemm, MT AG. Mithilfe von JetCharts stellt dieser unter anderem die Verteilung von PL/SQL, JavaScript und CSS-Code anschaulich dar, womit sich potenziell kritische Komponenten einer Apex-Anwendung identifizieren lassen *(siehe Abbildung 1)*.

Debugging in Apex

Das Debugging in Apex lässt sich über die Entwickler-Toolbar ganz einfach aktivieren. Der Button „Debug“ lässt die aktuelle Seite neu laden und aktiviert den Debugging-Modus. Anschließend werden alle

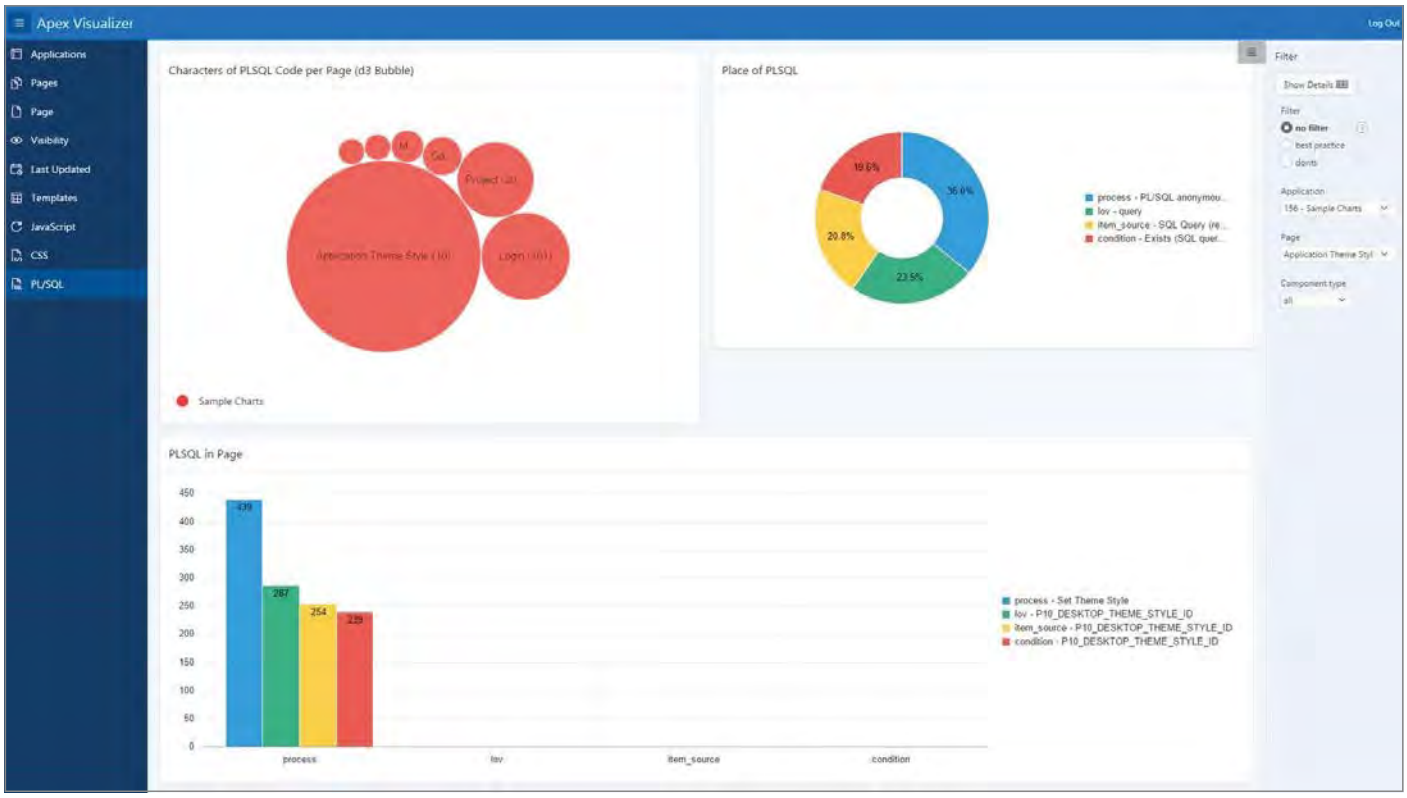


Abbildung 1: Dashboard im Apex Visualizer

Prozesse protokolliert. Neben der veränderten Schaltfläche in der Entwickler-Toolbar, die nun die Möglichkeit bietet, den Debugging-Modus wieder zu verlassen, kann man auch an der URL erkennen, dass der Debug-Modus eingeschaltet ist. Der fünfte Parameter in der URL gibt an, ob das Debugging aktiviert ist, und enthält in diesem Fall ein „YES", beispielsweise „https://apex.oracle.com/pls/apex/f?p=MY_APP:9:116797769453367::YES". Damit lässt sich auch per Hand via URL in den Debugging-Modus wechseln. Möchte man dagegen das Debugging über einen Befehl im eigenen PL/SQL-Code aktivieren, so ist dies mit dem API-Aufruf „apex_debug.enable(p_level)" möglich. An dem Parameter „p_level" erkennt man bereits, dass es mehrere Stufen für das Debugging gibt *(siehe Tabelle 1)*.

Die Stufe 4 ist von besonderer Bedeutung. Sie wird automatisch aktiviert, sobald der Entwickler den Debug-Modus über die Entwickler-Toolbar oder die URL aktiviert. Je höher das Debug-Level gesetzt wird, desto mehr Meldungen werden ausgegeben. Dies sollte auch hinsichtlich der Performance beachtet werden, da sich die vielen Meldungen auch auf die Performance auswirken können.

Hat man schließlich mit einer der genannten Möglichkeiten den Debug-Modus aktiviert, lässt sich mit dem Eintrag „View Debug" in der Entwickler-Toolbar eine Übersicht der zuletzt protokollierten Ereignisse sehen. Das können zum Beispiel AJAX-Aufrufe sein, Plug-ins oder das Seitenladen selbst. Letzteres kann mit dem Eintrag „Show" in der Spalte „Path Info" identifiziert werden. Ruft man ihn auf, bekommt man, je nach gewähltem Debug-Level, eine mehr oder weniger detaillierte Übersicht über alle Prozesse, die beim Laden der Seite abgearbeitet wurden.

Diese Übersicht ist extrem hilfreich, wenn es darum geht, Schritt für Schritt nachzuvollziehen, wie die Seite aufgebaut wird. Dabei ist vom Setzen der NLS-Einstellungen bis hin zum finalen Commit alles aufgelistet. So kann zum Beispiel schnell der Grund für nicht ausgeführte Prozesse aufgrund von nicht erfüllten Bedingungen ermittelt werden. Auch Fehlermeldungen lassen sich der Übersicht schnell entnehmen.

Besonders interessant für das Performance-Tuning ist die Ausführungszeit der einzelnen Verarbeitungsschritte. Der auf der Debugging-Seite dargestellte Graph zeigt einfach, wie die Ausführungszeit verteilt ist. So sind lang laufende Prozesse schnell identifiziert und können hinsichtlich ihrer Optimierungs-Möglichkeiten mit den üblichen Methoden zum SQL-Tuning weiter untersucht werden *(siehe Abbildung 2)*.

Parameter	Funktion
0: Off	Debug ist deaktiviert
1: Error	Kritische Fehler
2: Warning	Weniger kritischer Fehler
4: Info	Wird mit dem Debug-Modus der Toolbar aktiviert, allgemeine Informationen
5 App Enter	Beim Aufruf von Prozeduren/Funktionen
6 App Trace	Weitere Meldungen in Prozeduren/Funktionen
8 Engine Enter	Beim Aufruf von Apex-internen Prozeduren/Funktionen
9 Engine Trace	Weitere Meldungen in Apex-internen Prozeduren/Funktionen

Tabelle 1: Übersicht der Debug-Level in Apex

Eigene Meldungen ausgeben

Neben den Informationen der Apex-Prozesse können auch eigene Meldungen in der Debug-Übersicht ausgegeben werden. Dazu stellt das „Apex_DEBUG"-API verschiedene Methoden bereit *[2]*. Diese sind an die anfangs erwähnten Debugging-Stufen gekoppelt. Wird etwa mit der Prozedur „apex_debug.info()" eine Ausgabe im PL/SQL-Programmcode getätigt, so wird diese nur ausgegeben, wenn mindestens die Debug-Stufe 4, also „Info", aktiviert ist. Möchte man dagegen unabhängig von der gesetzten Debug-Stufe eine Ausgabe tätigen, so hilft „apex_debug.error()" weiter, die immer ausgeführt wird. Neben

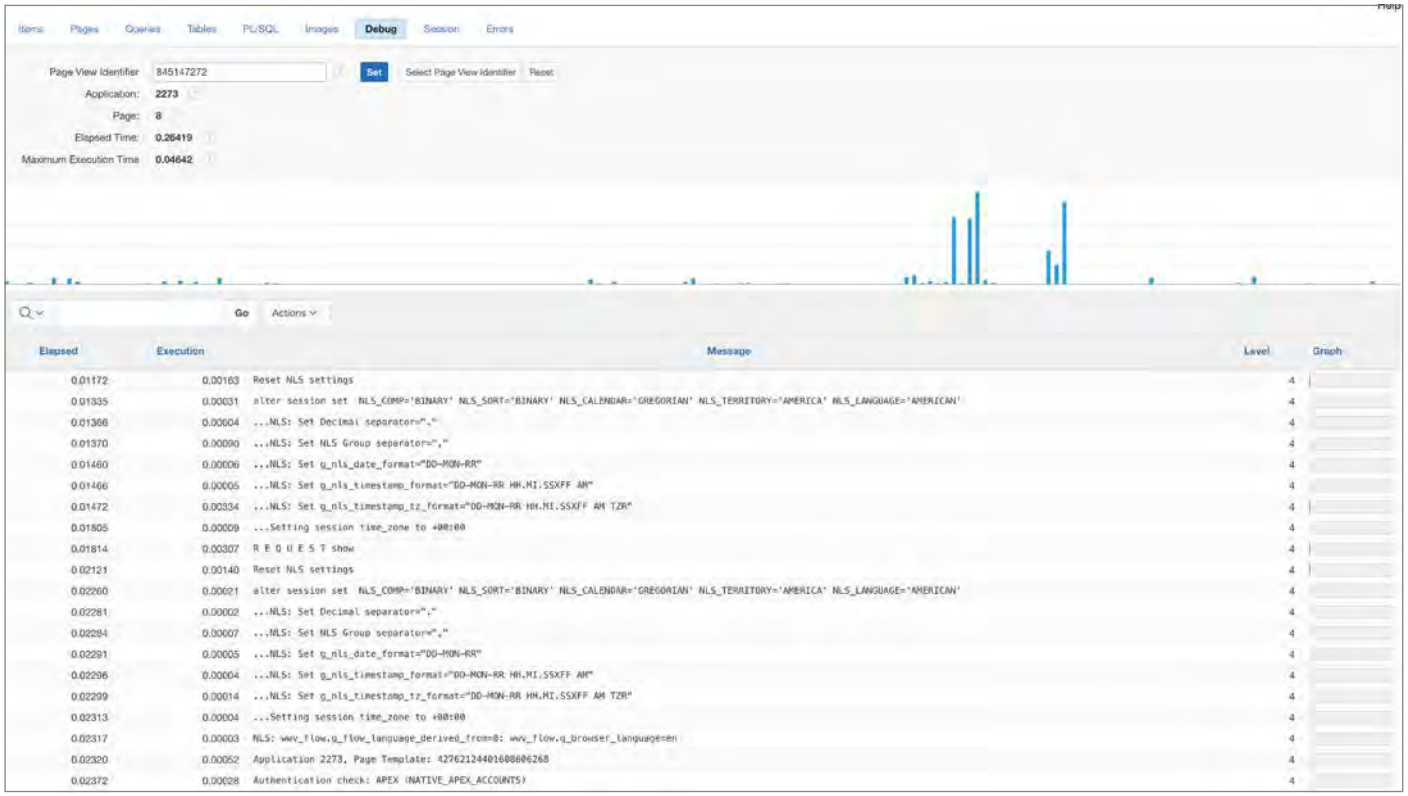


Abbildung 2: Debug-Detailansicht in Apex

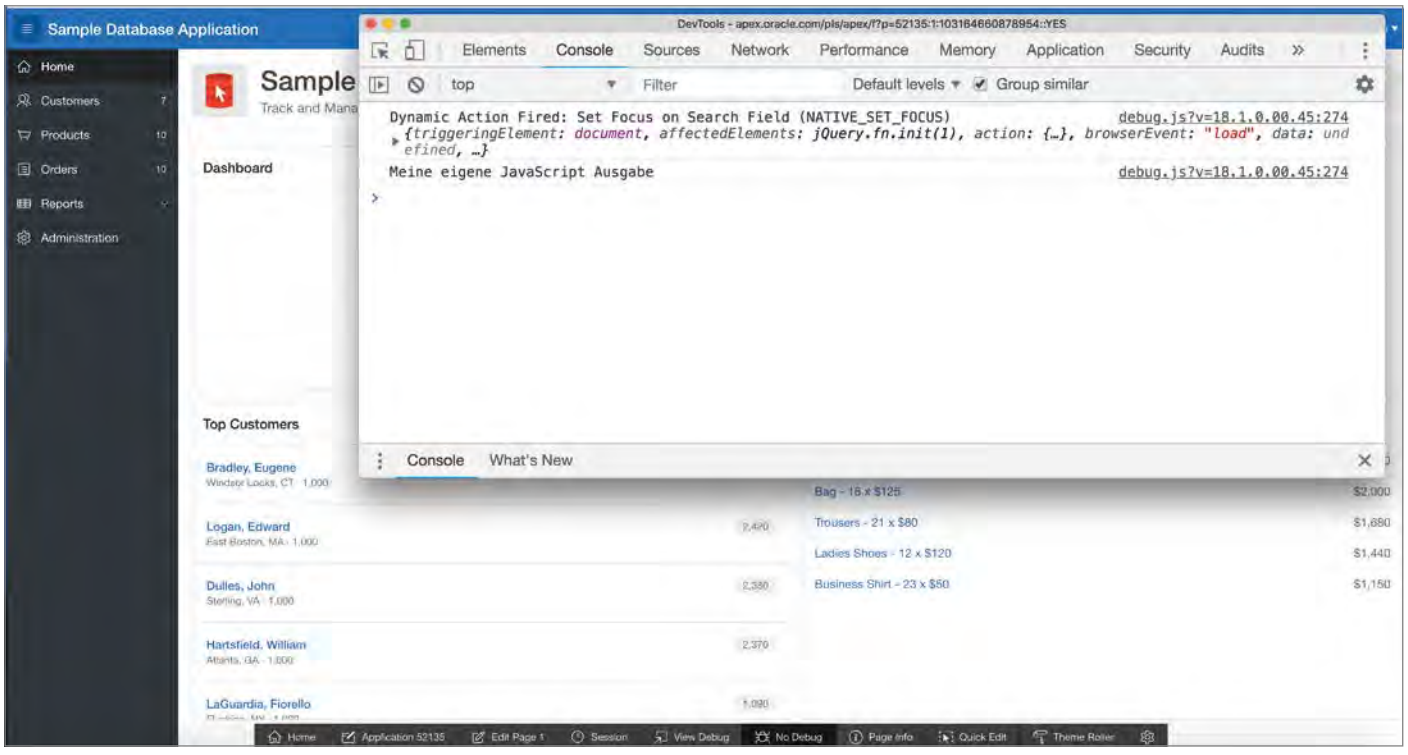


Abbildung 3: Die Entwickler-Tools im Chrome-Browser

der anfangs erwähnten Methode, um das Debugging zu aktivieren, kann dieses mit „apex.debug.disable()“ebenso wieder deaktiviert werden.

JavaScript-Debugging in Apex

Auch wenn Apex zu einem großen Teil aus PL/SQL besteht, sollte auch die clientseitige Verarbeitung betrachtet werden. Mit der aktuell wachsenden Popularität von JavaScript gewinnt dieser Punkt immer mehr an Bedeutung. Das Debugging ist dabei im Kern ähnlich aufgebaut, nämlich ebenfalls an die Debug-Stufen von Apex angelehnt. Das bedeutet konkret: Wird der Debug-Modus von Apex aktiviert, so werden ebenfalls, gemäß den Debug-Stufen, interne Meldungen protokolliert. Diese lassen sich jedoch nicht mit den Apex-eigenen Mitteln zum Debugging betrachten, sondern müssen mit dem Web-Browser eingesehen werden. Dazu stellt jeder moderne Browser Entwicklertools bereit, die sich üblicherweise mit der F12-Taste öffnen lassen. Mithilfe der Konsole in den Entwicklertools kann man nun diese Meldungen einsehen.

Im Gegensatz zu den PL/SQL-Meldungen lassen sich die JavaScript-Meldungen nur einsehen, wenn die aktuelle Seite beziehungsweise der Browser noch geöffnet sind. Wird die Seite verlassen oder neu geladen, werden auch die JavaScript-Meldungen wieder verworfen. Das liegt daran, dass die Meldungen der clientseitigen Verarbeitung nicht mit der Datenbank kommunizieren und somit nicht gespeichert werden.

Die Ausgaben werden direkt in die Konsole des Browsers geschrieben. Um sie permanent nachvollziehen zu können, müsste der Client sie bei jeder Meldung mithilfe eines AJAX-Aufrufs zurück in die Datenbank speichern. Wer diese Funktionalität nicht missen möchte und die bestehenden Ausgaben, mit nützlichen Informationen angereichert, in einem Dashboard auswerten möchte, findet mit dem Tool „LogChase“ *[3]* der MT AG eine geeignete Lösung *(siehe Abbildung 3)*.

Neben den Apex-internen Meldungen lassen sich – wie auch in PL/SQL – in JavaScript Meldungen aus dem eigenen Programmcode ausgeben. Das funktioniert analog zu dem Vorgehen in PL/SQL. Das Apex-JavaScript-API „apex.debug“ *[4]* stellt die dafür notwendigen Funktionen bereit. So lässt sich auch hier mit „apex.debug.setLevel(p_level)“ die Debug-Stufe setzen und zum Beispiel mit „apex.debug.info()“ eine Meldung ausgeben, wenn zumindest die Debug-Stufe 4 gesetzt ist.

Seit Apex 5.1 wird übrigens automatisch eine kleine Benachrichtigung in der Entwickler-Toolbar angezeigt, sobald ein kritischer JavaScript-Fehler aufgetreten ist. Klickt man auf das rote Fehler-Symbol am linken Rand der Entwickler-Toolbar, erfolgt der Hinweis, einen Blick in die Konsole der Entwicklungs-Tools des Browsers zu werfen. Auch wenn diese Meldung leider auch in diesem Fall nicht in die Datenbank protokolliert wird, so wird man als Entwickler zumindest schnell auf einen kritischen Fehler aufmerksam *(siehe Abbildung 4)*.

JavaScript-Performance in Apex

Ebenfalls neu seit Apex 5.1 ist die Möglichkeit, sich die JavaScript-Page-Performance anzeigen zu lassen. Dazu befindet sich in der Entwickler-Toolbar unter dem Eintrag „Page-Info“ ein entsprechender Eintrag. Dieser bietet eine Übersicht über die einzelnen Schritte und Anfragen, die gemacht wurden, um den DOM und alle notwendigen Dateien (JavaScript, CSS, Bilder, Fonts etc.) zu laden. Hier können relativ schnell auffällige Dateien und Vorgänge entdeckt werden. Wer einen noch tieferen Einblick in den genauen Seitenaufbau des DOM erhalten möchte, muss sich des Entwickler-Tools des Browsers bedienen. Dazu eignen sich am besten die Tools von Google Chrome oder Firefox.

In beiden Browsern gibt es in den Entwickler-Tools einen Performance-Tab, mit dem sich die Prozesse sowie Lade- und Ausführungszeiten einer Seite analysieren lassen. Damit können Informationen darüber gewonnen werden, welche Ressourcen das Seitenladen blockieren beziehungsweise verzögern und welcher JavaScript-Code zu viel Zeit benötigt *(siehe Abbildung 5)*.

