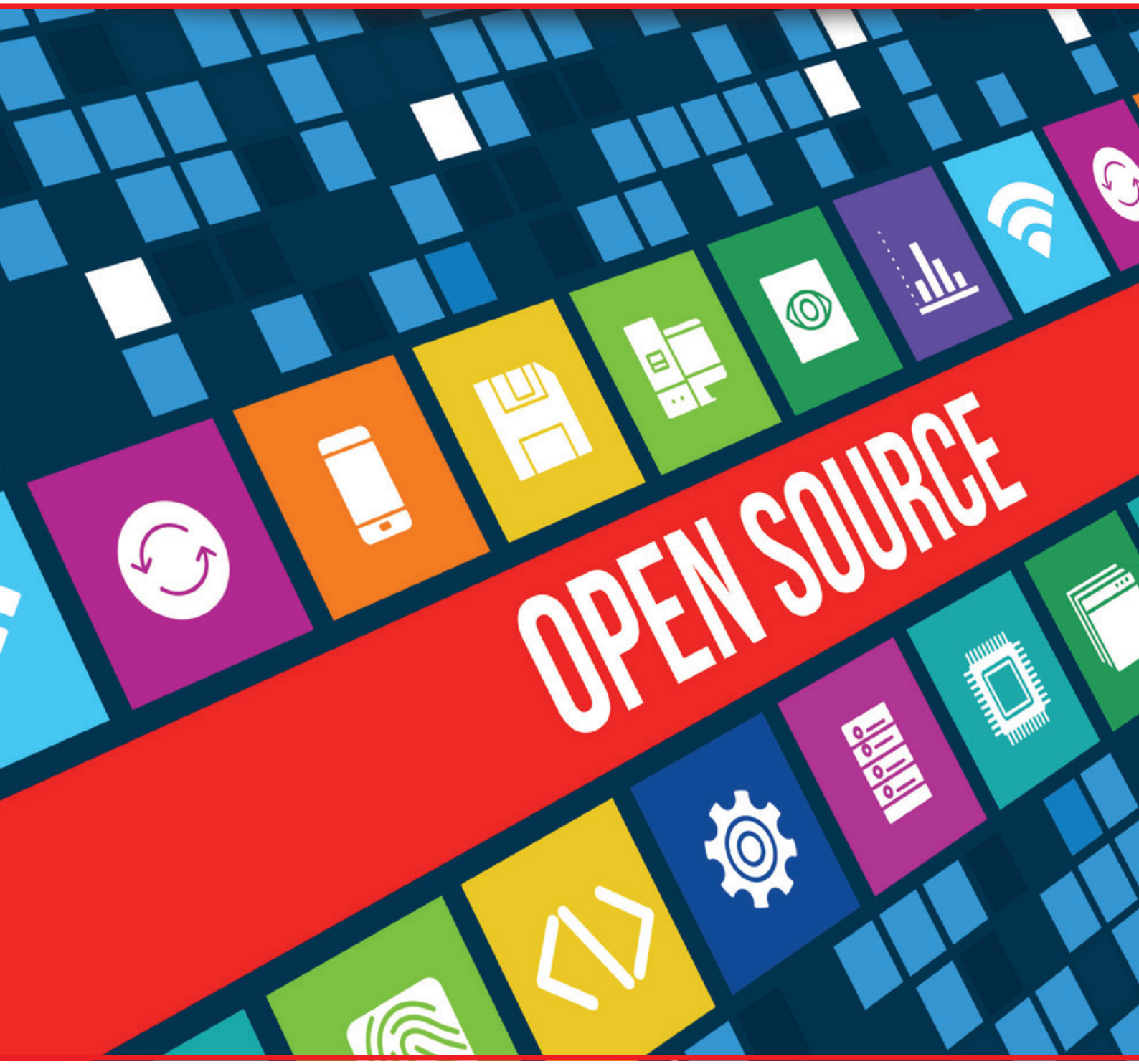
Nr. 6 | Januar 2019 | ISSN 2366-7915 | www.doag.org

Red Stack Magazin

DOAG SOUG swiss oracle user group AOUG AUSTRIAN ORACLE USER GROUP



Tuning

Wichtige Erfahrungen aus der Praxis

Im Interview

Fried Saacke, DOAG-Vorstand und Geschäftsführer

Oracle auf VMware

Tipps und Tricks aus dem VMware-Support

Christian Trieb
Leiter DOAG Datenbank Community

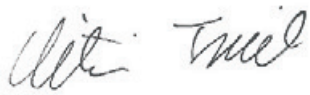
Liebe Mitglieder, liebe Leserinnen und Leser,

Open-Source-Datenbanken sind inzwischen eine wahrnehmbare Größe im Datenbank-Markt. Auch in der Vergangenheit haben bereits Open-Source-Produkte im Umfeld der Oracle-Datenbanken eine große Rolle gespielt, sei es im Monitoring und der Überwachung oder bei den Entwicklungswerkzeugen.

Vermehrt kleinere Anwendungen nutzen nun auch Open-Source-Datenbanken, um ihre Daten sinnvoll verarbeiten können. Sie sind in einigen Anwendungsbereichen eine sinnvolle Ergänzung zur Oracle-Datenbank. Dabei stellt sich nicht die Frage: „Oracle-Datenbank oder Open-Source-Datenbank?", sondern die, welche Open-Source-Datenbank am besten zur Oracle-Datenbank passt, um das gewünschte Ergebnis zu erreichen. Insofern ergänzen sich Oracle-Datenbank und Open-Source-Datenbanken sehr gut.

Ich wünsche Ihnen viel Spaß und neue Erkenntnisse beim Lesen dieser Ausgabe.

Ihr



Oracle bietet mehr Daten-Management-Systeme als nur die Oracle-Datenbank

PostgreSQL hat in den vergangenen Jahren enorm an Bedeutung gewonnen

Tipps für die Konfiguration einer VM zur optimalen Unterstützung der Oracle-Datenbank

Einleitung

Open-Source-Datenbanken

Tuning

Entwicklung

Datenbank

Intern

Timeline

24. August 2018

Nach dem letztjährigen Besuch der Konferenz der polnischen Oracle User Group (POUG) reist die DOAG Next Generation Community in diesem Jahr zum APEX Day 2018 in Stockholm (Schweden) und tauscht mit 70 Teilnehmern die aktuellsten News und Tipps rund um die beliebte Oracle-Datenbank-Technologie aus. Durch den kostenlosen Zugang zum Event und das Bereitstellen von zwölf Teilnehmerplätzen kommt die schwedische Oracle User Group der DOAG Next Generation Community herzlich entgegen und zeigt ihre Gastfreundschaft und Offenheit. Vier Apex-Spezialisten aus der Next Generation Community halten einen gemeinsamen Vortrag. Den Twitter-Sturm rund um den Event muss man da gar nicht weiter erwähnen.

Die DOAG Next Generation Community bei ihrem Vortrag

7. September 2018

Das JavaLand-Programmkomitee trifft sich in Berlin, um die Vorträge für die JavaLand 2019 zusammenzustellen. Keine leichte Aufgabe, aus den insgesamt 637 Einreichungen die 100 passendsten auszuwählen. Das sind ein Viertel mehr Einreichungen als im vergangenen Jahr. Auch das Newcomer-Programm für Referenten ohne Bühnenerfahrung fand reges Interesse mit 60 Einreichungen (zum Vergleich: 23 in 2018). Beim Schulungstag haben die Stream-Leiter dieses Mal die Qual der Wahl zwischen 37 Einreichungen von 23 Partnern. Alle Teilnehmer können sich damit auf das beste JavaLand-Programm aller Zeiten freuen.