Allgemeine Tipps zum Tuning in Apex

Nutzt man die zum Debuggen erläuterten Hilfsmittel von Apex, kann man den Auf-

bau von Apex-Seiten einfach nachvollziehen. Dabei sollte man zunächst auf die Prozesse achten, die bei jedem Seitenladen ausgeführt werden. Sie sollten hinterfragt und optimiert werden. Dazu gehören zum Beispiel die Seite 0 oder auch die globalen Anwendungsprozesse aus den gemeinsamen Komponenten. Das gilt auch für Autorisierungs-Schemata. Diese können wahlweise ein Mal pro Session oder bei jedem Seitenladen ausgeführt werden. Letztere Opti-

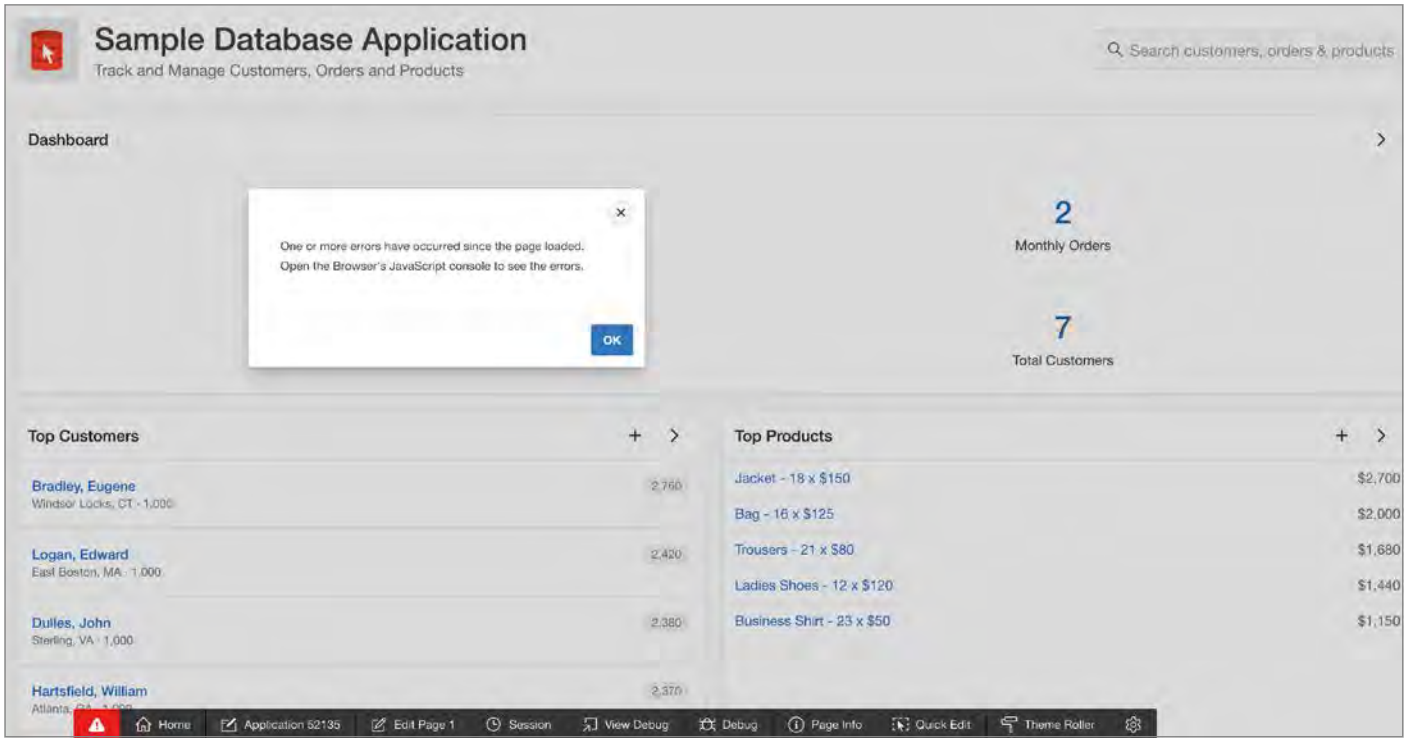


Abbildung 4: Hinweis auf einen JavaScript-Fehler in Apex

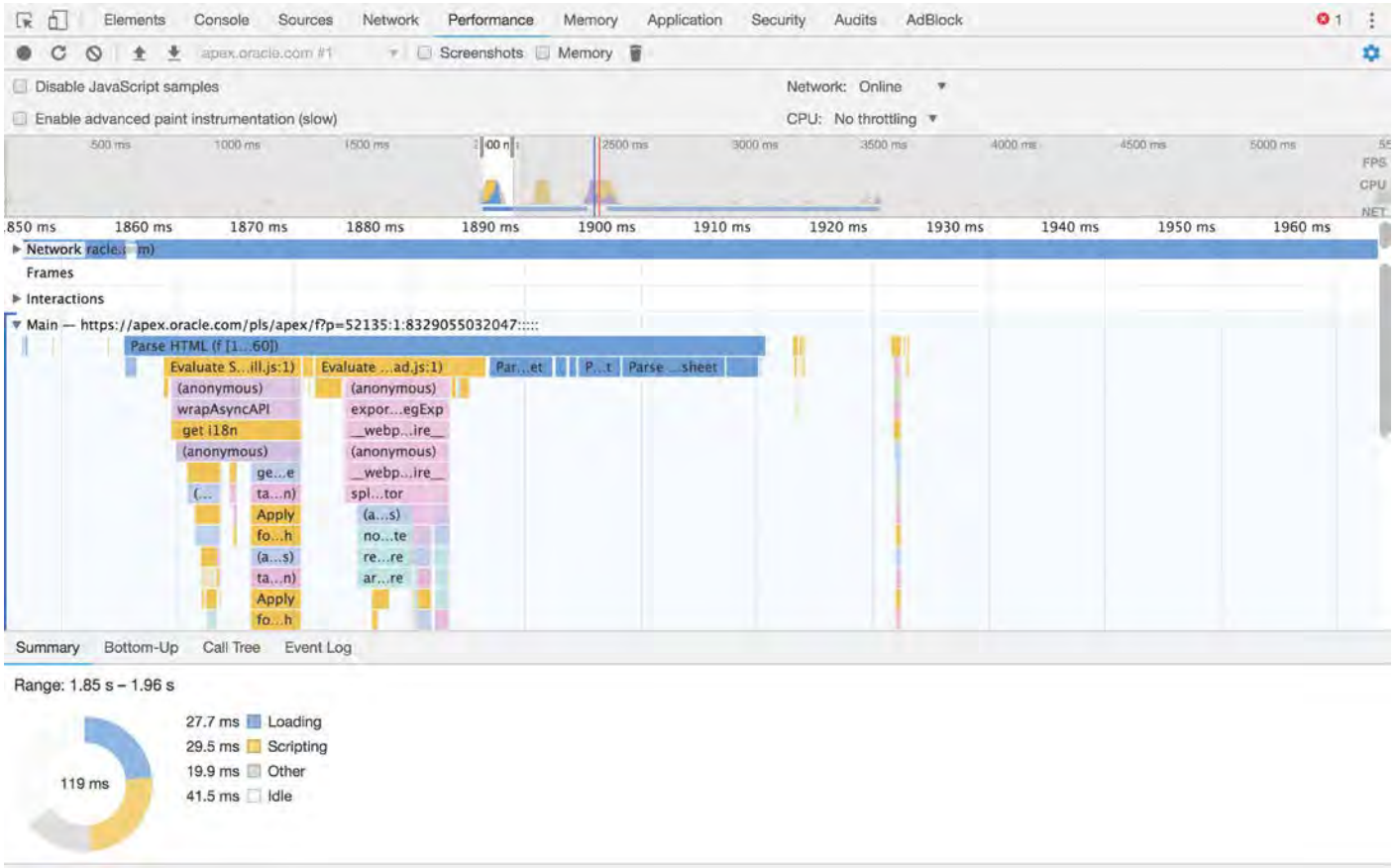


Abbildung 5: Analyse der JavaScript-Performance

on sollte man nur wählen, wenn sich die Bedingung auch wirklich während des Workflows der Anwendung für den Benutzer ändert.

Generell sollte darauf geachtet werden, die Apex-Seiten nicht mit zu vielen Inhalten zu füllen. Je mehr Regionen und komplexere Berechnungen vorhanden sind, desto länger lädt auch die Seite. Enthalten mehrere Regionen innerhalb einer Seite ähnliche Bedingungen, kann es sich lohnen, diese in einem Prozess vor dem Laden der Regionen in ein Page-Item zu speichern und zu referenzieren. Generell gilt: Deklarative Bedingungen sind denen mit PL/SQL-Code vorzuziehen.

Vorsicht ist auch bei der Benutzung von Plug-ins geboten. Während diese auf der einen Seite eine praktische Möglichkeit bieten, um Code zu modularisieren und immer wiederverwenden zu können, können sie ebenso ein Performance-Problem auslösen. Wird ein komplexeres Plug-in auf einer Seite mehrfach genutzt, muss auch mit jeder neuen Instanz der Code immer wieder interpretiert und ausgeführt werden. Eine Lösung für dieses Problem ist das Auslagern des Plug-in-Codes in eigene Datenbank-Packages.

Für Interactive Reports gilt: Dem Benutzer sollten nicht immer alle verfügbaren Optionen zur Verfügung stehen. Diese kann der Entwickler über die deklarativen Eigenschaften eingrenzen. Sind alle Optionen aktiviert, werden sie auch bei jedem Aufruf einer Seite mit geladen und erzeugen zusätzlichen Overhead. Es gilt, die Anforderungen abzustimmen; nicht immer sind Filter-, Pivot- oder Charting-Features erforderlich, es lassen sich so zusätzliche Ressourcen einsparen.

Möglichst sparsam sollte auch mit dem Inline-Code innerhalb Apex umgegangen werden. Generell gilt: Möglichst viel Code sollte in Packages und Views in die Datenbank ausgelagert sein. Das macht den Code nicht nur übersichtlicher und einfacher zu warten, sondern erhöht auch die Performance.

Dasselbe gilt auch für JavaScript-Code. Er sollte ebenfalls in Dateien ausgelagert und über die JavaScript-Referenzen auf Seitenebene in Apex eingebunden sein. Wird neben der normalen Version der Datei auch eine minimierte Version angegeben, wird Apex diese der normalen Version vorziehen, was die Ladezeit der Datei erheblich verringert. Die unkomprimierte Datei wird herangezogen, wenn der Debug-Modus aktiv ist, damit der Code auch vom Entwickler gelesen werden kann.

Wenn SQL-Tuning nicht mehr hilft

Trotz allen SQL-Tunings, manchmal kommt es vor, dass komplexe Abfragen mit großen Datenmengen einfach lange dauern. Hier kann es hilfreich sein, dem Benutzer zumindest das Gefühl zu vermitteln, dass die Seite schneller lädt, als sie es eigentlich tut. Normalerweise werden die Abfragen aller Regionen zunächst ausgeführt, bevor der Benutzer ein erstes Lebenszeichen der Seite zu Gesicht bekommt. Das hat zur Folge, dass der Benutzer zunächst warten muss und kein Feedback bekommt, bis die Abfragen letztendlich ausgeführt wurden.

Eine Möglichkeit, dem Benutzer ein sofortiges Feedback beim Laden der Seite zu geben, besteht darin, die Verarbeitung der Regionen asynchron auszuführen. Somit wird die Seite zunächst nur rudimentär ohne die komplexen Abfragen geladen und erst anschließend werden diese ausgeführt.

Das Interactive Grid verfügt über ein solches Feature, das sich mit dem Attribut „Lazy Loading“ aktivieren lässt. Andere Regionen wie der Classic Report verfügen über ein solches Hilfsmittel nicht. Um dieses Verhalten dennoch zu erreichen, kann der Report mit einer Bedingung über ein Page-Item beim Seitenladen ohne Daten geladen werden. Ist die Seite geladen, werden die Bedingung über das Page-Item auf „wahr“ gesetzt und der Bericht aktualisiert.

Fazit

Dieser Artikel hat vor allem gezeigt, wie die Apex-Anwendung selbst hinsichtlich Performance überprüft und verbessert werden kann. Generell ist Apex jedoch mehr als nur die Anwendung. Daher sollten darüber hinaus immer auch die Datenbank, der Webserver sowie die Kommunikation untereinander untersucht werden, um die beste Performance aus Anwendung und Umgebung herausholen zu können.

Weiterführende Links

[1] *https://github.com/OliverLemm/Apex-Visualizer*
[2] *https://docs.oracle.com/cd/E71588_01/AEAPI/Apex_DEBUG.htm*
[3] *https://apex.mt-ag.com/tools*
[4] *https://docs.oracle.com/cd/E71588_01/AEAPI/apex-debug-namespace.htm*

Till Albert
till.albert@mt-ag.com

Oracle Optimizer System-Statistiken-Update 2018

Randolf Geist, oracle-performance.de

Seit einigen Jahren arbeitet der Oracle Optimizer, der dafür zuständig ist, SQL in einen ausführbaren Plan umzusetzen, mit sogenannten „System-Statistiken" im Rahmen der Kostenabschätzung, die zur endgültigen Planauswahl herangezogen wird. Dieser Artikel geht auf die Grundlagen dieser System-Statistiken ein, erklärt deren Historie und zeigt auf, was sich in den aktuellen Versionen 12c und 18c diesbezüglich getan hat. Zudem wird auch auf die mit Oracle 11.2 eingeführte I/O-Kalibrierung eingegangen, die eine sehr ähnliche Funktionalität bietet.

System-Statistiken sind verschiedene Messwerte, die Auskunft über die CPU- und I/O-Leistung eines Systems geben. Sie werden vom Oracle Optimizer dazu verwendet, die Kosten für Index- und Tabellen-Zugriffe zu berechnen.

Historie der System-Statistiken

Seit Einführung des kostenbasierten Optimizer (Cost Based Optimizer, CBO) in der Version 7 geschieht die Auswahl des auszuführenden Plans, der eine Umsetzung der 4GL-Anweisung in Form von SQL – das nur beschreibt, was man sehen/tun möchte, aber nicht wie – in eine prozedurale 3GL darstellt, in Form der berechneten Kosten; bis auf wenige Ausnahmen wird der Plan mit den geringsten Kosten ausgewählt.

„Kosten" bedeuten in diesem Zusammenhang maßgeblich die Anzahl der notwendigen I/Os – das Kostenmodell des CBO beruht auf der Annahme, dass jeglicher Zugriff auf einen Block I/O notwendig macht – eventuelles Caching im Buffer Cache wird hier also wohl aus Konsistenzgründen nicht berücksichtigt. Die Einheit der berechneten Kosten ist der Einzelblock-Zugriff („Single Block Read"), also der Zugriff auf einen einzelnen Block. Insofern stellt der CBO eine Art Compiler dar, der SQL in einen ausführbaren Pseudocode übersetzt. Dabei entsteht eine Menge möglicher Code-Alternativen, die über die Kostenabschätzung bewertet werden.

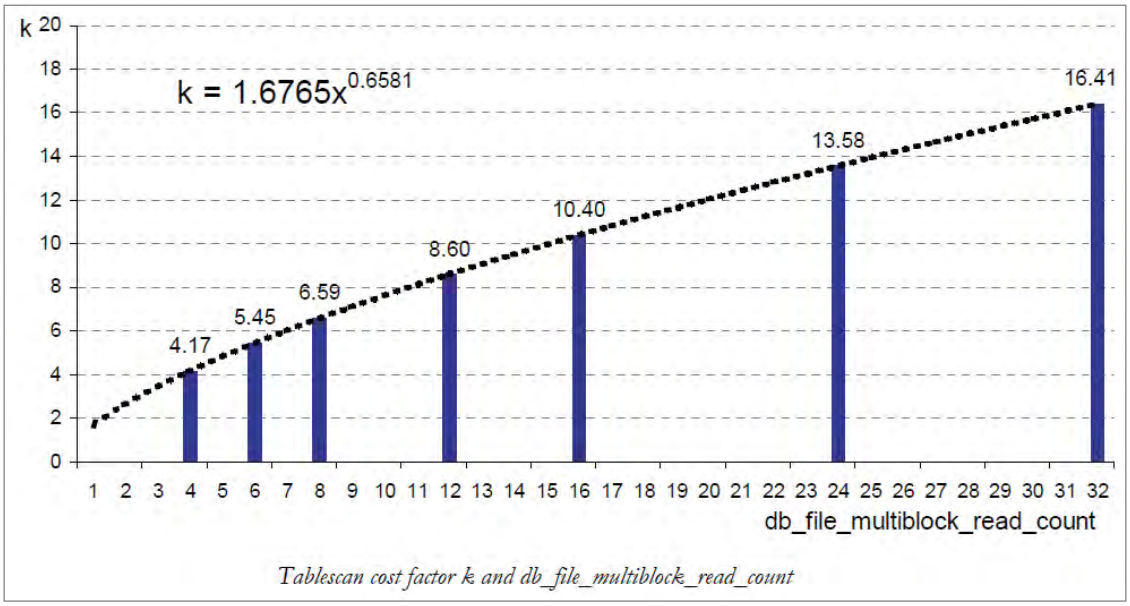


Abbildung 1: Anpassung der Kostenberechnung für Full Table Scans bei höheren Werten von „db_file_multiblock_read_count" vor Verwendung von System-Statistiken, Quelle: „A look under the hood of the CBO" (2004) von Wolfgang Breitling

Wesentliche Anteile der berechneten Kosten beruhen auf Mengen-Abschätzungen und darauf, wie kostspielig die dazugehörigen Operationen sind, zum Beispiel wie häufig eine Schleife – als konkretes Beispiel sei ein Nested Loop Join genannt – durchlaufen werden muss und wie aufwendig eine solche Iteration geschätzt wird, was dann als Multiplikation aus Anzahl der Iterationen und Kosten pro Iteration die Gesamtkosten einer solchen Operation ergibt. Hierbei führt es häufig zu Irritationen und ist daher wichtig zu verstehen, dass insbesondere diese Mengen-Abschätzungen, also beim genannten Beispiel, wie häufig die Schleife durchlaufen werden muss, aus verschiedensten Gründen sehr leicht falsch sein können und daher der CBO Pläne auswählen kann, die sich in der Realität als nicht effizient erweisen.

Grundsätzlich kann man beim heutigen Stand des CBO davon ausgehen: Wären die Abschätzungen korrekt, wäre der ausgewählte Plan – im Rahmen der zur Verfügung stehenden Zugriffsmöglichkeiten – auch effizient. Dies soll aber nicht weiter Thema dieses Artikels sein. Bei weiterem Interesse diesbezüglich verweist der Autor auf seinen Artikel „Cost Based Optimizer Grundlagen", erschienen in der DOAG News, Ausgabe 06/2012, Seite 47, der auf dieses Thema ausführlicher eingeht.