Das komplette JavaLand-Programmkomitee

7. September 2018

Thomas Kurian, President of Product Development und einer der führenden Leitungskräfte bei Oracle, gibt in einer E-Mail bekannt, dass er der Firma für unbestimmte Zeit fernbleibe. Das ist keine gute Nachricht für die Oracle-Anwender, da Kurian für Oracle auf dem Weg in die Cloud eine wichtige Rolle spielt. Es hält sich das Gerücht, dass unterschiedliche Auffassungen über die zukünftige Cloud-Strategie Grund für die Trennung von Oracle ist. Kurian war seit dem Jahr 1996 bei Oracle tätig und Leiter der Software-Entwicklung. Eine Oracle-Sprecherin gibt bekannt, dass die Firma davon ausgeht, dass Kurian bald zurück sei, äußert sich aber nicht zu den Hintergründen.

11. September 2018

Die SOUG Romandie organisiert ihr zweites Meetup des Jahres. Bertrand Drouvot, Mitglied des „Oaktable", hält zwei Vorträge; der erste behandelt Automation im Datenbank-Umfeld. Es geht dabei um die automatische Software-Verteilung und Installation durch Open-Source-Komponenten wie Ansible. Der Vortrag deckt alle Bereiche bis auf die Automatisierung von betrieblichen Aktivitäten wie Überwachung etc. ab. Der zweite Vortrag zeigt verschiedene 12c-Konsolidierungs-Strategien mit deren Vor- und Nachteilen. Die fünfzehn Teilnehmer sind begeistert und schließen den Event mit einem Apéro ab.

Konzentrierte Atmosphäre beim Meetup der SOUG Romandie

12. September 2018

Der Vorstand der Austrian Oracle User Group (AOUG) tagt, um die vergangene Anwenderkonferenz noch einmal Revue passieren zu lassen und bereits die ersten Aktivitäten für die AOUG Anwenderkonferenz 2019 zu planen.

14. September 2018

Das Organisations-Team der DOAG stimmt sich in einer Telefonkonferenz mit dem Nürnberger Convention Center ab, um die Durchführung der DOAG 2018 Konferenz + Ausstellung zu finalisieren. Fried Saacke, DOAG-Vorstand und Geschäftsführer, ist überzeugt, auch in diesem Jahr wieder optimale Rahmenbedingungen für die Jahreskonferenz der DOAG bieten zu können.

17. September 2018

Dr. Dietmar Neugebauer, ehemaliger Vorstandsvorsitzender der DOAG, stimmt in einer Telefonkonferenz die Rechtsanwälte des DOAG Legal Council auf die DOAG 2018 Konferenz ein. Sie widmen sich in zwei Podiumsdiskussionen zahlreichen rechtlichen Themen, darunter „Lizenzierung bei Virtualisierung/VMware“ sowie „Audits, Datenschutz, Cloud-Verträge, Lizenzübertragung und mehr“. Zudem engagiert sich jeder Anwalt auch mit ein bis zwei Vorträgen.

18. September 2018

Die Community der Swiss Oracle User Group trifft sich in Baden Dättwil in den Räumen von Oracle zum SOUG Day 2018. Auch in diesem Jahr bieten exklusive Fachvorträge und praxisnahe Erfahrungsberichte von Experten den zahlreichen Teilnehmern Einblicke in aktuelle Themen rund um Database, DevOps und Cloud. Keynote-Speaker Markus Michalewicz, Senior Director von Oracle, kommt aus den USA, um mit dem spannenden Thema „Von HA zu MAA – eine holistisch historische Betrachtung“ den Auftakt zur lehrreichen Nachmittagsveranstaltung zu machen. In den Kaffeepausen trifft man sich zum inspirierenden Austausch oder nutzt die Gelegenheit, sich Fragen von den Experten beantworten zu lassen. Nach den anregenden Vorträgen lässt die SOUG-Community den Abend bei einem Apéro ausklingen. Die Präsentationen der Experten stehen den SOUG-Mitgliedern auf der SOUG-Webseite zum Download zur Verfügung.

20. September 2018

Die Austrian Oracle User Group (AOUG) widmet sich im Rahmen eines technischen Frühstücks dem Thema „Veeam-Backup für Oracle-Datenbanken“. Andreas Neufert von Veeam Software GmbH präsentiert in Zusammenarbeit mit DBConcepts GmbH das brandneue Veeam RMAN Plug-in für Oracle-Datenbanken sowie das Veeam Image Level Backup.

20./21. September 2018

Die fünften DOAG Big Data Days finden in Dresden statt. Neben den durchgängigen Big-Data-Tracks am ersten und zweiten Tag gibt es parallel dazu den Reporting Day sowie den Geodata Day. Der dreiteilige Workshop „Oracle Big Data Connectors Oracle Big Data SQL“ durch Dr. Nadine Schöne, Detlef Schröder und Gavin Dupre von Oracle rundet die Veranstaltung ab. Die rund 50 Teilnehmer können wie immer frei zwischen den Tracks wechseln. Zahlreiche spannende Vorträge sorgten für ein positives Feedback zum Abschluss an die Veranstaltung. Zudem profitiert das Networking von den vielen persönlichen Begegnungen, insbesondere während des Abendevents.

21. September 2018

Der DOAG-Vorstand hält eine mehrtägige Sitzung ab. Im Mittelpunkt steht die strategische Entwicklung des Vereins für die nächsten Jahre. Grundlage dafür ist das Ergebnis des Arbeitskreises der Delegiertenversammlung zur künftigen Struktur der DOAG. Wesentliche Erkenntnis des Vorstands ist, dass die DOAG mehr über den Tellerrand hinausblicken und sich auch Technologien stellen muss, die in Bezug zu Oracle-Lösungen stehen. Anders als Oracle-Chef Larry Ellison sich das vorstellt, ist der DOAG-Vorstand davon überzeugt, dass sich die Oracle-Cloud-Lösungen überwiegend im hybriden Umfeld durchsetzen werden.

26./27. September 2018

Das Berliner Expertenseminar mit Marco Patzwahl zum Thema „Backup & Recovery“ ist sehr gut besucht. Es zeigt, dass selbst sogenannte „alte Themen“ immer noch von großem Interesse sind.

1. Oktober 2018

Oracle teilt mit, dass wie viele schon vermutet haben, Thomas Kurian, President of Product Development und einer der führenden Leitungskräfte bei Oracle, das Unternehmen endgültig verlassen hat.

18. Oktober 2018

Fried Saacke, DOAG-Vorstand und Geschäftsführer, und Wolfgang Taschner, Chefredakteur der DOAG-Zeitschriften, treffen sich im Berliner DOAG-Büro mit Mylène Diacquenod, um die Übergabe des Red Stack Magazin, der Java aktuell und der DOAG Business News an die Kommunikationsabteilung der DOAG zu besprechen. Die drei Zeitschriften werden im Januar 2019 von Lisa Damerow, Sanela Lukavica und Martin Meyer übernommen. Wolfgang Taschner begleitet ihre Arbeit dann noch bis zum 31. März 2019, um einen fließenden Übergang zu gewährleisten.

22. Oktober 2018

Wolfgang Taschner, Chefredakteur der DOAG-Zeitschriften, moderiert in Vertretung des regionalen Repräsentanten, Andreas Ströbel, das Regiotreffen München/Südbayern. Jürgen Haas vom Oracle-Support referiert über die beiden Themen „Support Accreditation“ sowie „Get Proactive“ und gibt wertvolle Tipps im Umgang mit den Support-Mitarbeitern. Durch seine kompetente und anwenderfreundliche Art kann Jürgen Haas neue Sympathien für die Oracle-Support-Organisation aufbauen.