Bevor es System-Statistiken gab, verwendete der CBO für die Kostenberechnung von Tabellen-Zugriffen (Full Table Scans) eine interne Formel, basierend auf dem eingestellten „DB_FILE_MULTIBLOCK_READ_COUNT"-Parameter, um die Kosten eines Full Table Scan im Vergleich zu einem Index-Zugriff zu berechnen. Da diese Berechnung auch kostenmäßig keine Unterscheidung zwischen Einzelblock- und Mehrfachblock-Zugriffen machte – es wurde einfach die Anzahl notwendiger Zugriffe gezählt, wobei ein Einzelblock-Zugriff in diesem Sinne äquivalent mit einem Mehrfachblock-Zugriff war und daher tendenziell Tabellen-Zugriffe gegenüber Index-Zugriffen favorisierte –, kamen mit der Version 8i noch weitere Parameter hinzu, vor allem „OPTIMIZER_INDEX_CACHING" und „OPTIMZER_INDEX_COST_ADJ", die auf verschiedene Art und Weise die berechneten Kosten für Index-Zugriffe beeinflussen beziehungsweise reduzieren können. Dieser Ansatz hatte allerdings verschiedene Nachteile:

- Der eingestellte „DB_FILE_MULTIBLOCK_READ_COUNT" ist nur sehr indirekt eine Maßzahl für die tatsächliche Performance von Full Table Scan. In der Praxis kann zum einen die Anzahl der beim Full Table Scan tatsächlich pro I/O-Request gelesenen Blöcke aus verschiedensten Gründen vom eingestellten Wert deutlich (mehrheitlich nach unten) abweichen, zum anderen kann der Durchsatz solcher Operationen von System zu System je nach eingestelltem Wert und eingesetzter Technologie sehr unterschiedlich sein. Die erwähnte Berechnungsformel versuchte zumindest der ersten Tatsache teilweise Rechnung zu tragen, indem der eingestellte Wert mithilfe einer internen Formel entsprechend angepasst wurde – vereinfacht ausgedrückt, je größer der Wert für „DB_FILE_MULTIBLOCK_READ_COUNT" eingestellt ist, desto stärker die Reduzierung des Werts für die Kostenberechnung; Full Table Scans wurden also bei höher eingestellten Werten nicht im gleichen Maße günstiger. Zum Beispiel wurde für die Anzahl der benötigten I/O-Anfragen bei einem eingestellten Wert von 32 intern stattdessen ein Wert von 16,41 verwendet – die Anzahl der zu lesenden Blöcke also durch 16,41 anstatt 32 geteilt, um die Kosten für die Full-Table-Scan-Operation zu berechnen *(siehe Abbildung 1)*.
- Es gab keine guten Vorgaben für die Parameter „OPTIMIZER_INDEX_CACHING" und „OPTIMZER_INDEX_COST_ADJ", wenn nicht die Standard-Werte verwendet werden sollen. So haben über die Jahre viele Anwender mit diesen Parametern experimentiert. Meistens entstand daraus eine Situation, in der einige SQLs davon profitiert haben und bessere Performance zeigten, während andere deutlich langsamer wurden. Insofern wurden eventuelle Probleme dadurch häufig nur verlagert, aber nicht wirklich gelöst.

- Die beiden Parameter wurden aufgrund des beschriebenen Verhaltens, dass tendenziell Full Table Scans favorisiert wurden, meistens dazu eingesetzt, die Index-Kosten im Vergleich zu Full Table Scans zu reduzieren – den CBO also Index-freundlicher zu konfigurieren. Bei entsprechend extremen Einstellungen – bis heute bei einigen Software-Paketen wie zum Beispiel Siebel vom Hersteller immer noch als Vorgabe empfohlen – kam es häufig zu dem Problem, dass die Kostenabschätzungen für verschiedene Indizes so niedrig berechnet wurden, dass der CBO aufgrund von Rundungen keine sinnvolle Unterscheidungsmöglichkeit hatte und ineffiziente Index-Zugriffe gleich bewertete wie andere, deutlich effizientere Zugriffe. Da unter Umständen die Auswahl nur noch über den Index-Namen geschieht, wenn sich keine anderen Merkmale unterscheiden, konnte es zur Auswahl dieser ineffizienteren Index-Zugriffe kommen.

Aus allen diesen Erfahrungen und Gründen hat Oracle mit der Version 9i die System-Statistiken eingeführt, die diese Probleme lösen sollten. Insbesondere lassen sich darüber die Kosten für Full Table Scans nach oben korrigieren, anstatt die Kosten für Index-Zugriffe zu verringern, was die erwähnten Parameter zur Anpassung der Index-Kosten überflüssig machen sollte. Tatsächlich mag es Situationen geben, in denen zumindest der Einsatz von angepassten „OPTIMIZER_INDEX_CACHING"-Einstellungen aufgrund der Nicht-Berücksichtigung von Caching-Effekten bei der Kostenberechnung immer noch Sinn ergeben kann und das beschriebene Problem der Rundungen bei niedrigen Kosten für Index-Zugriffe umgangen wird – abgesehen davon, dass der CBO bei der Verwendung von System-Statistiken die berechneten Kosten eben nicht mehr rundet.

In Version 9i waren die System-Statistiken noch optional und wurden wahrscheinlich nur sehr begrenzt eingesetzt. Die große Umstellung geschah mit Oracle 10g: Seitdem arbeitet der CBO mit sogenannten „Standard-System-Statistiken", die bei Standard-Einstellungen immer aktiv sind. Da sich aufgrund dieser Veränderung im Standard-Verhalten im Grunde alle Kostenberechnungen im Vergleich zu den Zeiten vor System-Statistiken verändert haben, gab es damals auch massive Probleme beim Upgrade von Oracle 9i nach 10g – manche Leser mögen sich noch an diese Zeit erinnern.

Wenn nicht per Parameter ein 9i-Verhalten des Optimizer erzwungen wurde (per „OPTIMIZER_FEATURES_ENABLE"), änderten sich potenziell sehr viele Ausführungspläne und aufgrund der Vielzahl von Fehlerquellen, die die Berechnungen des CBO beeinflussen können, waren viele dieser Änderungen nicht positiv, es kam also bei vielen Kunden nach dem Upgrade zu Performance-Problemen. Diese Standard-System-Statistiken sind bis heute aktiv und können nur über undokumentierte Parameter oder ein Zurückstellen des Verhaltens per „OPTIMIZER_FEATURES_ENABLE" deaktiviert werden. Dies bedeutet: System-Statistiken sind grundsätzlich relevant, auch wenn man keine aktiven Schritte unternommen hat, um sie zu verwenden.

Wie System-Statistiken funktionieren

Die System-Statistiken umfassen verschiedene Messwerte, die die Fähigkeiten von CPU und I/O beschreiben. Sie erlauben dem CBO, verschiedene Berechnungen anders als zuvor durchzuführen:

- Für Einzelblock- und Mehrfachblock-Zugriffe beschreiben die System-Statistiken, wie lange diese durchschnittlich benötigen („SREADTIM" und „MREADTIM" genannt), also eine Zeitangabe in Millisekunden. Dies löst als Seiteneffekt das beschriebene Problem, da jetzt die beiden Zugriffsarten im Sinne der berechneten Kosten unterschiedlich bewertet werden. Die Annahme ist, dass ein Mehrfachblock-Zugriff länger als ein Einzelblock-Zugriff dauert – was in der Realität nicht immer unbedingt der Fall sein muss, da es zum Beispiel SAN-Systeme mit Read-Ahead-Effekten gibt, bei denen Daten für Full Table Scans schon im SAN-Cache vorliegen und daher sehr schnell gelesen werden können. Die Zeitangaben erlauben dem CBO auch, die geschätzten Kosten in Zeit auszudrücken – dies ist in der Spalte „TIME" eines Ausführungsplans bei Verwendung von System-Statistiken zu finden. Intern wird der Berechnungsmodus „CPU Costing" genannt, siehe nächster Punkt.
- Die CPU-Geschwindigkeit wird auch gemessen („CPUSPEED" beziehungsweise „CPUSPEEDNW") und eine CPU-Kosten- beziehungsweise Zeit-Komponente berechnet, basierend auf den durchgeführten Operationen. Jeder Zeilen- beziehungsweise Spalten-Zugriff benötigt CPU-Zeit, je nach Position der Spalte in einer Zeile auch unterschiedlich. Angewendete SQL-Funktionen (wie „UPPER", „LOWER", „TO_CHAR") werden ebenso im Sinne von CPU-Zeit berücksichtigt. Dies erlaubt dem CBO auch, Filter in einer intelligenten Art und Weise anzuwenden („Predicate re-ordering"), indem diejenigen zuerst ausgewertet werden, die weniger CPU verbrauchen und stärker filtern, und erst danach andere Prädikate, die CPU-intensiver sind.
- Ein weiterer, maßgeblicher Messwert beschreibt, wie viele Blöcke Oracle tatsächlich durchschnittlich bei einem Mehrfachblock-Zugriff lesen kann („MBRC"). Ist dieser Messwert verfügbar, ergibt sich eine Unabhängigkeit der Kostenberechnung von einem eventuell eingestellten „DB_FILE_MULTIBLOCK_READ_COUNT"-Parameter und es wird stattdessen dieser gemessene Wert verwendet. Je nach Konfiguration (Parameter explizit gesetzt oder Standardwert) sowie Art und Weise der Messung hat der „DB_FILE_MULTIBLOCK_READ_COUNT"-Parameter dann doch wieder indirekt einen Einfluss, da bei explizit gesetztem Parameter Oracle diesen in den meisten Fällen als Obergrenze für die Anzahl zu lesender Blöcke pro Operation verwendet.
- Weitere Messwerte beziehen sich auf Parallelverarbeitung („Parallel Execution") und geben an, was als durchschnittlicher I/O-Durchsatz eines einzelnen Parallel-Execution-Servers gemessen wurde und was der maximale Durchsatz des gesamten Systems war. Später mehr dazu, wie diese Parameter die Kostenberechnung beeinflussen können.

Konfiguration der System-Statistiken

Wichtig: Nicht immer sind alle diese Messwerte verfügbar. Recht verwirrend

bei System-Statistiken ist die Tatsache, dass es unterschiedliche Arten gibt, je nachdem, auf welche Art und Weise sie ermittelt wurden („WORKLOAD“ und „NOWORKLOAD“). Für die Konfiguration von System-Statistiken steht ein umfangreiches Interface im „DBMS_STATS“-Paket zur Verfügung. Damit können sie per Messung ermittelt, exportiert, importiert, ausgelesen, gelöscht und auch manuell gesetzt werden.

Die sogenannten „NOWORKLOAD-System-Statistiken“, zu denen auch die mit 10g eingeführten Standard-System-Statistiken gehören, können durch entsprechenden Aufruf im „DBMS_STATS“-Paket („GATHER_SYSTEM_STATS“) gemessen werden. Sie ermitteln nur einen eingeschränkten Umfang an Werten:

- CPU-Geschwindigkeit in Oracle-eigener Einheit („CPUSPEEDNW“)
- Durchschnittliche Zeit zur Festplatten-Kopfpositionierung in Millisekunden („IOSEEKTIM“)
- Durchschnittliche I/O-Datenübertragungsrate von Festplatte in Bytes pro Millisekunde („IOTFRSPEED“)
- Durchschnittliche Anzahl von Blöcken bei Mehrfachblock-Zugriff („MBRC“, nur im neuen „EXADATA“-Modus, später dazu mehr)

Die Messung von NOWORKLOAD-System-Statistiken erfolgt mit einer künstlichen Last, die die Datenbank bei der Messung selbst erzeugt – daher der Name „NOWORKLOAD“, da die Messung nicht auf einem Lastprofil basiert, die eine Applikation verursacht. Idealerweise sollte so eine Messung daher auch auf einem System mit möglichst wenig anderer Last durchgeführt werden, also am besten im Leerlauf.

Im Gegensatz dazu basieren die sogenannten „WORKLOAD-System-Statistiken“ auf der Messung einer tatsächlichen Last auf der Datenbank. Hier wird also von der Datenbank während der Messung (auch mittels „GATHER_SYSTEM_STATS“) keine künstliche Last erzeugt, sondern die tatsächliche Aktivität gemessen. Das heißt auch, dass die Datenbank während einer solchen Messung aktiv sein muss, also ohne eine möglichst repräsentative, durch eine Anwendung erzeugte Last. Dabei werden umfangreichere Messwerte ermittelt:

- CPU-Geschwindigkeit in Oracle-eigener Einheit („CPUSPEED“)
- Durchschnittliche Dauer des Einzelblock-Zugriffs in Millisekunden („SREADTIM“)
- Durchschnittliche Dauer des Mehrfachblock-Zugriffs in Millisekunden („MREADTIM“)
- Durchschnittliche Anzahl von Blöcken bei Mehrfachblock-Zugriff („MBRC“)
- Maximaler I/O-Durchsatz des Systems in Bytes pro Sekunde („MAXTHR“)
- Durchschnittlicher I/O-Durchsatz eines Parallel-Execution-Servers in Bytes pro Sekunde („SLAVETHR“)

Bei der Messung von WORKLOAD-System-Statistiken hängt die Verfügbarkeit mancher Messwerte davon ab, welche Operationen bei den Messungen tatsächlich durchgeführt wurden; erfolgten zum Beispiel keine Mehrfachblock-Zugriffe während einer entsprechenden Messung, sind sowohl der oben erwähnte „MBRC“ als auch „MREADTIM“ nicht in den Systemstatistiken verfügbar. Je nachdem, welche Messwerte fehlen oder auch bestimmten Überprüfungen nicht standhalten (wie zum Beispiel, dass „MREADTIM“ größer als „SREADTIM“ sein muss (siehe oben), was nicht immer in der Realität der Fall ist), werden diese entweder durch Standard-/Ersatz-Werte ersetzt oder die Art der Kostenberechnung fällt auf eine andere Art der Berechnung zurück („NOWORKLOAD“ anstatt „WORKLOAD“).

Die Standard-System-Statistiken umfassen nur einen sehr eingeschränkten Teil dieser Messwerte und gehören zu der „NOWORKLOAD“-Art. Dadurch definieren sie nur die CPU-Geschwindigkeit („CPUSPEEDNW“) und mithilfe einer indirekten Berechnung die Zeiten für Einzelblock- und Mehrfachblock-Zugriffe („SREADTIM“ und „MREADTIM“, berechnet durch Blockgröße, „IOSEEKTIM“ und „IOTFRSPEED“, siehe *Listing 1*).

Wer mehr zu den Details der verschiedenen Arten und der dazugehörigen Berechnungen erfahren möchte, dem sei die (schon recht alte) Blog-Artikel-Serie des Autors empfohlen *(siehe „https://oracle-randolf.blogspot.com/2009/04/understanding-different-modes-of-system.html“)* sowie vor allem Christian Antogninis Buch „Troubleshooting Oracle Performance“, das sogar ein eigenes Kapitel den System-Statistiken widmet und auch aktuellere Entwicklungen berücksichtigt (zumindest inklusive Oracle-Version 12.1).

Empfehlungen/Best Practices bezüglich System-Statistiken

Eines der Probleme im Umgang mit System-Statistiken ist, dass es recht unterschiedliche Aussagen gibt. Als sie

```
Default System Statistics values:
IOSEEKTIM = 10 ms
IOTFRSPEED = 4096 bytes per millisecond = approx. 4 MB per second
db_block_size = 8192 = 8 KB
mbrc = 8 (default used for cost calculation when "db_file_multiblock_read_count" is left unset)
SREADTIM = IOSEEKTIM + db_block_size / IOTFRSPEED

MREADTIM = IOSEEKTIM + mbrc * db_block_size / IOTFRSPEED

SREADTIM = 10 + 8192 / 4096 = 10 + 2 = 12ms

MREADTIM = 10 + 8 * 8192 / 4096 = 10 + 16 = 26ms
```

Listing 1: Beispielhafte Berechnung/Ableitung der „SREADTIM“- und „MREADTIM“-Werte bei Verwendung der Standard-NOWORKLOAD-System-Statistiken, einer Standard-Blockgröße von 8 KB und ungesetztem „DB_FILE_MULTIBLOCK_READ_COUNT“ (wie von Oracle seit Version 10g empfohlen)

```
Full Table Scan costs without System Statistics:
-----------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost  |
-----------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1518 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |       |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1518 |
-----------------------------------------------------------
Full Table Scan costs with default System Statistics:
--------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  2709   (0)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  2709   (0)| 00:00:33 |
--------------------------------------------------------------------------------
```

Listing 2: Beispielhafte Kostenberechnung für einen Full Table Scan mit und ohne System-Statistiken

eingeführt wurden, gab es offizielle Empfehlungen seitens Oracle, sie explizit zu messen, und zwar in Form von WORKLOAD-System-Statistiken, also gemessen basierend auf dem tatsächlichen Last-Profil der Datenbank *(siehe zum Beispiel „https://docs.oracle.com/cd/A91202_01/901_doc/server.901/a87503/stats.htm#28811")*. Christian Antognini stuft in seinem Buch „Troubleshooting Oracle Performance" System-Statistiken als Initialisierungsparameter ein und empfiehlt weiterhin grundsätzlich, diese als WORKLOAD-System-Statistiken zu ermitteln; er geht allerdings detaillierter auf den Prozess ein und stellt verschiedene Alternativen vor.

Die offizielle Sprachregelung seitens Oracle hat sich in der Zwischenzeit geändert – dort empfiehlt man inzwischen, nur noch mit den Standardwerten zu arbeiten und die gemessenen System-Statistiken nur dann zu verwenden, wenn man entweder bereits positive Erfahrungen damit gemacht hat, also eine bereits etablierte Umgebung mit entsprechenden System-Statistiken hat, oder an einem Punkt ist, an dem man die Auswirkungen ausführlich testen kann und wird (zum Beispiel neue Hardware, Upgrade etc.) und dazu bereit ist, den Mehraufwand des Managements von System-Statistiken in Kauf zu nehmen *(siehe zum Beispiel „https://sql-maria.com/2018/04/10/should-you-gather-system-statistics" und „https://blogs.oracle.com/optimizer/should-you-gather-system-statistics")*.

Eines der Probleme beim Messen von System-Statistiken ist, dass die Messwerte sich deutlich unterscheiden können – die Resultate sind also keineswegs konsistent, sondern können stark schwanken, je nach Messzeitpunkt und Aktivität in der Datenbank. Von daher kann man zumindest die Empfehlung aussprechen, System-Statistiken nicht regelmäßig zu aktualisieren, wie es der Autor schon bei einigen Kunden vorgefunden hat, die zum Beispiel einen wöchentlichen oder täglichen Job zur Ermittlung etabliert hatten. Aufgrund der Tatsache, dass System-Statistiken direkte Auswirkungen auf alle Ausführungspläne haben können, führt man auf diese Art und Weise eine potenzielle Instabilität in die Umgebung ein, auf die man möglichst verzichten sollte.

Eine Idee, die Christian Antognini auch als mögliche Alternative empfiehlt, ist die Variante, System-Statistiken regelmäßig zu ermitteln, aber nicht direkt zu aktivieren, sondern in einer entsprechenden Backup-Tabelle zu speichern. Diese Option ist vielen nicht bekannt, wird jedoch von den meisten „DBMS_STATS"-Aufrufen unterstützt. Mit dieser Methode kann man die ermittelten Messwerte auch auf ihre Volatilität hin überprüfen und dann entscheiden, welche Werte am ehesten Sinn ergeben könnten.

Egal für welche Variante man sich entscheidet, klar sollte sein, dass System-Statistiken kein Allheilmittel sind und es meistens eine Gruppe von Abfragen geben wird, die weiterhin nicht optimal performen und um die man sich getrennt kümmern muss. Zwar sollten System-Statistiken in der Theorie dem CBO helfen, auf die jeweilige Umgebung angepasst die effizientesten Ausführungspläne zu finden. In der Praxis ist dies jedoch nicht immer der Fall und der Tatsache geschuldet, dass es viele Parameter gibt, die die Entscheidungen des CBO beeinflussen (System-Statistiken, Objekt-Statistiken, Art der Abfragen und Filter, eventuell Dynamic Sampling/Statistics etc.). Insofern stellen die System-Statistiken nur einen Teil davon dar. Je nachdem, wie weit die anderen abgeschätzten Eingabewerte für die Kostenberechnung von der Realität abweichen, können hier auch Effekte in der Art auftreten, dass eigentlich besser auf die Umgebung abgestimmte System-Statistiken genau den gegenteiligen Effekt erreichen und es im Endeffekt zu weniger effizienten Ausführungsplänen kommt.

Der praktische Nutzen von System-Statistiken

Neben der Berücksichtigung der CPU-Komponente ist also die unterschiedliche und konfigurierbare Bewertung von Einzel- und Mehrfachblock-Zugriffen die wichtigste Neuerung, die mit System-Statistiken eingeführt wurde. Der Hintergedanke bei deren Einführung in Oracle 10g war offensichtlich, die Kostenberechnung des CBO Index-freundlicher zu gestalten. Dieses Ziel wurde grundsätzlich erreicht, so steigen die Kosten für einen Full Table Scan je nach eingestellter/gemessener durchschnittlicher Anzahl von Blöcken bei Mehrfachblock-Zugriff (MBRC) zwischen Faktor 1,8 und 7 im Vergleich zur Kostenberechnung ohne System-Statistiken *(siehe Listing 2)*; mehr Details dazu in der oben genannten Blog-Serie.

Wichtig ist in diesem Zusammenhang zu verstehen, dass es neben der angenommenen Anzahl von Blöcken pro Mehrfachblock-Zugriff (MBRC) maßgeblich das Verhältnis zwischen angenommener Zeit für Einzelblock- und Mehrfachblock-Zugriff („SREADTIM" und „MREADTIM") ist, das für die Kostenberechnung relevant ist, nicht so sehr die absoluten Werte.

Möchte man also die Kostenberechnung Index-freundlicher gestalten, kann man dies über das Verhältnis von „SREADTIM" und „MREADTIM" beeinflussen. Ebenso wird die Kostenberechnung Index-freundlicher, wenn die durchschnittliche Anzahl der Blöcke bei Mehrfachblock-Zugriff (MBRC) niedriger eingestellt wird – auch ein manuelles Setzen von System-Statistiken per „DBMS_STATS.SET_SYSTEM_STATS" ist möglich.

```
select max(n1) from t1;

-------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   278   (0)| 00:00:02 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   278   (0)| 00:00:02 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   278   (0)| 00:00:02 |
-------------------------------------------------------------------------------------

Increased parallel cost at same degree when SLAVETHR is lower than default value,
which is 0.9 * MBRC * db_block_size / MREADTIM bytes per millisecond

-------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   800   (0)| 00:00:05 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   800   (0)| 00:00:05 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   800   (0)| 00:00:05 |
-------------------------------------------------------------------------------------
```

Listing 3: Höhere Kosten bei der Berechnung von Parallel-Execution-Plänen und der Verwendung von „SLAVETHR"-Werten unterhalb des Standard-Werts

Umgekehrt kann man Full Table Scans favorisieren, indem das Verhältnis zwischen „MREADTIM" und „SREADTIM" kleiner und/oder der „MBRC" größer eingestellt wird.

All dies kann im Grunde auch mit dem Parameter „OPTIMIZER_INDEX_COST_ADJ" erreicht werden – allerdings gibt es zwei wesentliche Unterschiede:

- Wenn der CBO mithilfe des Parameters „OPTIMIZER_INDEX_COST_ADJ" Index-freundlicher konfiguriert werden soll, verringern sich die Kosten für Index-Zugriffe. Bei der Anpassung mit System-Statistiken erhöhen sich die Kosten für Full Table Scans – die Kosten für den Index-Zugriff bleiben unverändert.
- Während es keine offiziell dokumentierte Möglichkeit gibt, einen passenden Wert für „OPTIMIZER_INDEX_COST_ADJ" per Messung oder Statistiken festzulegen, können System-Statistiken über ein offiziell dokumentiertes Interface gemessen werden und repräsentieren somit in der Theorie die CPU- und I/O-Möglichkeiten der jeweiligen Umgebung.

Aufgrund der beschriebenen Effekte einer möglichen Volatilität bei der Messung der System-Statistiken, des Zusammenspiels mit anderen Eingabewerten bei der Kostenberechnung (vor allem Mengen-Abschätzungen und Clustering-Faktor von Indizes) und auch der Tatsache, dass das Kostenmodell des CBO grundsätzlich von physischem I/O ausgeht und Caching nicht berücksichtigt, müssen System-Statistiken in der Realität nicht unbedingt zu den erwarteten Verbesserungen bei der Kostenberechnung führen. Gerade Ausführungspläne, die von Caching stark profitieren, können weiterhin falsch vom CBO eingeschätzt werden; daran ändern auch System-Statistiken nichts.

Handelt es sich um eine Umgebung, die auch die Parallelverarbeitungs-Option der Enterprise Edition verwendet („Parallel Execution Features"), können System-Statistiken auch dazu eingesetzt werden, die Kostenberechnung für parallele Ausführungspläne zu beeinflussen. Insbesondere können über die Parameter „SLAVETHR" und „MAXTHR" zu optimistische Kostenberechnungen nach unten korrigiert werden. Ohne diese Parameter nimmt der CBO zum einen an, dass die Parallelverarbeitung im Sinne der Kostenberechnung unendlich skaliert (was auf jeden Fall unrealistisch ist), und zum anderen, dass die Parallelverarbeitung ungefähr mit dem Faktor 0.9 der seriellen Verarbeitung skaliert. Mit „SLAVETHR" lässt sich letzterer Faktor nach unten korrigieren, dem CBO also mitteilen, dass der Faktor niedriger als 0.9 liegt, die Parallelverarbeitung somit schlechter skaliert. Eine bessere Skalierbarkeit als 0.9 kann damit nicht erreicht werden, man kann also den Maximalwert von 0.9 nicht überschreiten – ansonsten wird „SLAVETHR" ignoriert und 0.9 angenommen. Damit können die Kostenberechnungen für Parallelverarbeitungen höher eingestellt werden – der CBO wird also „parallelunfreundlicher" konfiguriert und eventuell verfügbare serielle, Index-basierte Zugriffswege potenziell favorisiert *(siehe Listing 3)*.

„MAXTHR" setzt der standardmäßigen, unendlichen Skalierbarkeit eine Obergrenze – ab einem bestimmten Grad an Parallelität verringern sich dann die berechneten Kosten nicht mehr, was durchaus sinnvoll ist und insbesondere zu verhindern hilft, dass unrealistisch hohe Parallelitätsgrade den CBO einen Ausführungsplan bevorzugen lassen, der in der Realität so nicht skalieren kann. Allerdings gilt dies nur für die Kostenberech-

```
select max(n1) from t1;

Serial cost:
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     5 |  1280   (3)| 00:00:07 |
|   1 |  SORT AGGREGATE    |      |     1 |     5 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 40000 |   195K|  1280   (3)| 00:00:07 |
-------------------------------------------------------------------------------

Parallel(T1 42) cost without MAXTHR set
-----------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |    33   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|    33   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|    33   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Parallel(T1 42) cost with MAXTHR set
-----------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   250   (0)| 00:00:02 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   250   (0)| 00:00:02 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   250   (0)| 00:00:02 |
-----------------------------------------------------------------------------------------
```

Listing 4: Berücksichtigung der limitierten Skalierbarkeit von Parallel Execution bei der Kostenberechnung durch „MAXTHR“

nung – wenn nicht andere Restriktionen über Parameter oder den Resource Manager eingreifen, würde bei der Ausführung eines eventuell dann ausgewählten Ausführungsplans mit hohem Parallelitätsgrad diese dann doch zur Laufzeit angewendet werden *(siehe Listing 4)*.,

Neuerungen bezüglich System-Statistiken

In Bezug auf System-Statistiken haben sich in den letzten Jahren nur wenige Veränderungen ergeben. Die meisten Änderungen haben sich auf Bugs in der Berechnung bezogen. So wurde mit der Version 11.2.0.1 und 11.2.0.2 ein Fehler eingeführt, der völlig unrealistische Werte für die Parameter „SREADTIM“ und „MREADTIM“ bei der Messung von WORKLOAD-System-Statistiken berechnet hat; bei der Berechnung der parallelen Kosten, die „SLAVETHR“ und „MAXTHR“ verwenden, wurde ebenfalls ein fragwürdiger Berechnungsfaktor (und die verwendete Einheit) verändert. Alle diese Fehler sind ab der Version 11.2.0.3 behoben.

Mit der Version 12c (und auch verfügbar ab Version 11.2.0.4) wurden sogenannte „EXADATA-System-Statistiken“ eingeführt. Hier handelt es sich um eine Abwandlung der sogenannten „NOWORKLOAD-System-Statistiken“. Die Einführung dieses neuen Modus hat verschiedene Gründe:

- Der CBO berücksichtigt bei seinen Kostenberechnungen standardmäßig nicht den möglichen Zeitgewinn der Exadata Smart Scans – im Gegensatz zu In-Memory-Scans des In-Memory Column Store, die der CBO explizit entsprechend günstiger bewertet.
- Die Messungen von System-Statistiken funktionieren erfahrungsgemäß auf Exadata-Systemen nicht wirklich gut. Das hat mit technischen Details zu tun, wie gemessen wird und wie Smart Scans auf Exadata funktionieren (sogenannte „Buffer Cache Reads“ vs. „Direct Path Reads“).

Der „EXADATA“-Modus entspricht im Grunde einer Messung der NOWORKLOAD-System-Statistiken, setzt aber zusätzlich den Parameter „MBRC“ (durchschnittliche Anzahl Blöcke bei Mehrfachblock-Zugriff) auf den Wert von „DB_FILE_MULTIBLOCK_READ_COUNT“. Dies verändert die Kostenberechnung also tatsächlich nur dann, wenn der Parameter „DB_FILE_MULTIBLOCK_READ_COUNT“ nicht explizit gesetzt ist, da ansonsten für die Berechnung schon der explizit eingestellte Wert benutzt wurde. Den Parameter nicht explizit zu setzen, ist zwar seit Oracle 10g die offizielle Empfehlung, trotzdem gibt es wohl immer noch

```
Sample Full Table Cost with default System Statistics and "db_file_multiblock_read_count" left unset
 ------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  2716   (1)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  2716   (1)| 00:00:33 |
------------------------------------------------------------------------------

Sample Full Table Cost with EXADATA System Statistics, which sets MBRC to 128 in most cases when using 8 KB
default block size and leaving "db_file_multiblock_read_count" unset
------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1738   (1)| 00:00:21 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1738   (1)| 00:00:21 |
------------------------------------------------------------------------------
```

Listing 5: Verringerte Kosten für einen Full Table Scan bei Verwendung von EXADATA-System-Statistiken

```
TIME represents cost multiplied by SREADTIM without I/O calibration, in this case here 2716 * 12 ms equals ap-
prox. 32.6 seconds
---------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  2716   (1)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T    | 10000 |  2716   (1)| 00:00:33 |
---------------------------------------------------------------------
New calculation of TIME in the presence of I/O calibration results
---------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  2716   (1)| 00:00:06 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T    | 10000 |  2716   (1)| 00:00:06 |
---------------------------------------------------------------------
```

Listing 6: Veränderte Zeit-(„TIME")Berechnung bei Vorhandensein von I/O-Kalibrierungswerten

viele Installationen, die den Parameter angeben, meistens aus historischen Gründen.

Ist der Parameter nicht explizit gesetzt, verwendet Oracle zwei verschiedene Werte für die Kostenberechnung von Full Table Scans und bei der tatsächlichen Ausführung. Die Kostenberechnung verwendet den Wert „8", was relativ wenig ist und daher Full Table Scans relativ teuer macht – es war ja die Intention bei der Einführung von System-Statistiken, die Kostenberechnung standardmäßig Index-freundlicher zu gestalten –, während bei der Ausführung ein deutlich höherer Wert zum Einsatz kommt.

Wenn das Verhältnis zwischen SGA-Größe und maximaler Anzahl konfigurierter Prozesse nicht einen bestimmten Wert unterschreitet, also nicht sehr viele Prozesse für eine recht kleine SGA konfiguriert sind, dann wird bei der Ausführung versucht, typischerweise 1-MB-Mehrfachblock-Zugriffe zu verwenden, was bei einer Blockgröße von 8 KB dem Wert „128" entspricht. Dieser Wert von 1 MB/128 wird dann also im „EXADATA"-Modus der System-Statistiken in den Wert „MBRC" übernommen, was die Kostenberechnung Full-Table-Scan-freundlicher macht. Damit sollten die Exadata und auch andere Systeme mit sehr schnellen Full Table Scans diese in den Ausführungsplänen eher favorisieren.

Sinn ergibt das allerdings nur dann wirklich, wenn man eine Applikation benutzt, die entsprechend ausgelegt ist, also typischerweise eine Data-Warehouse-Anwendung. Da inzwischen auch viele OLTP-Anwendungen oder Mixturen aus beiden auf solchen Plattformen betrieben werden, mag eine solche Favorisierung von Full Table Scans für diese Art der Verwendung nicht unbedingt hilfreich sein *(siehe Listing 5)*.

Historie der I/O-Kalibrierung

Zusätzlich zu den mit Oracle 9i eingeführten System-Statistiken kam zusammen mit dem neuen „Auto DOP"-Feature in Oracle 11.2 die sogenannte „I/O-Kalibrierung". Das „Auto DOP"-Feature dreht sich hauptsächlich um die automatische Berechnung des Parallelisierungsgrads bei Parallel-Ausführungen. Diese Berechnung wiederum fußt maßgeblich auf dem durch I/O-Kalibrierung ermittelten I/O-Durchsatz. Auf den ersten Blick er-

```
Change in behaviour between 11.2 and 12c/18c when using PARALLEL statement level hint

select /*+ parallel */ * from t2 in 11.2.0.4 without I/O calibration results, fallback to default parallel de-
gree, which is 8 in this case here:
-------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  1000K|   113M|   651   (1)| 00:00:08 |
|   1 |  PX COORDINATOR      |          |       |       |            |          |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  1000K|   113M|   651   (1)| 00:00:08 |
|   3 |    PX BLOCK ITERATOR |          |  1000K|   113M|   651   (1)| 00:00:08 |
|   4 |     TABLE ACCESS FULL| T2       |  1000K|   113M|   651   (1)| 00:00:08 |
-------------------------------------------------------------------------------

Note
-----
   - automatic DOP: skipped because of IO calibrate statistics are missing


select /*+ parallel */ * from t2 in 12.2.0.1 without using I/O calibration results, using default values now,
computes a degree of 2 here:
-------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  1000K|   113M|  2600   (1)| 00:00:01 |
|   1 |  PX COORDINATOR      |          |       |       |            |          |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  1000K|   113M|  2600   (1)| 00:00:01 |
|   3 |    PX BLOCK ITERATOR |          |  1000K|   113M|  2600   (1)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| T2       |  1000K|   113M|  2600   (1)| 00:00:01 |
-------------------------------------------------------------------------------

Automatic Degree of Parallelism Information:
--------------------------------------------

   - Degree of Parallelism of 2 is derived from scan of object CBO_TEST.T2_11204

Note
-----
   - automatic DOP: Computed Degree of Parallelism is 2
```

Listing 7: Verändertes Verhalten bei fehlenden I/O-Kalibrierungsergebnissen und Verwendung des „PARALLEL"-Hints auf Statement-Ebene ab Oracle 12c durch Verwendung/Annahme von Standardwerten für die I/O-Kalibrierung

scheint es allerdings merkwürdig, warum diese I/O-Kalibrierung zusätzlich zu den bereits bestehenden System-Statistiken eingeführt wurde – geht es doch bei beiden hauptsächlich um die Messung von I/O-Durchsätzen –, auch wenn sowohl die System-Statistiken als auch die I/O-Kalibrierung noch andere Werte umfassen.

Der Hintergrund dieser Überschneidung kommt wahrscheinlich daher, dass die System-Statistiken nicht unbedingt gemessen werden müssen beziehungsweise deren Messung alle Ausführungspläne beeinflussen kann. Anstatt also das potenzielle Risiko einzugehen, Kunden, die das „Auto DOP"-Feature verwenden wollen, dazu zu zwingen, die System-Statistiken zu ermitteln und damit globale Auswirkungen auf Ausführungspläne zu verursachen, hat man sich wohl für diese separate Messung entschieden, die dann nur für das „Auto DOP"-Feature relevant ist.

Wer genau hinschaut, wird merken, dass sich die Spalte „TIME", also die Zeitabschätzung des CBO im Rahmen der Ausführungsplan-Erstellung, die vor der I/O-Kalibrierung eine direkte Relation zu den Kosten hatte (Kosten mal Zeit für einen Einzelblock-Zugriff („SREADTIM") gleich Zeit), mit Vorhandensein von I/O-Kalibrierungswerten entsprechend verändert – und damit direkten Einfluss auf die Entscheidung hat, ob bei Verwendung von „Auto DOP" eine Ausführung überhaupt parallelisiert wird. Der Parameter „PARALLEL_MIN_TIME_THRESHOLD" steht standardmäßig auf zehn Sekunden. Nur wenn also gemäß der neuen „TIME"-Berechnung eine Ausführung geschätzt länger als dieser Parameterwert dauert, wird Parallelverarbeitung in Betracht gezogen *(siehe Listing 6)*.

Wie die I/O-Kalibrierung funktioniert

Leider ist das Handling der I/O-Kalibrierung bei Weitem nicht so ausgereift wie bei den System-Statistiken, die über das „DBMS_STATS"-Paket ein umfassendes API besitzen. Stattdessen gibt es genau einen Aufruf im Paket „DBMS_RESOURCE_MANAGER", das die I/O-Kalibrierung durchführt, genannt „CALIBRATE_IO". Dessen Parameter sind nicht wirklich gut dokumentiert – man soll die Anzahl der physischen Festplatten angeben, die der Datenbank zur Verfügung stehen, sowie die maximale Latenz für einen Festplat-

tenzugriff. Was der Sinn vor allem des letzteren Parameters sein soll, ist nicht weiter dokumentiert.

Vorausgesetzt, asynchrones I/O ist auf allen Data Files aktiviert – andernfalls gibt es eine Fehlermeldung –, führt ein Aufruf der Prozedur einen künstlichen Benchmark aus und sollte daher zu einem Zeitpunkt durchgeführt werden, zu dem möglichst wenig Aktivität auf dem System ist. Dieser künstliche Benchmark erzeugt sowohl Einzelblock- als auch Mehrfachblock-Zugriffe und ermittelt darüber eine maximale „IOPS"-Rate, den maximalen I/O-Durchsatz und auch eine Latenz. Der Messvorgang kann in „V$IO_CALIBRATION_STATUS" überwacht und beobachtet werden – die Ergebnisse nach Abschluss stehen dann in „DBA_RSRC_IO_CALIBRATE" zur Verfügung.

Interessanterweise ist bisher nur einer dieser Messwerte relevant für die Berechnung des „Auto DOP" – der sogenannte „MAX_PMBPS", was dem I/O-Durchsatz in Megabyte pro Sekunde eines Parallel-Execution-Servers entspricht und damit eigentlich genau dem Messwert „SLAVE-THR" der System-Statistiken (auch wenn die Einheit unterschiedlich ist, Bytes pro Sekunde vs. Megabyte pro Sekunde).

Leider hat die I/O-Kalibrierung in der Praxis ähnliche Probleme wie die Messung der System-Statistiken – die Ergebnisse können sehr stark schwanken und vor allem auf Systemen mit sehr schnellem I/O, wie zum Beispiel Exadata, nicht wirklich repräsentative Werte ermitteln. Zusätzlich ist wichtig zu verstehen, dass Änderungen an der I/O-Kalibrierung nur nach Neustart der Instanz gültig werden – im Gegensatz zu System-Statistiken, die jederzeit online verändert werden können und sich sofort auswirken.

Ein weiterer signifikanter Unterschied ist das fehlende API zur Pflege der I/O-Kalibrierungswerte – so empfiehlt Oracle in Versionen vor 12c im Falle von ungünstig ermittelten Werten oder generell auf Exadata-Systemen beziehungsweise Systemen mit schnellem I/O, die I/O-Kalibrierungswerte manuell zu setzen. Allerdings ist dieses manuelle Setzen im Gegensatz zu den System-Statistiken, bei denen Werte direkt mit entsprechenden „DBMS_STATS.SET_SYSTEM_STATS"-Aufrufen gesetzt werden können, nur durch die direkte Manipulation einer „SYS"-Tabelle („SYS.RESOURCE_IO_CALIBRATE$", die Tabelle hinter dem View „DBA_RSRC_IO_CALIBRATE") möglich – ein eher ungewöhnliches Vorgehen, das zeigt, dass ein diesbezügliches offizielles API hilfreich wäre.

Neuerungen bei der I/O-Kalibrierung

Ähnlich wie bei den System-Statistiken und Oracle 10g hat Oracle mit der Version 12c sogenannte „Standardwerte" für die I/O-Kalibrierung eingeführt. In Versionen vor 12c mussten diese Werte vorhanden sein – entweder echt gemessen oder eben manuell per „DELETE/INSERT" gesetzt, um die „Auto DOP"-Berechnung verwenden zu können. Ansonsten wurde eine Anmerkung bei der Planerstellung generiert, die darauf hingewiesen hat, dass die Werte fehlen, und der Parallelitätsgrad wurde über die bisherigen („Manual DOP"-) Verfahren festgelegt.