22. Oktober 2018

Oracle-Chef Larry Ellison eröffnet mit seiner Keynote die Oracle OpenWorld in San Francisco. Die DOAG entsendet in diesem Jahr keine Repräsentanten mehr, weil Oracle die Möglichkeiten zur Präsentation der Usergroups sehr stark eingeschränkt hat, sodass Kosten und Nutzen damit in keinem Verhältnis mehr stehen. Deshalb verfolgen manche DOAG-Aktive die Keynote in ihrem Web-Browser. Niemand glaubt so richtig an die Strategie, sich mit monolithischen Cloud-Lösungen auf

dem Markt durchsetzen zu können. Vielmehr deutet viel darauf hin, dass Oracle mit starken Partnern in der Cloud zusammenarbeiten muss.

23./24. Oktober 2018

In der DOAG-Konferenz-Lounge in Berlin findet das Expertenseminar zum Thema „Hochverfügbarkeit" statt. Ziel der Expertenseminare ist es, zu verschiedenen Themen tiefreichende Informationen zu vermitteln, was bei normalen Vorträgen auf Konferenzen und Tagungen in diesem Detaillierungsgrad nicht möglich ist. Das Hochverfügbarkeits-Expertenseminar wird in diesem Jahr nach 2011 und 2015 bereits zum dritten Mal von Robert Bialek und Mathias Zarick geleitet. Dass es in diesem Bereich immer wieder Neues gibt, ist allein daran zu erkennen, dass die Anzahl der Slides von anfänglich 440 auf fast 600 in diesem Jahr gestiegen ist, dazu kommen natürlich noch die verschiedensten Livedemos. Auch bei den Teilnehmern stieß das Thema auf großes Interesse, ist es doch mit zwölf Teilnehmern eines der bestbesuchten Expertenseminare der DOAG in der letzten Zeit, wobei sowohl Wiederkehrer als auch Erstmalige begrüßt werden können. Der Schwerpunkt liegt am ersten Tag bei der Grid Infrastructure mit Unterthemen wie Architecture, New Features, Backup & Recovery und Monitoring sowie beim Cluster Database Management. Nach einem arbeitsintensiven Tag steht ein gemeinsames Network-Dinner in einem rustikalen Lokal im traditionsreichen Nikolaiviertel auf dem Programm. Der zweite Seminartag bringt dann Themen wie „Database Upgrades in HA Environments", „Standby DB Management" und „Maximum Availability Architecture". Fehlen darf natürlich auch nicht die Container DB Architecture in einem HA Environment und, was man nicht vergessen darf, wie die Client Configuration in einer HA-Umgebung aussieht, wo es mittlerweile viele verschiedene Möglichkeiten gibt, die leider viel zu selten genutzt werden. Trotz der vielen Folien, Demos und Diskussionen schaffen es Robert Bialek und Mathias Zarick, das Seminar pünktlich mit der letzten Folie zu beenden, sodass alle Teilnehmer ihre Rückreise mit einer Vielzahl von neuen Informationen pünktlich antreten können.

Viel Raum für jeden Teilnehmer auf dem Expertenseminar

Dr. Dietmar Neugebauer
Ehemaliger DOAG-Vorstandsvorsitzender

Aus der Ferne betrachtet: Open Source – nur für Bastler und ohne Gewähr?

Lange Zeit klebte an den Open-Source-Produkten der Makel einer Software, die freiwillig von Bastlern in ihrer Freizeit und ohne Gewähr entwickelt worden ist. Für produktive Anwendungen viel zu gefährlich und völlig ungeeignet. Erst mit dem Siegeszug von Linux hat sich diese Einstellung langsam geändert und Linux hat sich dort als Betriebssystem auf verschiedensten Infrastrukturen durchgesetzt. Nicht von ungefähr hat jetzt IBM mit Red Hat die größte Open-Source-Firma der Welt für 30 Milliarden Euro übernommen. Hinzu kommt inzwischen auch eine riesengroße Zahl von Java-Entwickler, die mit OpenJDK- und OpenJRE-Software für die unterschiedlichsten Plattformen entwickeln.

Lange Zeit war man der Meinung, dass eine ausgereifte Datenbank-Technologie nur von den großen Herstellern wie IBM, Oracle und Microsoft kommen kann. Den Open-Source-Produkten wurde in diesem Markt nur eine kleine Nische vorausgesagt. Aber auch dies hat sich inzwischen komplett geändert. Immer mehr Anwender setzen Produkte wie PostgreSQL oder MariaDB ein und migrieren ihre Datenbanken auf diese Umgebungen. Damit werden hohe Lizenzgebühren eingespart und auch den Einsatz in produktiven Umgebungen lassen sich große Firmen mit Support-Verträgen zu diesen Open-Source-Produkten garantieren.

Es gibt kein Monopol oder Alleinstellungsmerkmal für Software-Firmen. Die langjährigen Kundenbindungen werden aufgelöst und dem Angebot auf dem Markt angepasst. Der Schnellere und Flexiblere gewinnt gegen den Langsamen und Trägeren. Ob der Größere den Kleineren schluckt, ist noch abzuwarten. Zumindest ist es hier nur schwer verständlich, warum Oracle auf seiner nur auf die eigenen Produkte eingeschränkten Strategie beharrt. Bei anderen Cloud-Anbietern kann man inzwischen ein weites Spektrum an Open-Source-Produkten mieten, auch Open-Source-Datenbanken wie PostgreSQL und MariaDB – erstaunlicherweise alles außer einer mit allen gängigen Features ausgestatteten Oracle-Datenbank. Dies wiederum versucht Oracle, warum auch immer, bei anderen Cloud-Anbietern zu verhindern.

Doch Kunden, die Anwendungen On-Premises, als Hybrid-Lösung oder in der Cloud betreiben, wollen keine Vendor-bezogenen Einschränkungen akzeptieren. Für sie gilt heute schon, Open Source ist über alle Plattformen eine gewollte und beabsichtigte Alternative – auch für Produktiv-Systeme und mit Support-Garantie!

Fried Saacke (rechts) im Gespräch mit Dr. Dietmar Neugebauer

„Die Probleme der Kunden verstehen und ernst nehmen …“

Fried Saacke, DOAG-Vorstand und Geschäftsführer, ist seit mehr als zwanzig Jahren für die DOAG aktiv. Dr. Dietmar Neugebauer, ehemaliger Vorstandsvorsitzender, und Wolfgang Taschner, Chefredakteur des Red Stack Magazin, sprachen mit ihm über den Verein.

Was bedeutet die DOAG für dich?

Fried Saacke: Die DOAG ist im Lauf der Jahre zu einer Lebensaufgabe für mich geworden, von daher ist es für mich nicht nur wichtig, wie die Dinge heute laufen, sondern den Verein auch für die Zukunft gut auszurichten. Die enge Bindung kann natürlich auch zu emotionalen Reaktionen führen, beispielsweise als der Vorstand den Beschluss fasste, unseren Videokanal DOAG.TV einzustellen. Das konnte ich in keiner Weise nachvollziehen, weil DOAG.TV weltweit in der Community bekannt und anerkannt ist.

Wie bist du zur DOAG gekommen?

Fried Saacke: Ich hatte mich beruflich bereits längere Zeit mit Oracle-Technologien beschäftigt und im Jahr 1996 den Auftrag, für meinen damaligen Arbeitgeber eine Geschäftsstelle in Berlin aufzubauen. Dabei bin ich auf einen Flyer der DOAG gestoßen, der einer Lieferung von Oracle beilag. Daraufhin habe ich noch im selben Jahr erstmals die DOAG Konferenz besucht.

Was waren deine ersten Aktivitäten bei der DOAG?

Fried Saacke: Auf der DOAG Konferenz hat mich ein früherer Arbeitskollege, der bei der DOAG aktiv war, darauf angesprochen, ob ich denn die DOAG-Regionalgruppe Berlin gründen wolle, wenn ich denn schon dort arbeite. Ich habe dann nach einer ausführlichen Information über die Abläufe die Fäden in die Hand genommen und die Regionalgruppe Berlin-Brandenburg ins Leben gerufen.

Wie ging es dann weiter?

Fried Saacke: Damals wurden die Regionalleiter noch zur Vorstandssitzung der DOAG eingeladen. In deren Rahmen ging es darum, ein Gespräch mit Oracle vorzubereiten, um deren Sponsoring für den Verein fortzuführen. Ich war dann am nächsten Tag in der Oracle-Geschäftsstelle in München beim Treffen mit dem Marketingleiter Claus-Peter Unterberger dabei. Er machte uns im Anschluss an unseren Vortrag deutlich, dass er uns kein Geld geben wird, solange wir keine klare Strategie für den Verein hätten. Ich habe mich daraufhin für eine Arbeitsgruppe gemeldet, um die Strategie der DOAG auszuarbeiten. Dabei arbeitete ich schon sehr eng mit dem damaligen DOAG-Vorstand zusammen und als dann im Jahr 1999 Neuwahlen im Vorstand anstanden, wurde ich gefragt, ob ich denn nicht kandidieren wolle. Nach meiner anschließenden Wahl in den Vorstand wurde ich auch gleich gebeten, den Vorsitz in diesem Vorstand zu übernehmen, was ich dann auch gemacht habe.

Was waren deine ersten Aktivitäten als Vorstandsvorsitzender der DOAG?

Fried Saacke: Um mich in dieses neue Amt einzufinden, bestand mein erster Schritt darin, Gespräche mit früheren Vorstandsvorsitzenden zu führen. Insbesondere Agnes Hombrecher hat mir dabei sehr bei der Orientierung geholfen. Dietmar Neugebauer, der zuvor als stellvertretender Vorsitzender die Amtsgeschäfts in die Hand genommen hatte, hat mir dann bei einem Termin einen Ordner übergeben mit allem, was er an Unterlagen hatte. Er hat mir dann viele Informationen mit auf den Weg gegeben und später auch dabei unterstützt, als es Probleme mit der Beziehung zu Oracle gab. Die DOAG wurde zum damaligen Zeitpunkt, angefangen bei den Veranstaltungen bis hin zur Buchführung, stark von Oracle organisiert. Ich habe sehr schnell gemerkt, dass das keine Zukunft hat, weil Oracle beispielsweise bei meiner ersten Konferenz als Vorstandsvorsitzender von sich aus ohne Rücksprache einfach das Vortragsprogramm geändert hat.

Welche Konsequenz hast du daraus gezogen?

Fried Saacke: Ich habe in meinem Unternehmen zwei Mitarbeiterinnen ausgewählt, für die die DOAG die Kosten übernommen hat, um mit ihnen die DOAG-Geschäftsstelle einzurichten. Wir haben alle Unterlagen bei Oracle abgeholt und diese sukzessive ausgewertet, um einen tieferen Einblick in die Dinge zu bekommen. Nach Rücksprache mit dem Vorstand habe ich mit Carsten Diercks einen vereinsaffinen Anwalt gewinnen können, der uns auch heute noch in Rechtsangelegenheiten zur Verfügung steht und mit dem ich dann ab dem Jahr 2004 die Gründung der DOAG Dienstleistungen GmbH vorbereitete, um die organisatorischen Tätigkeiten aus dem Verein auszulagern. Am 3. November 2004 konnten wir dann die GmbH nach einer vorbereitenden Satzungsänderung auf der Mitgliederversammlung 2003 und weiteren Vorbereitungen in einem Berliner Notariat endlich gründen.

Wie hat sich die DOAG Dienstleistungen GmbH seitdem entwickelt?

Fried Saacke: Die Weiterentwicklung lief immer parallel zum Verein. Das enorme Mitgliederwachstum der DOAG und die zunehmende Anzahl von Veranstaltungen führten auch zur Personalentwicklung in der Geschäftsstelle. Analog zu den Aktivitäten des Vereins ist die Geschäftsstelle in die Bereiche „Veranstaltungen", „Publikationen" und „Web-Services" aufgeteilt. Hinzu kommen die ganzen Dienstleistungen hinsichtlich IT und Buchhaltung. Mittlerweile sind fünfundzwanzig Mitarbeiterinnen und Mitarbeiter in der DOAG Dienstleistungen GmbH angestellt, darunter drei Auszubildende.

Du hast es erstmals geschafft, dass die DOAG Konferenz einen Überschuss erzielte. Wie ist dir das gelungen?

Fried Saacke: Bis zum Jahr 2006 lag die wirtschaftliche Verantwortung bei Gerhard Schreiber, dem damaligen Geschäftsführer der Dienstleistungen GmbH, der die DOAG Konferenz zusammen mit dem Vorstand Dieter Ketterle organisierte. Gesamtwirtschaftlich betrachtet war die Veranstaltung unter Vollkosten betrachtet nur durch Zuschüsse aus dem Verein zu finanzieren, was auf die Dauer nicht tragbar ist. Der zwischenzeitlich wieder im Vorstand aktive Dr. Dietmar Neugebauer hat nach langen internen Diskussionen schlussendlich mich gebeten, den Vorstandsvorsitz abzugeben und selbst die Geschäftsführung der Dienstleistungen GmbH zu übernehmen, und ich habe dem Vorschlag zugestimmt. Im Gegenzug ist Dr. Dietmar Neugebauer Vorstandsvorsitzender geworden. Die erste Konferenz unter meiner Verantwortung fand im Jahr 2008 in Nürnberg statt. Ich habe deren Finanzierung komplett neu gestaltet, mit allen Partnern neue Verträge verhandelt und dort, wo es notwendig war, die Partner gewechselt. Seitdem erwirtschaftet die DOAG Konferenz deutliche Überschüsse, die zur Finanzierung der verschiedenen Vereinsaufgaben eingesetzt werden.

Du und Dietmar wart lange Zeit das Dream-Team der DOAG. Was habt ihr in eurer gemeinsamen Zeit alles erreicht?

Fried Saacke: Ein erster gemeinsamer wichtiger Schritt war die Auslagerung der DOAG Konferenz in eine eigene GmbH zur wirtschaftlichen Stabilisierung und zur Trennung der geschäftlichen Risiken. Dann haben wir uns sehr viele Gedanken darüber gemacht, wie wir die DOAG für die Zukunft richtig aufstellen. Daraus resultierten zwei Satzungsänderungen, mit denen der Verein hinsichtlich seiner Strukturen viel flexibler wurde und die dazu führten, dass heute die Delegierten beziehungsweise die Delegiertenversammlung die Strategie des Vereins festlegen. Darüber hinaus hat uns die sich ändernde Geschäftspolitik von Oracle veranlasst, in der DOAG zunächst vier thematische orientierte Communities einzurichten, die später auf sechs und zuletzt auf sieben erweitert wurden, und die alle eigenständig in ihrem Bereich agieren können. Nicht zuletzt haben wir bereits im Jahr 2008 damit begonnen, die DOAG international auszurichten und die weltweiten Anwendergruppen untereinander zu vernetzen. Heute ist die DOAG selbst in den Oracle-Headquarters bekannt und spielt unter den europäischen Anwendergruppen eine führende Rolle. Lediglich den Dialog mit Oracle konnten wir nicht entscheidend verbessern, weil das Unternehmen permanent die Diskussion mit den Anwendergruppen verweigert und in keiner Weise daran interessiert ist, das Feedback der Anwender in die Produktentwicklung einfließen zu lassen.