Ab Oracle 12c wird bei nicht vorhandenen Werten einfach der bisher für Exadata-Systeme empfohlene Wert von 200 MB pro Sekunde für den Messwert „MAX_PMBPS" angenommen und die Berechnung durchgeführt. Damit man diesen Wert nicht manipulieren muss, wurde zusätzlich ein neuer Parameter „PARALLEL_DEGREE_LEVEL" eingeführt, mit dem man die „Auto DOP"-Berechnung entsprechend verändern kann. Der Standardwert beträgt „100", Werte kleiner als „100" ergeben niedrigere Parallelitätsgrade als normal, höhere Werte ergeben höhere Grade.

Ein Seiteneffekt der neuen Standardwerte für die I/O-Kalibrierung ist, dass SQLs, die einen „PARALLEL"-Hint auf Statement-Ebene beinhalten (ohne weitere Angabe, also einfach „SELECT /*+ PARALLEL */ …") ab Version 12c auch ohne I/O-Kalibrierungswerte die „Auto DOP"-Berechnung aktivieren, während in älteren 11.2-Versionen die fehlenden Werte eben zu dem Rückfall auf „Manual DOP" führen. Eine signifikante Änderung im Verhalten, die bei Entwicklern, die sich über die genaue Auswirkung des Hints nicht klar waren, zu Überraschungen bei einem Upgrade führen kann *(siehe Listing 7)*.

Randolf Geist
randolf.geist@oracle-performance.de

Oracle erwirkt dauerhafte Verfügung gegen Rimini Street

Der Federal Court in Nevada hat entschieden, dass Oracle zu einer dauerhaften Verfügung gegen Rimini Street berechtigt ist. Der Drittanbieter für Software-Wartung und Support-Dienstleistungen wurde außerdem zur Zahlung von rund 30 Millionen US-Dollar verpflichtet.

Rimini Street habe seine Geschäfte aufgebaut, indem es die urheberrechtlich geschützte Software von Oracle verletzte, so das Gericht. Weiterhin erhielt Oracle in der letzten Phase des acht Jahre anhaltenden Rechtsstreits über 28 Millionen US-Dollar an Anwaltsgebühren zugesprochen. Die Fehde hatte ursprünglich damit begonnen, dass Oracle behauptete, Rimini Street sei durch illegalen Zugriff auf die technischen Support-Websites von Oracle an „massivem Diebstahl" beteiligt gewesen.

Weitere Informationen unter *„https://www.oracle.com/corporate/pressrelease/rimini-street-081418.html"*.

Next Generation Engineered Systems X7 im Überblick

Frank Schneede, ORACLE Deutschland B. V. & Co. KG

Der Generationswechel auf die Oracle-Engineered-Systems-Familie X7 wurde am 3. Oktober 2017 mit der offiziellen Vorstellung der Exadata Database Machine X7-2 eingeläutet. Mancher Kunde mochte die aktuelle Generation der Engineered Systems im ersten Moment als das für System-Updates übliche „Schneller-Höher-Weiter" abtun, aber die Generation X7 hat bei genauerem Hinsehen weit mehr als nur stärkere Leistung zu bieten. In der Tat handelt es sich nämlich um eine der umfangreichsten Überarbeitungen der Hardware-Architektur seit langer Zeit. Dieser Artikel stellt die neue System-Generation der Engineered Systems vor. Dabei liegt der Schwerpunkt auf den Systemen, die primär als Plattform für die Oracle-Datenbank eingesetzt werden.

Wie bereits mehrfach in der Vergangenheit geschehen, standen die neuen Modelle der Exadata Database Machine X7-2, die üblicherweise als erste Maschine einer neuen Engineered-Systems-Generation vorgestellt wird, wenige Wochen nach der Veröffentlichung der aktuellen Intel-Chip-Generation mit „Skylake"-Architektur zur Verfügung. *Abbildung 1* bietet einen Überblick über die Evolution der Exadata-Maschinen seit ihrer ersten Vorstellung im Jahr 2008.

Exadata Database Machine X7-2

In den Datenbank-Servern der Exadata kommt nun der „24 Core Intel Xeon Platinum 8160"-Prozessor (2,1 GHz) zum Einsatz. Dieser verfügt nur über wenige Cores mehr, bietet aber eine 20 bis 40 Prozent höhere Leistung für Analytics- und OLTP-Workloads gegenüber dem Vorgänger. Mit neuen Ethernet-10/25-Gb/s-Anschlüssen steht eine State-of-the-Art-Netzwerk-Anbindung zur Verfügung. Es ist damit möglich, folgende Anschlüsse zu realisieren:

- 2 x 1G/10G Base-T Ethernet Ports oder 2 x 10G/25G Ethernet SFP28 Ports LAN auf dem Motherboard (LOM), basierend auf dem Broadcom Limited BCM57417 NetXtreme-E 10G/25G RDMA Ethernet Controller
- 2 x 10G/25G Ethernet SFP28 Ports Add in Card (AIC), basierend auf dem Broadcom Limited BCM57414 NetXtreme-E 10G/25G RDMA Ethernet Controller

Die Datenbank-Server werden standardmäßig mit 384 GB (12 x 32-GB-DDR4-DRAM-Module) ausgeliefert. Es besteht die Möglichkeit, bereits ab Werk die Maschinen mit 768 GB (12 x 64GB-DDR4-DRAM-Module) ausrüsten zu lassen. Bei einem absehbaren Ausbau der Exadata Database Machine empfiehlt es sich, aus Kostengründen bereits ab Werk die Maschine mit 64-GB-Modulen auszustatten, denn bei der späteren Aufrüstung müssen die installierten 32-GB-Module komplett gegen die größeren 64-GB-Module ausgetauscht werden – ein gemischter Betrieb der Module ist nicht möglich.

Die weiteren Änderungen der neuen Exadata-Generation X7-2 beziehen sich auf die Storage-Server. Darin werden nun „10 Core Intel Xeon Silver 4114"-Prozessoren (2,2GHz) verwendet. Der Hauptspeicher ist mit 192 GB (12 x 16 GB DDR4 DRAM) im Vergleich zum Vorgänger um 50 Prozent vergrößert worden. Der Hauptspeicher der Exadata-X7-Storage-Server kann darüber hinaus mithilfe der gleichen Upgrade-Kits (12 x 64-GB-DDR4-DRAM-Module) der Datenbank-Server auf bis zu 1.536 GB ausgebaut werden. Das ist in den Fällen ratsam, in denen der Hauptspeicher der Storage-Server als erweiterter Cache für In-Memory-OLTP genutzt werden soll.

Eine besonders interessante Änderung in den Storage-Servern ist die Verwen-

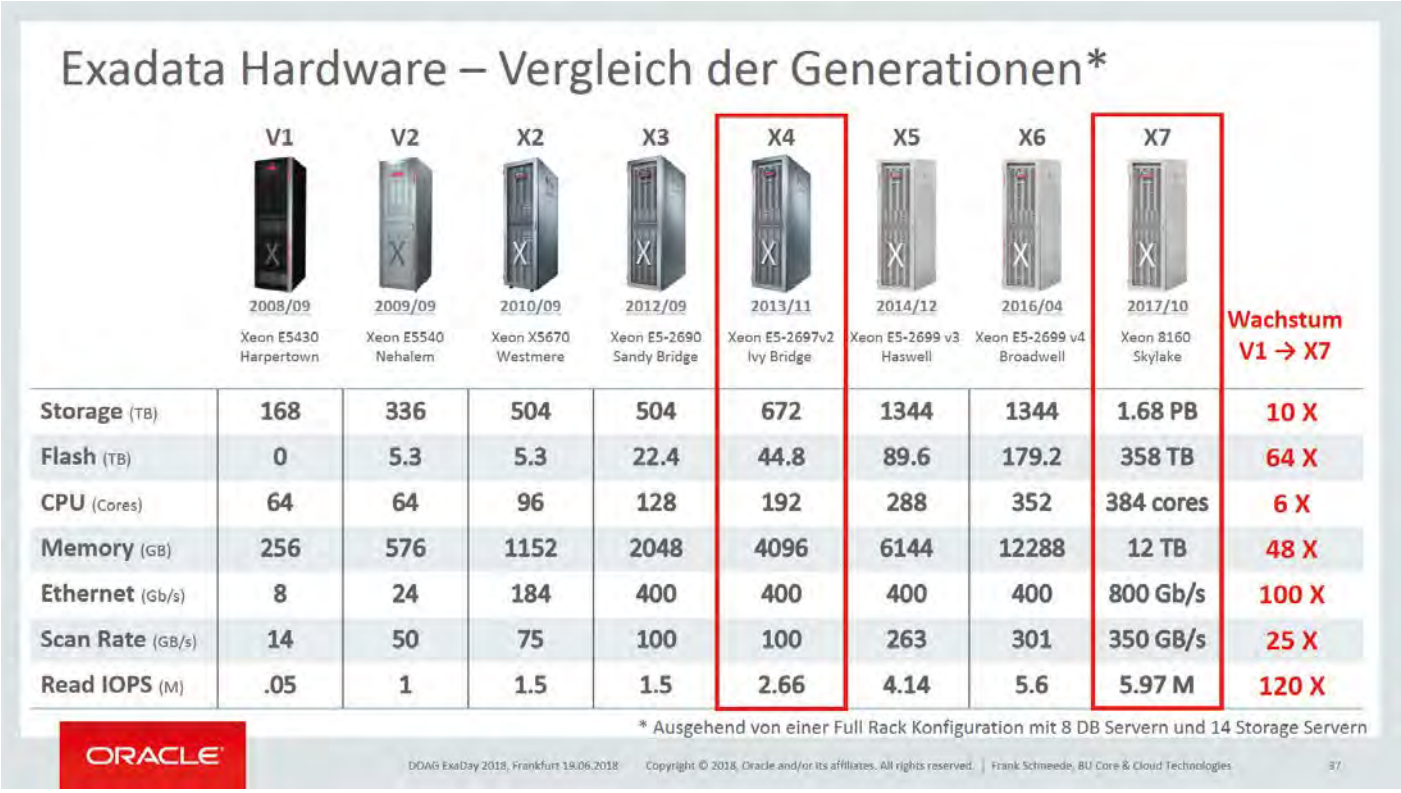


Abbildung 1: Exadata-Überblick der Generationen

dung von zwei 150-GB-M.2-Drives, die für Boot- und Wiederherstellungs-Operationen genutzt werden. Dadurch werden die zur Wiederherstellung der Storage-Server bisher genutzten USB-Sticks obsolet und es ist nicht mehr notwendig, dass auf den ersten beiden Festplatten der Exadata-Storage-Server X7-2 HC eine Systempartition eingerichtet wird. Entsprechend entfällt auch die bisherige standardmäßige Einrichtung einer separaten Diskgroup (meistens „dbfs_dg“) auf den verbleibenden zehn Festplatten. Folglich steht nun der gesamte Festplattenplatz für die Verteilung auf verschiedene Diskgruppen zur Verfügung. Sobald Exadata-Storage-Server X7-2 HC für Erweiterungen älterer Maschinen genutzt werden, ist natürlich die Existenz der Systempartitionen der alten Maschinen beim Layout der Diskgruppen zu berücksichtigen.

Als Festplatten sind im High-Capacity-Modell nun 20 Prozent größere, Helium-gefüllte Festplatten von 10 TB Größe verbaut. Somit hat ein Storage-Server HC 120 TB Rohdaten-Kapazität; abhängig von der genutzten ASM-Redundanz ergeben sich so ca. 45,5 TB (Redundanz „normal“) beziehungsweise ca. 35,6 TB (Redundanz „high“) Nutzerdaten-Kapazität. Die verwendeten Festplatten haben eine Baugröße von 3,5 Zoll und können nicht als „Disk-Swap“ für ältere Storage-Server genutzt werden. Bei der Erweiterung einer alten Maschine mit Storage-Servern X7-2 ergibt sich somit ein Verschnitt von 2 TB pro Festplatte, der entweder ungenutzt bleibt oder – sofern mindestens drei Storage-Server verbaut sind – auch als neue Diskgroup konfiguriert werden kann.

Als Flashmodule kommen nun „6,4 TB Flash Accelerator F640 PCI“-Cards (NVMe-Protokoll) zum Einsatz, deren Kapazität sich gegenüber dem Vorgängermodell verdoppelt hat. Somit stehen pro Storage-Server 25,6 TB Rohdaten-Kapazität im Flash zur Verfügung. Im Exadata Storage Server X7-2 EF finden sich in Bezug auf Prozessor, Hauptspeicher und System-Partitions die gleichen Änderungen wie im Exadata Storage Server X7-2 HC. Als Flash-Drives werden „6,4 TB 2,5 Zoll Flash Accelerator F640 PCIe“-Drives (NVMe-Protokoll) genutzt, sodass sich pro Storage-Server EF eine Rohdaten-Kapazität von 51,2 TB ergibt; abhängig von der genutzten ASM-Redundanz ergeben sich so ca. 18,8 TB (Redundanz „normal“) beziehungsweise ca. 14,7 TB (Redundanz „high“) Nutzerdaten-Kapazität. Der neue Cisco-Nexus-93108TC-EX-1G-Switch dient der Anbindung der Einzelkomponenten an das Management-Netzwerk.

Exadata Database Machine X7-2 HC Eighth Rack

Die kleinste Ausbaustufe der Exadata Database Machine ist nach wie vor das Eighth Rack, das aus zwei Datenbank-Servern, drei Storage-Servern und den zugehörigen Netzwerk-Komponenten besteht. In den Datenbank-Servern ist nur jeweils ein Sockel mit dem „24 Core Intel Xeon Platinum 8160“-Prozessor (2,1 GHz) bestückt. Das bedeutet, dass die Datenbank-Server einen maximalen Hauptspeicher von 768 GB unterstützen, der so jedoch nicht ab Werk bestellt werden kann. In den Storage-Servern HC des Eighth Rack sind lediglich sechs Festplatten verbaut, die für das Rack insgesamt eine Rohdaten-Kapazität von 180 TB beziehungsweise 68,2 TB oder 53,5 TB Datenkapazität ergeben. Die Anzahl der „6,4 TB Flash Accelerator F640 PCI“-Cards (NVMe-Protokoll) ist ebenfalls auf zwei Karten reduziert.

Eine wesentliche Änderung der neuen Maschinen-Generation ist die Ausbaufähigkeit des Exadata Database Machine X7-2 HC Eighth Rack. Es wird nach wie

vor das Exadata Database Machine X7-2 Eighth Rack to Quarter Rack Database Server Upgrade angeboten, in dem zur Erhöhung der Rechenleistung der zweite Sockel der Datenbank-Server bestückt und die entsprechenden Prozessorkerne freigeschaltet werden können. Ein Pendant aufseiten des Storage (HC Modell) gibt es jedoch nicht.

Zur Erweiterung des Storage eines Eighth Rack wurde hingegen der neue Exadata Eighth Rack Storage Server X7-2 High Capacity (HC) eingeführt, der über die in diesem Abschnitt beschriebenen Komponenten verfügt. Mit diesem neuen Storage Server kann der Kunde in kleinen Schritten seinen zur Verfügung stehenden Festplatten-Speicher bedarfsgerecht ausbauen, ohne wie beim Vorgängermodell in einem Schritt 18 Exadata-Storage-Server-Software-Lizenzen erwerben zu müssen.

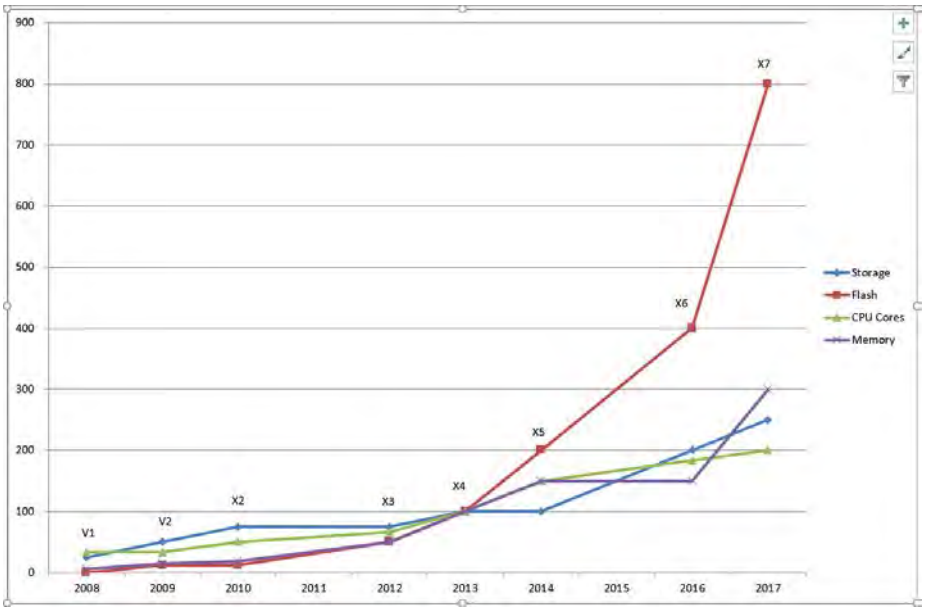


Abbildung 2: Kapazitätszuwachs von Exadata X4-2 zu Exadata X7-2

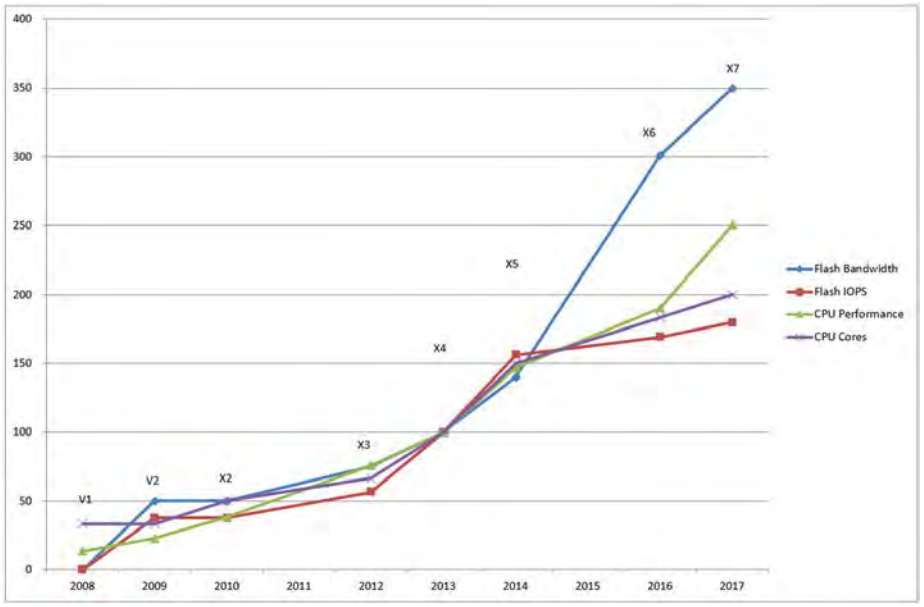


Abbildung 3: Leistungszuwachs von Exadata X4-2 zu Exadata X7-2

Exadata X7-2 im Leistungsvergleich

Durch die immer leistungsstärkeren Prozessoren, größere und schnellere Flash-Module sowie das optimierte Zusammenspiel aller Komponenten wird vermehrt auf die kleineren Exadata-Modelle zurückgegriffen; vor allem, wenn es darum geht, ältere Maschinen zu ersetzen, versuchen viele Kunden, durch entsprechendes Downsizing bei gleicher Systemleistung Kosten zu sparen.

Zum jetzigen Zeitpunkt laufen bei zahlreichen Kunden Exadata-X4-2-Maschinen, die zwischen Ende 2013 und Ende 2014 ausgeliefert worden sind, aus der Abschreibung, weshalb deren Ersatz geplant wird. Ein Beispiel verdeutlicht den zum Teil dramatischen Zuwachs an Kapazität und Leistung von Exadata X4-2 zu Exadata X7-2. *Abbildung 2* zeigt den Kapazitätszuwachs in Bezug auf CPU-Cores, Storage, Memory und Flash. Das Modell X4-2 stellt in der Abbildung den Bezugspunkt mit 100 Prozent dar.