Dr. Dietmar Neugebauer: Obwohl wir beide nach außen meist mit einer gemeinsamen Stimme aufgetreten sind, hatten wir intern oft unterschiedliche Meinungen, die wir dann so lange konstruktiv diskutiert haben, bis der richtige Weg für die DOAG klar war. Den sind wir dann auch gemeinsam gegangen.

Wie hat sich Oracle während deiner aktiven Zeit bei der DOAG entwickelt?

Fried Saacke: Aus meiner Sicht hat sich Oracle gravierend verändert, getrieben aus dem ständigen Bestreben, in allen Geschäftsbereichen die Nummer eins sein zu wollen. Der Versuch, vom Lieferanten für Technologie-Lösungen zum Anbieter für Unternehmens-Anwendungen zu werden, ist nicht gelungen. Die Firmenpolitik wird immer mehr aus den Headquarters in den USA bestimmt und unsere Ansprechpartner in Deutschland verlieren an Einfluss. Für Probleme, die Anwender in Europa haben, gibt es keine Anlaufstelle mehr.

Nach der Übernahme von Sun hast du den Interessenverbund der Java User Groups e.V. gegründet und die Zeitschrift Java aktuell herausgegeben. Was war deine Motivation dafür?

Fried Saacke: Zwischen den Jahren 2000 und 2010 hat Oracle ja fast jeden Monat ein Unternehmen übernommen, darunter große Firmen wie BEA, Siebel und PeopleSoft. Bei jeder dieser Übernahmen haben wir sofort versucht, Kontakt zu User Groups in diesem Umfeld aufzunehmen. In den meisten Fällen trafen wir auf keine Anwendervertretung, so wie wir sie verstehen, sondern lediglich auf vom Hersteller organisierte Marketing-Gruppierungen. Bei Sun war das insbesondere im Java-Umfeld anders, hier gab es viele selbstorganisierte, regional aktive Gruppen, die sich untereinander ausgetauscht haben. Ich verfolgte von Anfang an die Idee, diese Gruppen unter dem Dach der DOAG zusammenzubringen. Dazu habe ich in Frankfurt ein Treffen organisiert, zu dem alle uns bekannten Java User Groups eingeladen waren. Schnell war klar, dass die Gruppen niemals unter dem Dach einer DOAG zusammengehen würden; dafür waren wir zu rot. Daher habe ich dann die Gründung eines eigenständigen Interessenverbunds vorgeschlagen, in dem die DOAG ein starkes Mitglied wird. Daraus resultierte schließlich der Interessenverbund der Java User Groups. Aus den ursprünglich sieben Gründungsmitgliedern sind heute knapp vierzig Java User Groups aus Deutschland, Österreich und der Schweiz geworden. Ich hätte diesen Erfolg nie für möglich gehalten. Eines der ersten großen gemeinsamen Projekte war die Gründung der Zeitschrift Java aktuell, die sich erfolgreich auf dem deutschsprachigen Zeitschriftenmarkt etabliert hat.

Die später von dir ins Leben gerufene JavaLand ist mittlerweile als eine der erfolgreichsten Java-Konferenzen in Europa etabliert. Worauf führst du das zurück?

Fried Saacke: Es war sicher ein Glücksfall, dass ich auf der Suche nach einer geeigneten Location das Phantasialand in Brühl entdeckte, während Markus Eisele, der sehr aktiv auf allen Java-Konferenzen unterwegs war, als Zuständiger für das Vortragsprogramm seine ganzen Erfahrungen einbrachte. Von Tobias Frech von der Java User Group Stuttgart kam darüber hinaus die Idee mit zahlreiche Community-Aktivitäten und -Treffen, die parallel zu den Vorträgen laufen. Genau diese Mischung aus dem spektakulären Veranstaltungsort, dem hervorragend bestückten Konferenzprogramm und den integrierten Community-Aktivitäten sowie der aktiven Einbindung der gesamten deutschsprachigen Java-User-Group.Community macht den großen Erfolg der JavaLand aus.

Wie beurteilst du die heutige Strategie von Oracle, alles in die Cloud verlagern zu wollen?

Fried Saacke: Oracle folgt hier einem Markt-Trend, dem sie lange Zeit keine Beachtung geschenkt hatten. Das ist erst mal nicht verkehrt, das Problem ist allerdings, dass Oracle das jetzt mit der eigenen monolithischen Technologie schaffen möchte und nicht bereit ist, mit entsprechenden Partnern zusammenzuarbeiten. Gerade bei der Cloud-Technologie möchte der Kunde jedoch selbst bestimmen, welche Leistungen er von welchem Dienstleister bezieht, und denkt nicht daran, sich hier von einem einzigen Anbieter, wie in diesem Fall Oracle, abhängig zu machen. Oracle versucht dennoch, die Kunden unter Druck zu setzen, wie wir in der DOAG immer wieder zu hören bekommen, aber das ist für mich das falsche Signal.

Wie sieht für dich die Zukunft der DOAG aus?

Fried Saacke: Das Allerwichtigste für mich ist, die DOAG für die Zukunft weiterhin gut aufzustellen und zu positionieren. Das bedeutet, neben der wirtschaftlichen Unabhängigkeit auch die fachliche Kompetenz entsprechend den Bedürfnissen der Anwender zu wahren. Es wird dabei immer wichtiger, auch Anbieter und Anwender anderer Technologien einzubeziehen.

Was sind die großen Herausforderungen der DOAG in den nächsten Jahren?

Fried Saacke: In der von Oracle angestrebten monolithischen Technologiewelt gibt es nur noch wenige Ansatzpunkte für eine Anwendergruppe. Für die DOAG ist es deshalb enorm wichtig, die Probleme der Kunden zu verstehen, um nachhaltig deren Interessen vertreten zu können. Dazu gehört auch der Blick über den Tellerrand hinaus, um den Anwendern auf unseren Plattformen vermitteln zu können, wie man Oracle mit den Produkten und Lösungen anderer Hersteller gut verbinden kann.

Dietmar, in welcher Form bist du als Ruheständler noch für die DOAG aktiv? Wie gefällt dir die heutige DOAG? Was machen deine mittlerweile fünf Enkelkinder?

Dr. Dietmar Neugebauer: Mir war immer das Thema „Lizenzierung" sehr wichtig. Deshalb habe ich mich auch im Legal Council der DOAG als Moderator und Ansprechpartner für die DOAG-Mitglieder engagiert. Darüber hinaus arbeite ich an der ORAWORLD mit, einem elektronischen Magazin für die Mitglieder aller europäischen Oracle-Anwender. Ich sehe die DOAG heute auf einem guten Weg und bin mit Fried der gleichen Meinung, dass sich der Verein auch für die Lösungen anderer Anbieter öffnen muss. Gerade der Blick von außen kann für die Oracle-Anwender sehr hilfreich sein. Meine Enkelkinder machen mir sehr viel Freude. Dabei stelle ich auch einen „Technologiewechsel" zu meiner Kindheit fest. Sie lassen sich gerne von mir Bilderbücher zeigen und Geschichten vorlesen, auf der anderen Seite lernen sie jetzt schon, mit einem Smartphone oder einem Tablet umzugehen.

Referenzen

- Interview mit Dr. Dietmar Neugebauer, Red Stack Magazin, Ausgabe 03/2016, Seite 8

MySQL 8 XDevAPI: Neue Wege für Entwickler moderner Applikationen

Mario Beck und Carsten Thalheimer, Oracle MySQL Deutschland

NoSQL hat viele neue Ideen in den Datenbank-Bereich gebracht, etwa die Handhabung unstrukturierter Daten, simple APIs als Alternative zu SQL oder die Aufweichung strenger Anforderungen an Konsistenz zugunsten von Performance. MySQL 8 unterstützt viele dieser Konzepte und ermöglicht es so, das Beste aus beiden Welten in einem Datenbank-System zu nutzen: Beispiele sind MySQL als DocumentStore oder ein natives CRUD-API zum performanten Zugriff auf relationale Daten oder die Nutzung der Power von SQL, um nicht-strukturierte Daten dennoch effektiv in der Datenbank auswerten zu können.

Im Grunde beginnt die Geschichte bereits vor drei Jahren mit MySQL 5.7 und der Einführung des neuen Datentyps JSON, automatisch generierter, indizierbarer Spalten und einer Menge neuer SQL-Funktionen zur Manipulation von JSON-Daten. Seit MySQL 5.7 ist es möglich, Daten einer relationalen Tabelle im JSON-Format an die Applikation zu liefern *(siehe Listing 1)*. Unstrukturierte Daten können als JSON-Dokumente abgespeichert sein *(siehe Listing 2)*. Vor allem jedoch können seit MySQL 5.7 beide Welten verbunden und relationale Daten mit unstrukturierten Daten leicht kombiniert werden *(siehe Listing 3)*.

Das neue CLI: MySQL Shell

MySQL 8 geht einen Schritt weiter und bietet ein neues Protokoll (X-Protokoll) sowie ein neues API zur Nutzung der Datenbank (X DevAPI). Das neue X-Protokoll wird von allen aktuellen Konnektoren in MySQL unterstützt (Java, PHP, C, C++, Python, JavaScript, .NET). Es kann nun wahlweise das alte (SQL-) Protokoll oder das neue X-Protokoll verwendet werden *(siehe Abbildung1)*.

Das neue CLI unterstützt ebenfalls beide Protokolle und ist somit eine Obermenge des alten „mysql"-CLI. Bei neuerem MySQL 5.7 und MySQL 8 wird schwerpunktmäßig die neue MySQL-Shell „mysqlsh" anzutreffen sein. Bisher

```
mysql> SELECT json_object("family", name, "first", vorname) FROM foo WHERE id=2;
+-----------------------------------------------+
| json_object("family", name, "first", vorname) |
+-----------------------------------------------+
| {"first": "Micky", "family": "Maus"}          |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

Listing 1

```
mysql> SELECT * FROM restaurants LIMIT 1\G
*************************** 1. row ***************************
doc: {"_id": "564b3259666906a86ea90a99", "name": "Dj Reynolds Pub And Restaurant", "grades": [{"date":
{"$date": 1409961600000}, "grade": "A", "score": 2}, {"date": {"$date": 1374451200000}, "grade": "A", "score":
11}, {"date": {"$date": 1343692800000}, "grade": "A", "score": 12}, {"date": {"$date": 1325116800000},
"grade": "A", "score": 12}], "address": {"coord": [-73.98513559999999, 40.7676919], "street": "West   57
Street", "zipcode": "10019", "building": "351"}, "borough": "Manhattan", "cuisine": "Irish", "restaurant_id":
"30191841"}
_id: 564b3259666906a86ea90a99
1 row in set (0.00 sec)
```

Listing 2

ist dieser Client allerdings noch ein separates zu installierendes Paket. Beim Verbindungsaufbau wählt man bereits über den anzusprechenden Port das gewünschte Protokoll. Der Server kann über die Variable „@@mysqlx_port“ beliebig konfiguriert werden (Default 33060, *siehe Listing 4)*.

Das neue Passwort-Management fällt beim Anmelden sofort auf: MySQL Shell kann verwendete Passwörter in einem sicheren Passwort-Speicher lagern und bei Bedarf wiederverwenden. Dabei kann zwischen dem üblichen Login-Path (Plattform-übergreifend) oder OS-abhängigen Keystores (OS-X Keychain, Windows Credential Manager) gewählt werden.

```
mysql> SHOW CREATE TABLE products\G
*************************** 1. row ***************************
       Table: products
Create Table: CREATE TABLE `products` (
  `id` int(11) NOT NULL,
  `name` varchar(20) DEFAULT NULL,
  `price` int(11) DEFAULT NULL,
  `attributes` json DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB …
1 row in set (0.00 sec)
```

Listing 3

MySQL Shell hat drei Modi (SQL, JavaScript und Python), zwischen denen mit den neuen Backslash-Direktiven „\js“, „\sql“ und „\py“ gewechselt werden kann. Der SQL-Mode verhält sich nahezu identisch zum alten MySQL-CLI. Der neue Java-

```
$ mysqlsh root@localhost:33060
Creating a session to 'root@localhost:33060'
Please provide the password for 'root@localhost:33060': ****
Save password for 'root@localhost:33060'? [Y]es/[N]o/Ne[v]er (default No): y
…
Your MySQL connection id is 8 (X protocol)
…
Type '\help' or '\?' for help; '\quit' to exit.

 MySQL  localhost:33060+ ssl  JS > \sql
Switching to SQL mode... Commands end with ;

 MySQL  localhost:33060+ ssl  SQL > show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| json_restaurants   |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.0008 sec)
MySQL  localhost:33060+ ssl  SQL > \py
Switching to Python mode...