Während die Anzahl der nutzbaren Prozessorkerne sich von X4-2 zu X7-2 nur verdoppelt hat, sind Memory um Faktor drei und Flash sogar um Faktor acht gewachsen. Ein ähnliches Bild zeigt sich in Hinblick auf den Zuwachs an Leistung *(siehe Abbildung 3)*.

Betrachtet wurden in *Abbildung 3* die Performance-Daten gemäß Oracle-Datenblatt, die CPU-Performance wurde anhand der „SPECint_rate2006" verglichen. Die Flash-IOPS sind mit einem Zuwachs von 80 Prozent noch vergleichsweise gering ausgefallen, die CPU-Performance hat sich mit dem Faktor 2,5 mehr als verdoppelt, die Flash-Bandbreite hat sich sogar um Faktor 3,5 gesteigert.

Bedeuten die vorgestellten Kennzahlen nun, dass man einfach beim Ersatz eines Exadata X4 2 Quarter Rack heutzutage nur noch ein Exadata X7 2 Eighth Rack benötigt, wie es die Schaubilder nahelegen könnten? Die Antwort auf diese Frage lautet eindeutig „Nein", denn man darf nicht vergessen, dass die Performance von Exadata nur im Zusammenspiel aller Komponenten erreicht werden kann. So ist im Exadata X7 2 Eighth Rack nur ein Sockel bestückt und es drehen sich nur sechs Festplatten pro Storage-Server – die Performance eines Exadata X4 2 Quarter Rack kann demnach schon aus rein physikalischen Gründen nicht erreicht werden.

Zudem darf man bei dieser Betrachtung ebenfalls nicht die individuelle Transaktions- oder Abfrage-Last vergessen, die in der Regel auch verschiedene Engpässe im System aufzeigen kann. Daher sollte bei einem Replacement eines älteren Exadata-Modells immer auch eine

Performance-Auswertung in Form eines AWR-Reports zurate gezogen werden. Im Zweifelsfall kann man durch den Ersatz mit einem neueren Exadata-Modell gleicher Ausbaustufe und die Nutzung von Funktionen wie „Capacity on Demand" trotzdem einige Kosten sparen und dabei gleichzeitig eine Steigerung der System-Leistung erreichen.

Einordnung des Exadata X7-2 HC Eighth Rack

Oracle hat das Eighth Rack erstmals im September 2012 eingeführt, um Kunden ein minimales Exadata-System zum Einstieg anbieten zu können, von dem es einen durchgängigen Wachstumspfad bis hin zu einer Multirack-Konfiguration gibt. Das Eighth Rack erfreute sich bei den Kunden schnell großer Beliebtheit, allerdings wurden nur sehr wenige Systeme tatsächlich über ein Quarter Rack hinaus aufgerüstet.

Um den Kunden einen Anhalt dafür zu geben, wie der Einstieg in die Exadata-Technologie, Renewal oder Upgrade einer bestehenden kleinen Exadata-Umgebung sinnvoll gestaltet werden können, sollte man eine Positionierung des Exadata X7-2 HC Eighth Rack vornehmen. Die folgende Aufstellung bietet einen Überblick darüber, in welchen Szenarien sich ein Eighth Rack anbietet:

- Erstbeschaffung
  - Eighth Rack für Test- und Entwicklungs-Umgebungen
  - Eighth Rack für Produktions-Umgebungen mit klar definierten Anforderungen (definiertes geringes Wachstum)
  - Für wachsende Umgebungen besser Quarter Rack (mit CoD)
- Upgrade Eighth Rack
  - Eighth Rack Storage durch Eighth Rack Storage Server X7-2 HC erweitern
  - Datenbank-Server-Upgrade durch Einbau zweiter CPU komplex (Neuinstallation)
- Renewal
  - Eighth Rack durch Eighth Rack X7-2 (mit CoD) ersetzen
  - Quarter Rack durch Quarter Rack X7-2 (mit CoD) ersetzen
  - Kein Downsizing vornehmen
  - Gegebenenfalls Virtualisierung zur Optimierung der Lizenzkosten prüfen

Oracle Database Appliance X7-2

Aus der Oracle Database Appliance (ODA) ist in der sechsten Generation mittlerweile eine Familie von Systemen geworden, die aus folgenden Maschinen besteht:

- Oracle Database Appliance X7-2S
  - Single-Instance
  - SE/SE1/SE2 or EE
  - Virtualization
  - 10 Cores
  - 12.8 TB Data Storage (Raw)
- Oracle Database Appliance X7-2M
  - Single-Instance
  - SE/SE1/SE2 or EE
  - Virtualization
  - 36 Cores
  - Bis zu 51.2 TB Data Storage (Raw)
- Oracle Database Appliance X7-2HA
  - RAC, RAC One, SI
  - SE/SE1/SE2 or EE
  - Virtualization
  - 72 Cores
  - Bis zu 128TB SSD or 300 TB HDD Data Storage (Raw)

Die neue Generation verfügt wie die Exadata X7-2 über verschiedene Intel-Xeon-Prozessoren der neuen „Skylake"-Architektur. Die *Tabelle 1* bietet einen Überblick über die neuen Modelle der Oracle Database Appliance X7-2. Durch den Generationswechsel auf das Modell X7 bestehen mehr Auswahlmöglichkeiten bei der Konfiguration des Top-Modells, der Oracle Database Appliance X7-2HA. Es besteht die Wahlmöglichkeit, entweder ein auf SSD-Technologie basierendes High-Performance-Expansion-Shelf oder ein auf herkömmlichen 10-TB-Festplatten basierendes High-Capacity-Expansion-Shelf zu setzen.

Komponente	ODA X7-2S	ODA X7-2 M	ODA X7-2HA
Datenbank	SE/EE	SE/EE	SE/EE
Deployment Option	Bare Metal oder KVM-Virtualisierung	Bare Metal oder KVM-Virtualisierung	Bare Metal oder KVM-/OVM-Virtualisierung
Datenbank Deployment	Single-Instance	Single-Instance	Single-Instance, RAC, RAC One Node
CPU	10 Core Intel Xeon Silver 4114 Processor	2 x 18 Cores Intel Xeon Gold 6140 Processors	2 x 18 Cores Intel Xeon Gold 6140 Processors pro Server
Memory	192 GB (max. 384 GB)	384 GB (max. 768 GB)	384 GB (max. 768 GB)
Flash Storage (raw)	12,8 TB NVMe	12,8 TB NVMe (max. 51,2 TB)	3,2 TB SSD – REDO (raw) 16TB SSD – FLASH/DATA (raw)
Boot Disk	480 GB M.2 SATA SSD	480 GB M.2 SATA SSD	480 GB M.2 SATA SSD
Netzwerk	2 x 10 GbE Ports (RJ45) oder 2 x 10/25 GbE Ports (SFP28)	2 x 10 GbE Ports (RJ45) oder 2 x 10/25 GbE Ports (SFP28)	2 x 10 GbE Ports (RJ45) oder 2 x 10/25 GbE Ports (SFP28)
Expansion	Memory ODA 192 GB (6 x 32 GB) 19,2 TB Storage (3 x 6,4 TB NVMe)	Memory ODA 192 GB (6 x 32 GB) SSD 3,2 TB – 5 Pack (16 TB) HDD 10 TB – 15 Pack (150 TB)	Memory ODA 192 GB (6 x 32 GB) Expansion Shelf High Performance 4 x 800 GB + 20 x 3,2TB Expansion Shelf High Capacity 4 x 800 GB + 5 x 3,2 TB + 15 x10 TB

Tabelle 1

In der Oracle Database Appliance X7-2HA wird für den Interconnect der beiden Server kein InfiniBand mehr genutzt. Es wurde innerhalb der ODA nur intern verwendet und war daher für viele Kunden eine unbekannte Technologie. Stattdessen wird nun auf eine 25-GbE-Ethernet-Verbindung gesetzt, womit die meisten Kunden bereits Erfahrungen haben dürften.

Software für die Oracle Database Appliance X7-2

Die Einrichtung und Verwaltung der Oracle Database Appliance erfolgte bislang ausschließlich über den Oracle Appliance Manager, der ein einfach zu bedienendes grafisches User-Interface zur Verfügung stellte. Zusätzlich stand dem Anwender mit dem „oakcli" ein Command Line Tool für die Administration zur Verfügung. Appliance Manager und „oakcli" werden in der ODA-Generation X7-2 nur noch zum Setup und zur Administration der Oracle Database Appliance X7-2HA im virtualisierten Modus verwendet. Wird die ODA als Bare-Metal-Maschine aufgesetzt, geschieht das über eine Web-Konsole und das Command Line Tool „odacli". Mit dem neuen Toolset hat der Anwender eine Umgebung zur Verfügung, die sowohl im „Look&Feel" als auch in der Funktionalität dem Toolset entspricht, das in der Oracle Cloud verwendet wird. Im Laufe der nächsten Releases wird das „odacli" schrittweise erweitert und so das „oakcli" langfristig ersetzen (siehe „https://blogs.oracle.com/oda/the-two-software-stacks-of-oracle-database-appliance").

Eine wesentliche Neuerung in Bezug auf den Einsatz von Virtualisierung auf der ODA X7-2 besteht darin, dass es nun möglich ist, eine KVM-basierte Virtualisierung zu nutzen, um Applikationen zu betreiben. Als Gast-Operation-System wird momentan nur Linux unterstützt, in diesem dürfen zwar Anwendungen, aber keine Oracle-Datenbanken laufen. Zur Verwaltung der virtuellen Maschinen unter KVM müssen KVM-native Kommandos oder Tools verwendet werden; der Appliance Manager kann zurzeit nur in Zusammenhang mit Oracle VM genutzt werden (Details im ODA-Blog unter „https://blogs.oracle.com/oda/kvm").

Fazit

Der Artikel fokussiert sich auf die Änderungen der Hardware der Engineered-Systems-Generation X7 und deren Auswirkungen auf die Konfigurations-Alternativen. Oracle Engineered Systems bestehen jedoch aus der hier vorgestellten Hardware und einem abgestimmten Software-Stack, der einen wesentlichen Teil des Leistungsvermögens dieser Systeme ausmacht. So steht das aktuelle Datenbank-Release 18.2 mittlerweile sowohl auf der Exadata als auch auf der Database Appliance zum Einsatz bereit.

Die Engineered-Systems-Generation X7-2 steht zwischenzeitlich auch in Form von Cloud Services in der Oracle Public Cloud zur Verfügung; das Tooling der On-Premises-Systeme und der angebotenen Cloud Services läuft immer mehr zusammen, sodass der Kunde nicht mehr unterscheiden muss, wo welche seiner Systeme laufen – „Look&Feel" und Funktionalität werden sich immer weiter annähern. Die Durchgängigkeit der On-Premises- und der Cloud-Umgebungen wird weiter vorangetrieben – die einfache Migration einer Datenbank aus der Oracle Database Appliance im kundeneigenen Rechenzentrum in einen Oracle Database Cloud Service durch ein Unplug der PDB von der Database Appliance und Plug in eine Container-Datenbank in der Oracle Cloud ist bereits heute möglich. Damit rücken On-Premises- und Cloud-Welt noch näher zusammen.

Weiterführende Informationen

- *https://www.oracle.com/engineered-systems/exadata/index.html*
- *https://www.oracle.com/engineered-systems/database-appliance/index.html*
- *https://blogs.oracle.com/oda/oracle-database-18c-oda*
- *https://blogs.oracle.com/oda/kvm*

Frank Schneede
frank.schneede@oracle.com

DOAG Legal Council: Erste Einschätzung zu kostenpflichtigen Java SE-Updates

Die öffentlichen Updates für Java SE 8 sollen nach Januar 2019 nur noch kostenpflichtig zur Verfügung stehen. Dann benötigen Unternehmen eine kommerzielle Lizenz. Das DOAG Legal Council nimmt sich aktuell des Themas an und prüft rechtliche Möglichkeiten, Oracle in dieser Angelegenheit zu begegnen.

!Die Kunden sollten nun schnell auf diese Ankündigung reagieren und verschiedene Fragestellungen im Unternehmen klären, insbesondere wie viele Java-Installationen im Einsatz sind, wo sich diese befinden und ob es sich um eine eingebettete Nutzung handelt", so Dr. Jana Jentzsch, Mitglied des DOAG Legal Council. „Vor dem Hintergrund der möglicherweise hohen Kosten, die ab Januar anfallen könnten, werden einige Unternehmen gegebenenfalls sogar erwägen müssen, ob Java durch eine andere Technologie ersetzt werden kann."

Das DOAG Legal Council nimmt sich aktuell des Themas an und prüft, ob eventuell rechtliche Möglichkeiten bestehen, dass die Updates auch weiterhin kostenfrei zur Verfügung stehen oder zumindest Kosten reduziert werden können. In diesem Zusammenhang stellen sich laut Jentzsch beispielsweise Fragen nach dem auf das „Oracle Binary Code License Agreement" (BCLA) anwendbare Recht und dem Einräumen angemessener Übergangszeiträume.

ODA-X7-Appliance from Hell

Johannes Kraus, HiSolutions AG

Im Juni 2016 erschienen die damals ganz neuen ODA-X6-Appliances auf dem Markt. Aufgrund ihrer technischen Features und des attraktiven Preises dauerte es nicht lange, bis sie zum echten Verkaufsschlager wurden. Ein weiteres Jahr später, im Oktober 2017, stellte Oracle die Nachfolgerfamilie ODA-X7-Appliances vor, die inzwischen wieder fast ein Jahr auf dem Markt sind. Zeit genug, um in Kombination mit der gesammelten Erfahrung, die in einem Kundenprojekt gewonnen werden konnte, ein ausgiebiges Review mit einer persönlichen Meinung abzugeben.

Zu Beginn des Reviews wird die soeben erwähnte Familie kurz vorgestellt. Die Tabellen 1 und 2 zeigen die Hauptmerkmale der neuen ODA-X7-Appliances. Was zunächst im Gegensatz zur X6-Familie *[6]* auffällt, ist, dass in der neuen X7-Familie kein Modell „L" mehr zur Verfügung steht. Dieses wurde im Zuge der Neugestaltung der Hardware-Konfiguration durch Oracle wegrationalisiert.

Während sich das neue Modell „S" kaum vom alten unterscheidet, ist es nun für die Modelle „M" und „HA" möglich, diese mit mehr RAM und Festplattenspeicher auszustatten. Die HA-Version kann sogar in zwei verschiedenen Optionen – High Performance oder High Capacity – konfiguriert werden. Dies waren dabei jedoch nicht die einzigen Änderungen, die durchgeführt wurden. So wurde zum Beispiel auch der liebgewonnene VGA-Port zum Anschluss eines Monitors, der für die Ersteinrichtung sehr angenehm war, wegoptimiert. Was sich Oracle dabei gedacht hat, bleibt unklar, da es wohl kaum an den immensen Kosten eines solchen Bauteils liegen kann. Stattdessen soll nun für die Ersteinrichtung die ILOM-Schnittstelle entweder an einen DHCP-Server angeschlossen oder aber ein spezielles Kabel verwendet werden. Dieses ermöglicht den Anschluss eines Laptops über den USB-Port an den Konsolen-Port (RJ45) der Appliance.

Last but not least ist es nun auch möglich, den neuen SFP28-Stecker für Glasfaser zu verwenden. Selbstverständlich können nach wie vor auch die alten SFP+-10GbE-Stecker verwendet werden. Doch Vorsicht! Wer nun denkt, dass ein einfacher Wechsel der SFP-Stecker möglich ist, der irrt. Dazu

Komponenten	ODA-X7-2S	ODA-X7-2M
Größe	Eine Höheneinheit	Eine Höheneinheit
Prozessor	Eine CPU mit 10 Kernen (Intel Xeon Silver 4114)	Zwei CPUs à 18 Kerne (Intel Xeon Gold 6140)
Speicher	Min. 192 GB Max. 384 GB	Min. 384 GB Max. 768 GB
Netzwerk	2 x 10GBase-T (RJ45) oder 2 x 10/25 GbE (SFP28)	2 x 10GBase-T (RJ45) oder 2 x 10/25 GbE (SFP28)
Boot-Festplatten	2 x 480 GB SSD (gespiegelt)	2 x 480 GB SSD (gespiegelt)
Storage (je nach Konfiguration kann eine Doppel- oder Dreifachspiegelung konfiguriert werden)	2 x 6,4 TB NVMe SSDs	Optionen: A: 2 x 6,4 TB NVMe SSDs B: 5 x 6,4 TB NVMe SSDs C: 8 x 6,4 TB NVMe SSDs
DB-Edition	• 11g SE1, SE • 11g R2 EE • 12c SE2 • 12c R1/R2 EE	• 11g SE1, SE • 11g R2 EE • 12c SE2 • 12c R1/R2 EE
Virtualisierung	Oracle Linux KVM	Oracle Linux KVM
Datenbank-Deployment	Single Instance	Single Instance

Tabelle 1: ODA-X7-2S/M-Übersicht

Komponenten	ODA-X7-2-HA
Größe	Sechs Höheneinheiten (Server/Storage)
Prozessor	Zwei CPUs à 18 Kerne (Intel Xeon Gold 6140)
Speicher	Min. 384 GB Max. 768 GB
Netzwerk	2 x 10GBase-T (RJ45) oder 2 x 10/25 GbE (SFP28)
Boot-Festplatten	2 x 480 GB SSD (gespiegelt)
Storage (je nach Konfiguration kann eine Doppel- oder Dreifachspiegelung konfiguriert werden)	Mehrere Optionen je HA-Variante: High Performance: Min. 5 x 3,2 TB SSD Max. 40 x 3,2 TB SSD High Capacity: Min. 15 x 10 TB HDD Max. 30 x 10 TB HDD
DB-Edition	• 11g SE1, SE • 11g R2 EE • 12c SE2 • 12c R1/R2 EE
Virtualisierung	Oracle VM
Datenbank-Deployment	Single Instance, RAC, RAC One Node

Tabelle 2: ODA-X7-2-HA-Übersicht

sind weitere Arbeiten notwendig, die allerdings im Rahmen dieses Artikels nicht näher erläutert werden können.

Kundenprojekt

Bevor es mit der Installation und der Konfiguration der Appliances weitergeht, wird an dieser Stelle ein kurzer Umriss über das Kundenprojekt gegeben, das Auslöser und Grund für diesen Artikel war. Da die vorhandenen Server in die Jahre gekommen waren, die Performance nicht mehr den Ansprüchen entsprach und noch eine 11g-Standard-Edition im Einsatz war, entschied sich der Kunde zum Kauf von zwei ODA-X7-2S-Appliances, die mithilfe einer Dbvisit-Standby-Installation und -Konfiguration zu einer Datenbank-Hochverfügbarkeitslösung in Betrieb genommen wurden. Im Verlauf der Inbetriebnahme der Appliances kam es anschließend zu interessanten Ereignissen, auf die im weiteren Verlauf des Artikels eingegangen wird.

Installation und Konfiguration

Als die Appliances beim Kunden eintrafen, wurden sie voller Vorfreude ausgepackt und genau inspiziert. Wie zuvor schon bei den X6-Appliances gestaltete sich auch der Einbau bei der ODA-X7-Familie in die Serverschränke recht einfach. Zum Standard-Lieferumfang gehören neben der Appliance zwei Netzkabel sowie die Serverschienen zum Einbau in den Serverschrank. Die Netzwerkkabel und die „SFP"-Adapterstecker sind nicht im Lieferumfang enthalten.

Aufgrund des fehlenden VGA-Ports und der damit verbundenen Abstinenz eines Monitors wurde für die initiale Einrichtung eine DHCP-Adresse für die ILOM-Schnittstelle verwendet. Sollte eine Firma kein DHCP im Einsatz haben, könnte es an dieser Stelle bereits zu ersten Unannehmlichkeiten kommen. Glücklicherweise war dies hier nicht der Fall und es konnte somit problemlos ein Konsolenfenster über die ILOM-Schnittstelle auf der Appliance initiiert werden.

```
# configure-firstnet
Select the Interface to configure the network on (btbond1): btbond1
Configure DHCP on btbond1 (yes/no): no
INFO: You have chosen Static configuration
Enter the IP address to configure: xxx.xxx.xxx.xxx
Enter the Netmask address to configure: xxx.xxx.xxx.xxx
Enter the Gateway address to configure: xxx.xxx.xxx.xxx
```

Listing 1

Wie erwartet begannen die Appliances nach dem Einstecken der Netzkabel mit einer Selbstprüfung und konnten erfolgreich gestartet werden. Unverändert sind auch die Anmelde-Informationen zur initialen Netzwerk-Konfiguration, die mit dem Kommando „configure-firstnet" durchgeführt wird. Sobald der Befehl abgesetzt wurde, können bereits die ersten Unterschiede zur Vorgängerfamilie festgestellt werden. Die Auswahl der Netzwerk-Konfiguration ist beschränkt, sodass nicht mehr zwischen „btbond1", „btbond2" oder „sfpbond1" unterschieden werden kann. Aus Sicherheitsgründen wurden die eingegebenen Daten anonymisiert *(siehe Listing 1)*.

Sobald die initiale Netzwerk-Konfiguration abgeschlossen ist, können die benötigten Patches, einsehbar und verfügbar unter Metalink-Dokumenten-ID 888888.1 *[8]*, auf die Appliance kopiert und mithilfe der entsprechenden Befehle im Repository angemeldet werden. Im Anschluss daran lässt sich die Appliance entweder über eine Weboberfläche oder über die Kommandozeile in Betrieb nehmen. Die in diesem Fall gewählte Vorgehensweise ist die erstgenannte Möglichkeit und unter „https://<IP-Adresse>:7093/mgmt/index.html" verfügbar. Nach der erfolgreichen Anmeldung ist ein weiterer Unterschied feststellbar. Sofern bereits die Version 12.2.1.2.0 vorinstalliert ist, muss der Benutzer nach der ersten Anmeldung sein Passwort ändern. Das Passwort muss dabei bestimmte Vorgaben erfüllen, sodass beispielsweise „welcome1" nicht mehr möglich ist *(siehe Abbildung 1)*.

Diese Vorgaben überraschten positiv, da in der Vergangenheit jedes beliebige Passwort verwendet werden konnte. In Bezug auf die Web-Oberfläche bewährt es sich nach wie vor, Google Chrome zu verwenden, da es bei anderen Browsern zu Anzeige- oder Funktionalitätsfehlern kommen kann. Die nächste Neuerung ist nach der Änderung des Passworts ersichtlich. Über den neuen Menüpunkt „Patch Ma-

Abbildung 1: Passwortwechsel

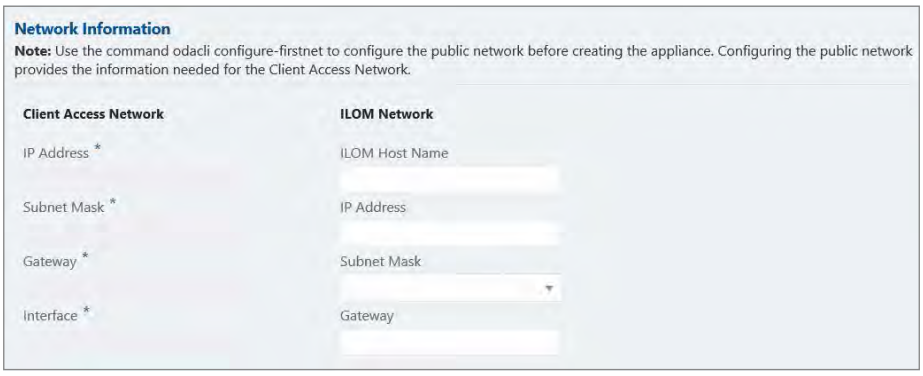


Abbildung 2: Netzwerk-Konfiguration

nager“ lassen sich nun auch ODA-Patches über die Web-Oberfläche installieren. Wie stabil und intuitiv diese neue Funktionalität ist, kann zu diesem Zeitpunkt nicht bewertet werden, da keine Tests erfolgten.

Eine weitere Neuerung zeigt sich bei der Eingabe der Appliance-Daten. Das Feld „Domain Name“ ist nun ein verpflichtendes Feld. Dies ist auf den ersten Blick kein negativer Punkt. Es sollte jedoch beachtet werden, dass dieses Feld auch dafür sorgt, dass der PFILE-Parameter „DB_DOMAIN“ gesetzt wird.

Schön zu sehen ist nun auch, dass die Werte aus dem Kommando „configure-firstnet“ zum Setzen des Public-Network automatisch in die entsprechenden Felder übernommen werden und nicht mehr manuell ein zweites Mal eingegeben werden müssen. Somit beschränkt sich die weitere Netzwerk-Konfiguration erstmals nur auf die ILOM-Schnittstelle *(siehe Abbildung 2)*.

Die weitere Konfiguration ist wie gehabt und einfach durchzuführen. Nachdem alle Daten eingegeben und geprüft sind, lässt sich die Appliance mit einem Knopfdruck aufbauen. An dieser Stelle ist noch zu erwähnen, dass die Konfiguration der Appliance derart durchgeführt wurde, dass mit der Erstellung keine Datenbank angelegt werden sollte.

Der Teufel steckt im Detail

Nachdem der Button „Submit“ betätigt wurde, wurde wie gewohnt ein Job gestartet, der die Appliance anhand der vorgenommenen Konfiguration aufbauen sollte. Kurz darauf trat allerdings der Fehler „Unable to create Group DBA“ auf. Das ist erstaunlich, da das Anlegen einer Linux-Gruppe im Grunde keine schwierige Aufgabe sein sollte. Nichtsdestotrotz war es nun an dieser Stelle erforderlich, ein Re-Image durchzuführen. Das zum damaligen Zeitpunkt aktuelle ISO-Image war die Version 12.2.1.3.0_180504.

Selbstverständlich sorgte dieser Fehler nicht gerade für Begeisterung. So ergab sich jedoch die gute Gelegenheit, dem Kunden live zu zeigen, wie die Appliance zurückgesetzt werden kann. Die Prozedur dafür ist schnell und einfach, sodass nach etwa 45 Minuten wieder eine saubere Appliance verfügbar war. Das Spiel konnte somit von vorne beginnen.

```
DCS-10001:Internal error encountered: Fail to run command Failed to
start the filesystem on commonstore.
Provisioning service creation June 14, 2018 2:39:26 PM CEST June 14,
2018 3:05:54 PM CEST Failure
Provisioning service creation June 14, 2018 2:39:26 PM CEST June 14,
2018 3:05:54 PM CEST Failure
OS usergroup 'asmdba'creation June 14, 2018 2:39:27 PM CEST June 14,
2018 2:39:27 PM CEST Success
…
ACFS File system 'DATA'creation June 14, 2018 3:05:47 PM CEST June 14,
2018 3:05:54 PM CEST Failure
Starting FileSystem June 14, 2018 3:05:50 PM CEST June 14, 2018 3:05:54
PM CEST Failure
```

Listing 2

```
DCS-10001:Internal error encountered: Fail to run root scripts
…
Grid stack creation June 14, 2018 4:21:54 PM CEST June 14, 2018 4:25:52
PM CEST Failure
Configuring GI June 14, 2018 4:21:54 PM CEST June 14, 2018 4:22:03 PM
CEST Success
Running GI root scripts June 14, 2018 4:22:03 PM CEST June 14, 2018
4:25:52 PM CEST Failure
```

Listing 3

Es dauerte nicht lange, bis das initiale Netzwerk konfiguriert, die Patches im Repository angemeldet und die benötigten Felder für die Appliance-Erstellung befüllt waren. Somit konnte Versuch Nummer zwei gestartet werden. Zunächst sah der Aufbau sehr vielversprechend aus, bis bei ca. 80 Prozent ein Fehler auftrat *(siehe Listing 2)*.

Was genau war passiert? Eine Analyse der Logfiles ergab keine nennenswerten Erkenntnisse, sodass der Beschluss gefasst wurde, vorsorglich einen Oracle-Service-Request zu erstellen. Da es durchaus einige Zeit dauern kann, bis eine hilfreiche Antwort von Oracle erwartet werden kann, entschied man sich, die Appliance erneut zurückzusetzen und einen dritten Versuch zu wagen. Also wieder das Prozedere von vorne: Re-Image, configure-firstnet, Repository-Update, Appliance-Felder befüllen, Submit und warten. Wie bereits vermutet, kam es bei ca. 80% erneut zu einem Fehler. Diesmal war es allerdings ein anderer als in den beiden vorangegangenen Versuchen *(siehe Listing 3)*.

Langsam wurde es unangenehm, vor allem auch deswegen, da die Auswertung der entsprechenden Logfiles keine wirkliche Erkenntnis brachte. Selbstverständlich wurden alle Fehler in dem entsprechenden Service Request notiert. Nach einer kurzen Absprache mit dem Kunden kam es zu Versuch Nummer vier.

Leider war auch dieser wieder erfolglos; dieses Mal auch wieder mit der gleichen Fehlermeldung. Es schien also, dass auf diese Art und Weise kein Vorankommen zu erwarten war. Doch was konnte nun getan werden? Bislang gab es auch keine Antwort von Oracle trotz eines Severity One Service Request.

Nachdem der Arbeitstag bereits weit fortgeschritten war, wurden alle weiteren Arbeiten auf den nächsten Tag verschoben. Auch in der Hoffnung, dass sich Oracle bis dahin meldet und mit einem guten Tipp weiterhelfen kann – neuer Tag, neues Glück. In der Tat, Oracle hatte sich gemeldet. Allerdings nicht so, wie erwartet. Es wurden verschiedene Einstellungen und Logfiles angefragt. Selbstverständlich wurde die Antwort nicht gerade positiv aufgenommen, da Oracle für diese Anfrage weit mehr als vier Stunden benötigt hatte. Diese Nachfrage hatte jedoch auch etwas Positives. So konnte weiteren Logfiles entnommen werden, dass es während der Appliance-Erstellung zu einer Null-Pointer-Exception gekommen war. Die Vermutung lag also nahe, dass es ein Problem mit einem der Eingabe-Parameter gab. Die Frage war nur: Mit welchem und war dies ein generelles Problem mit den X7-Appliances oder aber ein versionsabhängiger Fehler?

Um zumindest eine der Variablen streichen zu können, wurden das ISO-Image der vorherigen Version (12.2.1.2.0) heruntergeladen und ein erneutes Factory-Reset durchgeführt. Das Prozedere war mittlerweile bekannt, sodass sich auch der Kunde sicher im Umfeld des Re-Imaging- und Appliance-Erstellungs-Prozesses bewegen konnte. In der Zwischenzeit ereignete es sich sogar, dass der Oracle Support geantwortet hatte – aus Datenschutzgründen wird lediglich ein Auszug der Nachricht gezeigt *(siehe Listing 4)*.

```
[…
Hi,
Please verify all the network handshake is working among the Storage
servers to the ODA node?
…]
```

Listing 4

Die Antwort war etwas ernüchternd! Nach allen Informationen, die zur Verfügung gestellt wurden, und der extrem langen Wartezeit für einen Service Request mit Severity One wird nun gefragt, ob die Netzwerk-Anbindung mit dem Storage funktioniert, die übrigens bei der ODA-X7-2S in der Appliance verbaut ist. An dieser Stelle muss ganz klar gesagt werden, dass der Oracle Support in diesem Fall mehr als nur zu wünschen übrigließ. Selbstverständlich wurde der entsprechende Support-Mitarbeiter darauf hingewiesen, dass es keine Netzwerk-Verbindung zwischen Server und Storage geben kann.

In der Zwischenzeit wurde das Re-Image mit der alten Version abgeschlossen, sodass nun Versuch Nummer fünf gestartet werden konnte. Um zusätzlich eine weitere Änderung bewirken zu können, wurde entschieden, mit dem Appliance-Deployment eine Datenbank anlegen zu lassen. Nachdem der „Submit"-Button gedrückt war, stieg die Spannung proportional mit dem Fortschritt der Erstellung der Appliance und siehe da: Der Job wurde erfolgreich beendet.

Nach einem kurzen Test, in dem geprüft wurde, ob auch wirklich alles funktionierte und wie gewünscht erstellt wurde, begann das Update auf die damals neueste Version 12.2.1.3.0. Die Datenbank wurde vorsorglich wieder gelöscht, da diese auch zum derzeitigen Zeitpunkt nicht benötigt wurde. Auch hier gab es keine Vorkommnisse mehr, sodass das Update erfolgreich beendet werden konnte. So vergingen also 1,5 Tage, nur um eine Appliance in Betrieb nehmen zu können. An dieser Stelle sei kurz erwähnt, dass der Oracle Support bis heute noch keine Lösung zu dem Problem liefern konnte.

Mit neuem Enthusiasmus und viel Elan wurde jetzt versucht, die zweite ODA-X7-2S in Betrieb zu nehmen. Da die Patches der Version 12.2.1.3.0 bereits im Repository registriert wurden, hat man entschieden, einen Versuch mit der aktuellen Version zu wagen. Darüber hinaus wurde beschlossen, keine Datenbank mit der Appliance-Erstellung anzulegen, da es sich bei dieser Appliance um die Dbvisit-Standby-Appliance handelte und die benötigte Datenbank von Dbvisit selbst erstellt werden sollte – ein großer Fehler, wie sich im Nachhinein herausstellen sollte.

Zunächst konnte das Glück kaum gefasst werden, als die Appliance im ersten Versuch mit der neuen Version erfolgreich erstellt werden konnte. Die Ernüchterung kam allerdings, als die Details der Maschine eingesehen wurden. An der Stelle, an der stehen sollte, dass es sich um eine Standard-Edition-Appliance handeln sollte, stand, dass es eine Enterprise Edition war. Seltsamerweise ließ sich aber keine Standard-Edition-Datenbank über die Web-Oberfläche erstellen.

Eine kurze Überprüfung der ersten Appliance ergab, dass hier die richtige Edition angeben wurde. Es sieht also danach aus, dass die Edition der Appliance mit der Auswahl der Datenbank-Erstellung bei der Inbetriebnahme der Appliance zusammenhängt. Da es jedoch möglich war, eine SE-Datenbank zu erstellen, ignorierte man diese Anzeige und eröffnete erneut ein Ticket bei Oracle, um diesen Sachverhalt zu klären.

Währenddessen wurde mit der Installation und Konfiguration von Dbvisit begonnen. Wie zu erwarten gab es hier keinerlei Hürden zu überwinden. Im letzten Schritt sollte die Standby-Datenbank erstellt werden. In gewohnter Zuverlässigkeit und Schnelligkeit wurde die Standby-Datenbank eingerichtet, in der Appliance registriert und das Recovery gestartet.

Frohen Mutes wurden einige Prüfungen auf der Datenbank durchgeführt. Zum Entsetzten der Beteiligten wurde dabei festgestellt, dass die Standby-Datenbank nun eine Enterprise-Edition-Datenbank war. Des Rätsels Lösung war, dass Dbvisit anhand der Typen-Angabe der Appliance entscheidet, eine Standard Edition oder Enterprise Edition zu erstellen.

Was nun folgte, war zu erwarten. Die Appliance musste auf Werkseinstellungen zurückgesetzt und neu aufgesetzt werden; dieses Mal jedoch mit einer initialen Datenbank der SE-Edition. So wurde nun auch der Typ der Appliance richtig angezeigt und Dbvisit verwendete die richtige Edition. Was für ein Marathon für eine eigentlich simple und mit wenig Aufwand verbundene Tätigkeit. Auch eine Lösung für den zweiten Oracle-Service-Request wurde bis heute noch nicht zur Verfügung gestellt.

Fazit

Als die ODA-X6-Appliances erschienen, steckten sie in den Kinderschuhen und mussten erst das Laufen beziehungsweise Rennen lernen. Viele Punkte waren versteckt oder umständlich zu konfigurieren. Beim Patchen kam es aufgrund der verschiedensten Fehler zu Abbrüchen, die allerdings bisher allesamt mit einem positiven Ergebnis gelöst werden konnte. Das fehlerhafte Verhalten änderte sich jedoch mit jedem Patch mehr und mehr zum Positiven, bis es, bis kurz vor der Einführung der ODA-X7-Appliances, eine recht stabile Software/Appliance war. Die Kinderkrankheiten waren beseitigt, neue Features etabliert und es machte Spaß, mit den Maschinen zu arbeiten.

Die Performance suchte seit jeher ihresgleichen, sodass es an dieser Stelle nie etwas auszusetzen gab. Waren die Appliances einmal eingerichtet, liefen sie sehr stabil, sicher und vor allem schnell. Hinzu kam der wirklich unschlagbar günstige Preis. So könnte man meinen, dass mit den neuen Appliances aus den Erfahrungen der Vergangenheit gelernt wurde und mit der neuen Hardware auch eine überarbeitete Softwareversion auf den Markt gebracht werden würde. Doch leider kann diese Meinung zu diesem Zeitpunkt nicht

bestätigt werden. Was Oracle hier auf den Markt brachte oder mit der genannten Version veröffentlichte, ist, positiv ausgedrückt, suboptimal. Es scheint, als würde der Endbenutzer zum Beta-Tester werden. Der Support ist im Falle der aufgetretenen Fehler wenig hilfreich – hier besteht dringender Nachbesserungsbedarf.

Obwohl am Ende beide Appliances erfolgreich in Betrieb genommen werden konnten, war der Weg alles andere als selbsterklärend und einfach. Doch gerade diese beiden Punkte sind, neben weiteren Punkten, Argumente, die für eine Appliance sprechen sollten.

In Bezug auf die Stabilität lässt sich zum aktuellen Zeitpunkt keine negative Kritik anbringen: Sie ist wie gewohnt zufriedenstellend. Bleibt zu hoffen, dass sich die Qualität zum Positiven verbessert, sodass auch eine initiale Einrichtung in alter Frische erfolgreich über die Bühne gebracht werden kann.

Das Konzept der Appliances hat Potenzial und die Leistung ist fantastisch. Auch der Kunde war am Ende trotz der Fehler und Ereignisse relativ zufrieden und wird auch bis auf Weiteres an den Appliances festhalten. Die Maschinen sind nach wie vor, sobald sie erfolgreich eingerichtet werden konnten, stabil, schnell und ermöglichen es, mit wenigen Schritten neue Datenbanken zu erstellen, zu löschen oder zu patchen.

Links

[1] ODA-X7-2S/M White Paper: *https://www.oracle.com/technetwork/database/database-appliance/learnmore/odax7-2sm-ds-3933491.pdf*

[2] ODA-X7-2-HS White Paper: *https://www.oracle.com/technetwork/database/database-appliance/learnmore/odax7-2-ha-ds-3933489.pdf*

[3] Database Appliance X7-2 Deployment and User's Guide: *https://docs.oracle.com/cd/E93461_01/doc.122/e92398/toc.htm*

[4] Setup Booklet X7-2S/M: *https://docs.oracle.com/cd/E93461_01/doc.122/e88372.pdf*

[5] Setup Booklet X7-2-HA: *https://docs.oracle.com/cd/E93461_01/doc.122/e88371.pdf*

[6] Oracle Database Appliance X6-2 Model Family: *http://www.oracle.com/technetwork/database/database-appliance/documentation/oracle-db-appliance-whitepaper-1867695.pdf*

[7] Dbvisit Standby 8.0 User Guide: *https://dbvisit.atlassian.net/wiki/spaces/DS8QSG/overview*

[8] Doc ID 888888.1: *https://support.oracle.com/epmos/faces/DocContentDisplay?_afrLoop=195159138307909&id=888888.1*

Johannes Kraus
kraus@hisolutions.com

US-Anwaltskanzlei kündigt Klage gegen Oracle Corporation und einzelne Führungskräfte an

Die US-Anwaltskanzlei Bernstein Litowitz Berger & Grossmann LLP („BLB&G") gab am 10. August 2018 bekannt, für ihren Auftraggeber City of Sunrise Firefighters' Pension Fund und im Namen von Investoren eine Sammelklage gegen die Oracle Corporation und einige ihrer Führungskräfte eingereicht zu haben *(siehe „https://www.prnewswire.com/news-releases/bernstein-litowitz-berger-grossmann-llp-announces-securities-class-action-suit-filed-against-oracle-corporation-and-certain-of-its-executives-300695598.html")*. Die in Florida ansässige Investmentgruppe verwaltet rund 143 Millionen Dollar an Vermögen.