 MySQL  localhost:33060+ ssl  Py > \js
Switching to JavaScript mode...
```

Listing 4

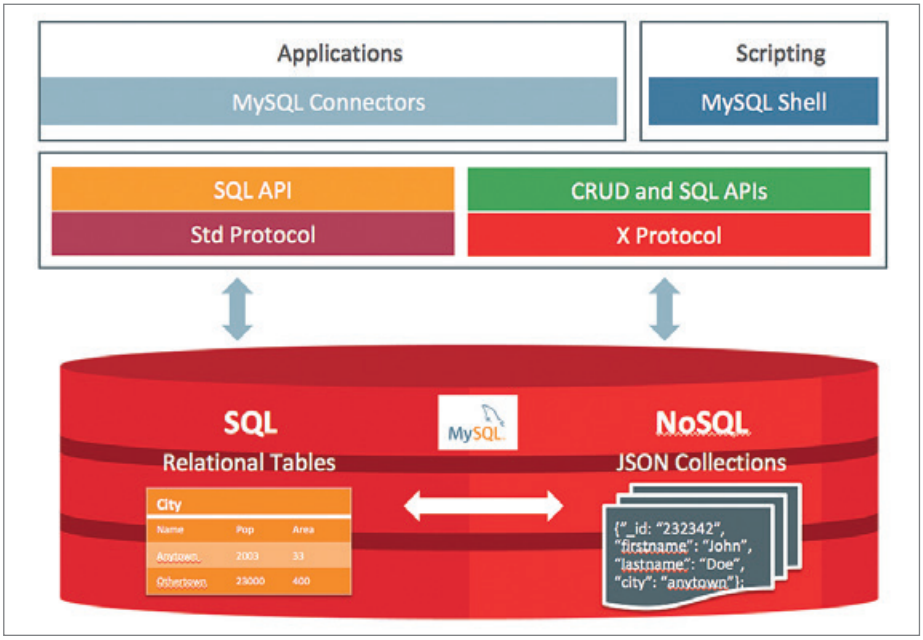


Abbildung 1: MySQL-Architektur

Script-Mode birgt mehr Neuerungen und wird durch die weiteren Beispiele begleiten. In Applikationen anderer Sprachen (PHP, Java, .NET etc.) sind die neuen API-Methoden jeweils vergleichbar aufgebaut. Dieser Artikel beschränkt sich auf die JavaScript-Notation. Alle Beispiele sind auf „mysqlsh/MySQL 8.0.12“ ausgeführt worden. Nach dem Login ist JavaScript der Default-Modus. Es sind einige globale Objekte automatisch definiert: „session“ beschreibt die aktuelle Verbindung, „db“ die aktuell gewählte Datenbank und „dba“ wird verwendet, um beispielsweise einen InnoDB Cluster zu administrieren. Ausgehend von diesen globalen Objekten lassen sich dann Benutzerobjekte erzeugen und manipulieren.

MySQL als DocumentStore

Als DocumentStore werden keine strukturierten Tabellen vom Anwender erzeugt, sondern lediglich Collections, die beliebig strukturierte JSON-Dokumente aufnehmen *(siehe Listing 5)*. Die Daten werden weiterhin in MySQL unter der InnoDB Storage Engine gespeichert. Dort erscheint die Collection als eine gewöhnliche, relationale Tabelle mit zwei Spalten: einer JSON-Spalte, die das Dokument enthält, und einer automatisch generierten Spalte mit dem „_id“-Feld, das jedem Dokument einen eindeutigen Identifier gibt. Bleiben wir aber zunächst auf der DocumentStore-Seite des Geschehens. Die üblichen CRUD-Operationen („Create“, „Read“, „Update“ und „Delete“) haben im DocumentStore-API ihre Entsprechung in den Methoden „collection.add()“, „collection.find()“, „collection.modify()“ und „collection.remove()“. Einige Beispiele dazu stehen in *Listing 6*.

Für performante Zugriffe ist es natürlich auch wichtig, sekundäre Indizes zu erzeugen. Dazu gibt es die Methode „collection.createIndex“, also „db.posts.createIndex('title_idx', {fields: [{field: "$.title", type: "TEXT(20)"}]})“. Das Index-Feld wird als Array übergeben. Daran ist bereits abzulesen, dass auch Compound-Indizes, also Indizes, die aus mehreren Feldern aufgebaut sind, erzeugt werden können. Nun ist es wieder interessant, das Geschehen auf SQL-Seite zu verfolgen. Auch im JavaScript-Mode ist es möglich, traditionelle SQL-Kommandos wie „session.sql("DESC posts")“ oder „session.sql("SHOW CREATE TABLE posts")“ zum Server zu schicken.

Die der Collection zugrunde liegende Tabelle hat eine weitere Spalte erhalten. Deren Daten werden automatisch aus der Expression „doc->$.title“ generiert und enthalten also den Wert des „title“-Attributs der JSON-Dokumente (oder NULL, falls das Attribut nicht im Dokument existiert). Außerdem ist diese Spalte indiziert. Der Optimizer in MySQL wird diesen Index automatisch verwenden, wenn Queries mit dem Prädikat „doc->$.title="XYZ"“ versehen sind. Entwickler müssen also diese Index-Spalte nicht berücksichtigen; das übernimmt der Optimizer automatisch.

Transaktionen im DocumentStore

Da die Daten in der normalen InnoDB-Engine gespeichert sind, stehen auch alle Eigenschaften der InnoDB im DocumentStore zur Verfügung; so auch Transaktionen: Es ist leicht möglich, Änderungen an Dokumenten oder ganzen Collections im Rahmen einer Transaktion zusammenzufassen und diese gemeinsam per „COMMIT“ auszuführen oder durch „ROLLBACK“ zurückzurollen. Der MySQL DocumentStore ist automatisch voll ACID-konform. Das Beispiel in *Listing 7* verschiebt ein JSON-Dokument von einer Collection in eine zweite. Dank der Transaktionsklammer ist sichergestellt, dass

```
$ mysqlsh root@localhost:33060/demo
Creating a session to 'root@localhost:33060/demo'
…
Type '\help' or '\?' for help; '\quit' to exit.

 MySQL  localhost:33060+ ssl  demo  JS > db.createCollection("posts")
<Collection:posts>

 MySQL  localhost:33060+ ssl  demo  JS > db.posts.add({"title":"MySQL
8.0 rocks", "text":"My first post!", "code": "42"})
Query OK, 1 item affected (0.0851 sec)

 MySQL  localhost:33060+ ssl  demo  JS > db.posts.
add({"title":"Polyglot database", "text":"Developing both SQL and NoSQL
applications"})
Query OK, 1 item affected (0.0169 sec)
```

Listing 5

```
db.posts.find().limit(1)
db.posts.find("code='42'")

# modifying documents
db.posts.modify("code='42'").set("code", "43")
db.posts.modify("true").set("cool", "yes")
db.posts.modify("true").unset("cool")
db.posts.modify("title like 'MySQL%'").set("color",['red', 'blue', 'black'])
db.posts.find("'red' in color")
```

Listing 6

auch Collection-übergreifend entweder alle oder keines der Statements ausgeführt wird.

Natürlich kann ein Entwickler je nach Anwendungsfall auf diese Eigenschaften verzichten, um zum Beispiel zugunsten der Performance auf 100-prozentige Durability zu verzichten („flush_logs_at_trx_commit=2"). Entscheidend ist, dass alle Möglichkeiten zur Verfügung stehen und somit verschiedene Anwendungsfälle abgedeckt werden können.

```
Db.createCollection(„archive")
session.startTransaction()
t=db.posts.find(„code='43'").execute().fetchAll()
db.archive.add(t)
db.posts.remove(„code='43'")
db.posts.find()
db.archive.find()
session.rollback()
db.posts.find()
db.archive.find()
```

Listing 7

Komplexes Reporting im DocumentStore

Ein häufiges Problem beim Einsatz von NoSQL-Datenbanken besteht in der geschickten Auswertung von Daten, dem Reporting. Hier besitzt MySQL besondere Vorteile: Zum einen unterstützt das X-DevAPI bereits die üblichen Filter-Prädikate und Aggregationen. Hinweis: Die folgenden Beispiele basieren auf Beispieldaten von Restaurants, die bei Document-Datenbanken gern genutzt werden. Diese können in MySQL als JSON-Dokumente geladen werden *(siehe Listing 8)*. Bemerkenswert ist hier die einfache Lesbarkeit der Query.

Der zweite Aspekt ist die Möglichkeit, die Daten jederzeit mit den umfangreichen Möglichkeiten von SQL auszuwerten. Auch in SQL kann die oben stehende Query ausgeführt werden und direkt eine Document-Collection auswerten *(siehe Listing 9)*.

Aber auch die komplexeren SQL-Konstrukte stehen nun zur Auswertung von Document-Collections zur Verfügung. MySQL 8 unterstützt beispielsweise auch einfache und rekursive Common Table Expressions („WITH-Queries") und auch Window-Functions. Die bestbewerteten Restaurants je Cuisine lassen sich mit einem SQL-Statement ermitteln *(siehe Listing 10)*. Die vergleichbare Auswertung mit Bordmitteln einer DocumentStore-Datenbank ist erheblich schwieriger oder teilweise schlicht nicht möglich.

X-DevAPI für relationale Daten

Das neue API unterstützt nicht nur DocumentStore. Es lassen sich auch relationale Tabellen mit den nativen Methoden erzeugen und manipulieren. Dabei beschränkt sich das X-DevAPI auf die gewöhnlichen CRUD-Funktionen (Create, Read, Update, Delete). DDL, höherwertige komplexe Aufgaben wie Join-Operationen oder die Möglichkeiten über Common-Table-Expressions, Window-Functions oder einige Aggregationen werden weiterhin mithilfe der Methode „session.sql()" über SQL abgewickelt *(siehe Listing 11)*.

Mischung von Documents und relationalen Daten

Da auch Document-Collections im Hintergrund als Tabellen mit einfacher Struktur abgebildet werden, ist es möglich, Document-Collections und relationale Tabellen in der gleichen Datenbank