Ihre Mandanten hatten Aktien von Oracle zwischen dem 10. Mai 2017 und dem 19. März 2018 gekauft. Diese behaupten, dass die Beklagten während der Sammelklageperiode gegen die Bestimmungen des sogenannten „Exchange Act" verstoßen hätten, indem von der Oracle Corporation falsche und irreführende Pressemitteilungen, Einreichungen bei der „U. S. Securities and Exchange Commission" (SEC) und Erklärungen während Telefonkonferenzen mit Investoren und Analysten herausgegeben hätten.

Entgegen den Darstellungen von Oracle sei der Verkauf von Cloud-Produkten in den letzten Jahren nicht durch eine normale Geschäftsentwicklung, sondern mithilfe von Drohungen und Erpressungstaktiken vorangetrieben worden, begründet die Kanzlei die Klage. Unter anderem habe das Unternehmen Bestandskunden mit Audits der Nutzung von Nicht-Cloud-Software-Lizenzen des Unternehmens gedroht, sofern sie sich nicht dazu bereit erklärten, ihr Geschäft auf Oracle-Cloud-Programme umzustellen.

Oracle hatte am 19. März 2018 bekannt gegeben, dass das Umsatzwachstum im Cloud-Geschäft stagniere und im Vergleich zu Wettbewerbern deutlich geringer ausfiele. Nach diesen Enthüllungen brachten Analysten und Marktbeobachter die schlechte finanzielle Performance von Oracle mit seiner unangemessenen Verkaufstaktik in Verbindung. Infolge dieser Angaben ging der Kurs der Aktie der Gesellschaft deutlich zurück.

Die Anwaltskanzlei BLB&G verfügt über Büros in New York, Kalifornien, Louisiana und Illinois und ist spezialisiert auf Wertpapierbetrug, Corporate Governance, Aktionärsrechte, Arbeitsdiskriminierung und Zivilrechtsstreitigkeiten. Sie erhebt weltweit Sammelklagen und Privatklagen für institutionelle und private Kunden.

Vom Nutzen vertikaler Dienste

Jürgen Sieben, ConDeS GmbH & Co. KG

In der letzten Folge wurden Objekte als Schnittstelle verwendet, dieser Artikel lässt es ein wenig schlichter angehen und kümmert sich wieder um normale Programmierung.

Obwohl: Wen der Ansatz der Objektorientierung grundsätzlich interessiert und wer Anwendungsbeispiele sucht, sollte sich einmal die Neufassung der ursprünglich von Steven Feuerstein entwickelten utPLSQL-Suite in Version 3 ansehen. Das extrem elegante Test-Interface ist nur durch Objektorientierung möglich und stellt eine sehr inspirierende Anwendung dieses Konzepts dar. Alternativ kann man auch gern einen Blick auf das PL/SQL Instrumentation Toolkit *(PIT, siehe „http://github.com/j-sieben/PIT“)* aus der Feder des Autors werfen, das dieses Mittel einsetzt, um Ausgabe-Module ansprechen zu können, ohne zur Kompilierzeit wissen zu müssen, welche Ausgabe-Module vorhanden sind. Dieses Package ist im Übrigen ein Beispiel für einen vertikalen Dienst, womit wir dann beim Thema dieser Folge wären.

Unter einem „vertikalen Dienst“ wird in dieser Folge die Funktionalität verstanden, die Schichten-übergreifend (wie View-, Control- oder Model-Layer) vorhanden sein muss. Klassische Bestandteile dieser vertikalen Dienste sind Methoden zur Code-Instrumentierung, Fehlerbehandlung, Performance-Messung etc. Im Zusammenhang dieses Artikels soll der Begriff allerdings allgemeiner als Sammlung von Utilities verstanden werden, die in der Entwicklung von Software innerhalb der Datenbank an allen möglichen Stellen verwendet werden.

Jeder Entwickler sollte ein gut gepflegtes Set dieser Werkzeuge besitzen, um zum Beispiel wiederkehrende Anforderungen einfacher lösen zu können. PIT ist ein solches Werkzeug, das sich um das Logging, die Fehlerbehandlung, aber auch die Ausgabe von Meldungen an die Anwendungen und um Assertionen kümmert. Dazu ein Blick auf den letzten Punkt, die Assertionen, um den Wert solcher Hilfsmittel am praktischen Beispiel zu erfahren. In einem offen zugänglichen Produktiv-Code (programmiert von Oracle) steht das folgende, stark gekürzte Beispiel für eine verbesserungsbedürftige Programmierung *(siehe Listing 1)*.

Dass dies ein schlechtes Beispiel für Programmierung ist, erkennt man schon an der Schwierigkeit zu ermitteln, was dieser Code eigentlich tut. Offensichtlich werden Benutzernamen geprüft und in eine Liste mit guten oder schlechten Benutzern aufgeteilt. Verbesserungspotenzial findet sich zuhauf:

- „L_VALID“ dient nur dem Zweck, eine „CASE“-Anweisung nachzuprogrammieren. Das wäre mit einer einfachen „CASE“-Anweisung deutlich leichter gewesen
- Die dauernde Wiederholung gleicher Code-Blöcke legt die Auslagerung in Hilfsmethoden nahe
- Der Code beschäftigt sich fast ausschließlich mit den Fehlern, die eigentliche Aufgabe wird überhaupt nicht klar, sie versteckt sich gegen Ende in einer der Prüf-Routinen

Wenn wir dies verbessern möchten, sollten wir über eine Assertions-Methode nachdenken, also eine einfache Methode, die einen Sachverhalt prüft, nichts tut, wenn die Bedingung erfüllt ist, und anderenfalls einen Fehler wirft. Im Kern ist so eine Methode trivial *(siehe Listing 2)*. Refaktorisieren wir das Beispiel und lagern die Prüfung gegen die Tabellen in Hilfsmethoden („CHECK_USER_NOT_EXISTING“, „CHECK_DUPLICATE_USER“) aus, sieht das Ergebnis wie in *Listing 3* aus.

Zunächst werden die Assertionen durchgeführt. Schlägt eine Assertion fehl, wird zum „EXCEPTION“-Block verzweigt und der aktuelle Benutzer mit der passgenauen Meldung in die Liste der ungültigen Benutzer eingefügt, ansonsten in die Liste der gültigen. Man sieht, wie drastisch sich diese einfache Hilfsmethode auswirkt:

- Wir benötigen überhaupt keine „CASE“-Anweisung mehr, diese Aufgabe übernimmt die Exception der Assertions-Methode
- Wir überblicken die Aufgabe der Methode sofort: Prüfe, ob der Benutzer valide ist, und speichere ihn anschließend in die entsprechende Liste. Hauptgrund ist, dass die Hilfsmethode „ASSERT“ schon durch ihren Namen anzeigt, was hier passiert

Solche Hilfsmethoden sollten also stets zur Hand sein und nicht jedes Mal neu programmiert werden müssen. Hat man andererseits ein solches Hilfspackage, das auf allen Code-Ebenen verwendet werden kann, lohnt sich dafür ein wenig mehr Aufwand. Hier ist es eine einfache Assertions-Methode, aber vielleicht sind ja speziellere Methoden sinnvoll? Eine Assertions-Methode, die auf „NULL“ prüft etwa, oder eine, die mindestens eine Zeile für eine übergebene „SELECT“-Anweisung erwartet oder im Gegenteil gerade keine Zeile? Das ist der tiefere Sinn dieser Packages: Dinge werden durch spezielle Hilfsmethoden konsistent und einfach gelöst, der hierfür nötige Aufwand muss jedoch nur einmal erbracht werden.

```
declare
  <...>
begin
  for j in 1..l_emails.count loop
      if eba_stds_fw.get_preference_value('USERNAME_FORMAT') = 'EMAIL' then
          if <...> then
              apex_collection.add_member(
                  p_collection_name => 'EBA_STDS_BULK_USER_INVALID',
                  p_c001            => l_username,
                  p_c002            => apex_lang.message('MISSING_AT_SIGN'));
              l_valid := false;
          end if;

          if <...> and l_valid then
              apex_collection.add_member(
                  p_collection_name => 'EBA_STDS_BULK_USER_INVALID',
                  p_c001            => l_username,
                  p_c002            => apex_lang.message('MISSING_DOT'));
              l_valid := false;
          end if;
      end if;

      if <...> and l_valid then
          apex_collection.add_member(
              p_collection_name => 'EBA_STDS_BULK_USER_INVALID',
              p_c001            => upper(l_username),
              p_c002            => apex_lang.message('USERNAME_TOO_LONG'));
          commit;
          l_valid := false;
      end if;

      if l_valid then
          for c1 in (select username
              from eba_stds_users
              where upper(username) = upper(l_username))
      loop
              wwv_flow_collection.add_member(
                  p_collection_name => 'EBA_STDS_BULK_USER_INVALID',
                  p_c001            => upper(l_username),
                  p_c002            => apex_lang.message(,ALREADY_IN_ACL'));
              l_valid := false;
              exit;
          end loop;
      end if;

      if l_valid then
          for c1 in (select c001
                  from wwv_flow_collections
                  where collection_name = 'EBA_STDS_BULK_USER_VALID'
                  and c001 = upper(l_username))
          loop
              apex_collection.add_member(
                  p_collection_name => 'EBA_STDS_BULK_USER_INVALID',
                  p_c001            => upper(l_username),
                  p_c002            => apex_lang.message(,DUPLICATE_USER'));
              l_valid := false;
              exit;
          end loop;
      end if;

      if l_valid then
          apex_collection.add_member(
              p_collection_name => 'EBA_STDS_BULK_USER_VALID',
              p_c001            => upper(l_username));
              commit;
      end if;
  end loop;
end;
```

Listing 1: Schlechtes Programmier-Beispiel

Hilfspackages dieser Art sind zudem in allen Projekten wiederverwendbar und vereinheitlichen so die Art, wie programmiert wird. Wichtig bei der Auswahl des passenden Hilfspackage ist allerdings, dass diese sich nicht auf eine konkrete Anwendungsschicht beziehen, etwa auf Apex als View-Layer. Der Grund liegt darin, dass wir mit diesem Ansatz nun wieder gegen die Idee des vertikalen Dienstes verstoßen würden, denn es ist nicht ausgeschlossen, dass eine Methode auf einer tieferen Anwendungsschicht sich nun mit den Anforderungen von Apex konfrontiert sieht, was eigentlich nicht passieren soll.

Insofern ist die Implementierung unserer „ASSERT"-Methode noch generisch genug, allerdings sollten wir überlegen, ob es nicht günstig wäre, die Methode in einen Kontext der Verwaltung von Meldungstexten einzubinden. Der Beispiel-Code delegiert die Verwaltung der tatsächlichen Meldung an Apex („apex_lang.message(<Name der Meldung>)"), was im Kontext einer Apex-Anwendung in Ordnung ist, aber einen Verstoß gegen die Schicht-Unabhängigkeit darstellt: Code der Geschäftslogik müsste sich nun darauf verlassen, dass Apex und damit auch die Verwaltung der Meldungen vorhanden sind, um auf gleiche Weise zu funktionieren.

Da Assertions-Methoden jedoch immer Meldungstexte benötigen, könnte man fordern, dass die entsprechenden Texte einfach übergeben werden müssen. Aber was ist nun mit internationalisierter Software? Wie übergeben wir die Meldung in der passenden Sprache? Was ist mit Ersetzungen innerhalb der Meldungstexte? Schnell wird klar, dass der Aufwand nun doch wieder sehr steigt. Das ist grundsätzlich kein Problem, nur sollte der Aufwand nicht dort steigen, wo die Hilfsmethode verwendet wird, sondern maximal in der Hilfsmethode selbst, denn dann steigt der Aufwand nur einmalig.

PIT macht dies so, indem es eigene Meldungstexte verwaltet und diese – umgekehrt – auch in Apex ausgeben kann. Daher entfällt die Notwendigkeit, Meldungen außerhalb des Codes im Metadatenmodell von Apex vorzuhalten, zudem sind Meldungen nun auch, als Teil des vertikalen Dienstes, unabhängig von den Anwendungsschichten. In PIT sähe der Code von oben dann wie in *Listing 4* aus.

```
procedure assert(
   p_condition in boolean,
   p_message in varchar2)
as
begin
   if not p_condition then
       raise_application_error(-20000, p_message);
   end if;
end assert;
/
```

Listing 2: Keimzelle einer einfachen Assertions-Methode

```
declare
  <...>
begin
   for j in 1..l_emails.count loop
       l_is_email := eba_stds_fw.get_preference_value('USERNAME_FORMAT') = 'EMAIL';

       -- VALIDATION
       if l_email then
           assert(<...>, apex_lang.message('MISSING_AT_SIGN'));
           assert(<...>, apex_lang.message('MISSING_DOT'));
       end if;
       assert(<...>, apex_lang.message('USERNAME_TOO_LONG'));
       assert(check_user_not_existing, apex_lang.message('ALREADY_IN_ACL'));
       assert(check_duplicate_user, apex_lang.message('DUPLICATE_USER'));

       -- PROCESSING
       apex_collection.add_member(
           p_collection_name => 'EBA_STDS_BULK_USER_VALID',
           p_c001            => upper(l_username));
   end loop;

exception
   when others then
       apex_collection.add_member(
           p_collection_name => 'EBA_STDS_BULK_USER_INVALID',
           p_c001            => upper(l_username),
           p_c002            => sqlerrm);
end;
```

Listing 3: Refaktorisierte Version mit ausgelagerten Hilfsmethoden

```
declare
   <...>
begin
   for j in 1..l_emails.count loop
       l_is_email := eba_stds_fw.get_preference_value('USERNAME_FORMAT')
= 'EMAIL';
       if l_email then
           pit.assert(<...>, msg.MISSING_AT_SIGN);
           pit.assert(<...>, msg.MISSING_DOT);
       end if;
       pit.assert(<...>, msg.USERNAME_TOO_LONG);
       pit.assert(check_user_not_existing, msg.ALREADY_IN_ACL);
       pit.assert(check_duplicate_user, msg.DUPLICATE_USER);

       apex_collection.add_member(
           p_collection_name  => 'EBA_STDS_BULK_USER_VALID',
       p_c001                 => upper(l_username));

   end loop;
exception
   when others then
       apex_collection.add_member(
           p_collection_name  => 'EBA_STDS_BULK_USER_INVALID',
           p_c001             => upper(l_username),
           p_c002             => sqlerrm);
end;
```

Listing 4: Fassung mit PIT

So kann man beides haben: gute Code-Qualität und geringstmöglichen Programmieraufwand.

Jürgen Sieben
j.sieben@condes.de

Wir begrüßen unsere neuen Mitglieder

Persönliche Mitglieder

- Mirko Freudenberger
- Matthias Röbisch
- Daniele Massimi
- Bruno Zimmermann
- Peter Bauer
- Marjan Rak
- Danica Sterk
- Andreja Veber Horvat
- Dr. Igor Salamun
- Dennis Pohlmann
- Dejan Hanzekovic
- Ales Sameja
- Henning Carbel

Firmenmitglieder DOAG

- Deutsche Bausparkasse Badenia AG, Markus Keese
- anykey GmbH, Timm Aust
- Hermes Germany GmbH, Rainer Bahr

Termine

01.11.2018
Regionaltreffen Dresdne/Sachsen
Helmut Marten

09.11.2018
DOAG Datenbank Webinar
online

09.11.2018
AOUG APEX 18.1 New Features
Dipl.Ing. Roland Willhalm
Wien

15.11.2018
Regionaltreffen Nürnberg/Franken
Martin Klier & Thomas Köppel
Nürnberg

20.11.2018
DOAG 2018 Konferenz + Ausstellung
Nürnberg

26.11.2018
Regionaltreffen Thüringen
Jörg Hildebrandt
Erfurt

03.12.2018
Regionaltreffen München/Südbayern
Andreas Ströbel
München

04.12.2018
Regionaltreffen Hannover
Andreas Ellerhoff
Hannover

04.12.2018
Regionaltreffen Hamburg/Nord
Jan-Peter Timmermann & Benjamin Kurschies
Hamburg

05.12.2018
Regionaltreffen Berlin/Brandenburg
Michael Keemers & Mylène Diacquenod
Berlin/Brandenburg

07.12.2018
Vorstandssitzung
Berlin

10.12.2018
Regionaltreffen Osnabrück/Bielefeld/Münster
Andreas Kother & Klaus Günther
Osnabrück/Bielefeld/Münster

14.12.2018
Datenbank Webinar
online

20.12.2018
Regionaltreffen Nürnberg/Franken
Martin Klier & Thomas Köppel
Nürnberg

Impressum

Red Stack Magazin wird gemeinsam herausgegeben von den Oracle-Anwendergruppen DOAG Deutsche ORACLE-Anwendergruppe e.V. (Deutschland, Tempelhofer Weg 64, 12347 Berlin, www.doag.org), AOUG Austrian Oracle User Group (Österreich, Lassallestraße 7a, 1020 Wien, www.aoug.at) und SOUG Swiss Oracle User Group (Schweiz, Dornacherstraße 192, 4053 Basel, www.soug.ch).

Red Stack Magazin ist das User-Magazin rund um die Produkte der Oracle Corp., USA, im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Oracle-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

Red Stack Magazin wird verlegt von der DOAG Dienstleistungen GmbH,Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

Die DOAG Deutsche ORACLE-Anwendergruppe e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Stefan Kinnen. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
(Anschrift s.o.)
Chefredakteur (ViSdP): Wolfgang Taschner
Kontakt: redaktion@doag.org
Weitere Redakteure (in alphabetischer Reihenfolge): Lisa Damerow, Mylène Diacquenod, Marina Fischer, Klaus-Michael Hatzinger, Sanela Lukavica, Martin Meyer, Yann Neuhaus, Fried Saacke

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH
(Anschrift s.o.)

Fotonachweis:
Titel: © Jacek Kita/123RF
S. 5: ©: DOAG
S. 6 links: © Java User Group Stuttgart e.V.
S. 6 rechts: Alex_Muchnik/DOAG
S. 7: Schuchrat Kurbanov/DOAG
S. 8+9: © Wolfgang Taschner/DOAG
S. 12: © Dzianis Rakhuba
S. 16: © aimage/123RF
S. 24: © bimdeedee/123RF
S. 31: © Lukas Kurka/123RF
S. 36: © Brett Lamb/123RF
S. 42: © Nirut Saelim/123RF
S. 50: © Anastasiia Nevestenko/123RF
S. 56: © Pannawat Muangmoon/123RF
S. 66: © ORACLE
S. 71: © sararoom/123RF
S. 76: © alphaspirit/123RF
S. 81: © Dzianis Kuryanovich/123RF

Anzeigen:
Simone Fischer,
DOAG Dienstleistungen GmbH
(verantwortlich, Anschrift s.o.)
Kontakt: anzeigen@doag.org
Mediadaten und Preise unter:
www.doag.org/go/mediadaten

Druck:
adame Advertising and Media GmbH,
www.adame.de

Inserentenverzeichnis

20. - 23. November in Nürnberg

2018.doag.org

PROGRAMM ONLINE
mit rund 450 Vorträgen

Eventpartner: AOUG SOUG swiss oracle user group iJUG Verbund ORACLE

Trivadis triCast

Von Profis für Profis. Jetzt anmelden!

Direkt zur Anmeldung:
m.trivadis.com/tricast

Der neue Trivadis triCast ist Ihr aktueller Webcast mit Themen rund um die IT-Welt. Kompakt und kompetent – vom Überblick bis hin zu vertiefenden Informationen. Als einer der führenden IT-Dienstleister bieten wir Ihnen News, Hintergründe und Wissen aus erster Hand zu Oracle-, Microsoft- und Open-Source-Technologien. Die Besonderheit: Findet ein Thema besonderen Zuspruch, werden wir sehr kurzfristig weitere und vertiefende triCasts für Sie bereitstellen.
Es lohnt sich. Melden Sie sich direkt an: m.trivadis.com/tricast

BASEL ▪ BERN ▪ BRUGG ▪ DÜSSELDORF ▪ FRANKFURT A.M. ▪ FREIBURG I.BR. ▪ GENF
HAMBURG ▪ KOPENHAGEN ▪ LAUSANNE ▪ MÜNCHEN ▪ STUTTGART ▪ WIEN ▪ ZÜRICH

trivadis
makes IT easier.