```
db.restaurants.find().fields(„$.cuisine", „count(,*')").groupBy(„$.cuisine").sort(„$.cuisine").execute()
```

Listing 8

```
SELECT doc->"$.cuisine", count(*) FROM restaurants
GROUP BY doc->"$.cuisine" ORDER BY doc->"$.cuisine";
```

Listing 9

zu speichern und diese gemeinsam in einer Applikation zu nutzen. Dies kann sogar so weit führen, dass Fremdschlüssel-Beziehungen zwischen Document-Collections und relationalen Tabellen erstellt werden können. Dazu wird zunächst das gewünschte Attribut in eine automatisch generierte Spalte der Document-Collection herausgezogen *(siehe Listing 12)*. Mithilfe dieser Spalte wird nun die Fremdschlüsselbeziehung definiert *(siehe Listing 13)*.

In diesem Beispiel erzwingt die Fremdschlüssel-Beziehung, dass nur noch JSON-Dokumente gespeichert werden können, deren Borough-Attribut auch in der referenzierten Tabelle enthalten ist. *Listing 14* zeigt einen kleinen Test.

Natürlich sind viele weitere Vorteile einer Mischung aus relationalen und dokumentbasierten Tabellen denkbar, etwa JOIN-Operationen, Fremdschlüssel, Views, die ein Reporting aus JSON-Documents vereinfachen, etc.

```
WITH cte1 AS
  (SELECT doc->>"$.name" AS name,
          doc->>"$.cuisine" AS cuisine,
          (SELECT AVG(score) FROM
              json_table(doc, "$.grades[*]"
                   COLUMNS (score INT PATH "$.score")) AS r
          ) AS avg_score FROM restaurants
  )
SELECT *,
       RANK() OVER (PARTITION BY cuisine ORDER BY avg_score) AS `rank`
FROM cte1
ORDER BY `rank`, avg_score DESC
LIMIT 30;
```

Listing 10

```
Session.sql("CREATE TABLE boroughs
             (name VARCHAR(20) PRIMARY KEY)")
db.boroughs.insert("name").values("Manhattan")
db.boroughs.insert("name").values("Staten Island")
i=db.boroughs.insert("name").values("Bronx")
i.values("Brooklyn").values("Queens").values("Missing")
db.boroughs.select().orderBy("name")
```

Listing 11

```
session.sql("ALTER TABLE restaurants ADD COLUMN borough varchar(20) AS (json_unquote(doc->'$.borough')) STORED")
```

Listing 12

```
session.sql("ALTER TABLE restaurants ADD FOREIGN KEY (borough) REFERENCES boroughs(name);")
```

Listing 13

```
db.restaurants.add({name:"Marios Pizza", borough:"Milan"})
ERROR: 1452: Cannot add or update a child row: a foreign key constraint fails…

db.restaurants.add({name:"Marios Pizza", borough:"Brooklyn"})
Query OK, 1 item affected (0.0453 sec)
```

Listing 14

Vorteile des X-DevAPI

Ein wichtiger Vorteil des neuen API besteht in der sauberen Trennung von Statement und Parametern. Während bei SQL die Applikation einen String erzeugt, der sowohl Befehle als auch Parameter enthält (etwa „INSERT INTO foo VALUES („42“);“), besteht beim X-DevAPI eine saubere Trennung zwischen Methoden und Parametern. Dies ist unter anderem der wirksamste Schutz gegen SQL-Injection, denn Anwender können nun nicht mehr Parameter eingeben, die versehentlich (oder absichtlich) als Statement interpretiert werden können.

Ein weiterer Vorteil des X-DevAPI besteht in der besseren Lesbarkeit von Code und der natürlicheren Verbindung von Applikation und Datenhaltung. Objekte können nun unkompliziert als JSON gespeichert und zurückgelesen werden. Eine oft komplexe Konvertierung in ein relationales Modell mittels Object-Relational-Mapping (ORM) lässt sich häufig vermeiden.

Ein dritter großer Vorteil des neuen API besteht in der nahtlosen Integration des alten SQL. Damit ist ein unkomplizierter Umstieg in das neue API möglich. Daten können jederzeit durch „session.sql(…)“ mit SQL-Statements manipuliert und die Übersichtlichkeit und Einfachheit des CRUD-API kann problemlos mit der Komplexität und Macht von SQL beim Reporting verbunden werden. Zu jedem Zeitpunkt lässt sich für den Anwender das Beste aus beiden Welten mit einem Datenbestand nutzen.

Document-Modell im RDBMS

Als Letztes stellt sich noch die Frage nach dem Vorteil, wenn ein Datenbank-System sowohl für relationale Datenmodelle als auch für Document-basierte Datenmodelle genutzt werden kann. Hier gibt es zwei Aspekte: Erstens stehen die seit vielen Jahren vorhandenen und weiterentwickelten Fähigkeiten eines RDBMS wie ACID-Transaktionen, Foreign Keys, Methoden zum komplexen Reporting schlagartig auch für den DocumentStore zur Verfügung. Während DocumentStore-Datenbanken mühsam alle vorhandenen Errungenschaften nachbilden müssen, stehen diese Features vollumfänglich in MySQL bereits zur Verfügung. Zusätzlich ergeben sich aus der Kombination von relationalen Daten und JSON-Documents viele Anwendungsfälle, die mit nur einer Technologie nicht einfach zu realisieren wären.

Der zweite Aspekt betrifft den Betrieb des Datenbank-Systems. MySQL bietet ein ausgereiftes und weitentwickeltes Ökosystem von Tools und Erweiterungen, um den Betrieb zu vereinfachen und teilweise auch erst zu ermöglichen. Dies reicht von Online-Backup über diverse Monitoring-Lösungen bis hin zu Security-Erweiterungen wie transparenter Datenverschlüsselung, Auditing, feingranularer Rechteverwaltung und mehr. Alle diese Funktionalitäten stehen automatisch in bewährter Qualität zur Verfügung, um auch den DocumentStore zu betreiben. GDPR-Compliance (DSGVO) kann mit den gleichen Mitteln erreicht werden wie auch für bereits vorhandene relationale Datenbanken. Betrieb und Support erfolgen mit den gleichen Tools und Service-Providern wie bisher. Dies ist eine signifikante Vereinfachung für den Betrieb. Trotzdem erhalten Entwickler alle Möglichkeiten, die neuen Ideen der NoSQL-Datenbanken einzusetzen.

Mario Beck
mario.beck@oracle.com

Carsten Thalheimer
carsten.thalheimer@oracle.com

MySQL HA – Lösungen für Back- und Frontend

Matthias Klein, InnoGames GmbH

Bereits seit dem Jahr 2001 bietet MySQL Replikations-Lösungen an. Während diese auf der Datenbank-Ebene bereits sehr ausgereift sind und stetig verbessert werden, muss man zur Anbindung seiner Anwendung meist noch auf Software von Drittanbietern zurückgreifen. Je nach Anwendungsfall ergeben sich hier unterschiedliche Möglichkeiten und Empfehlungen.

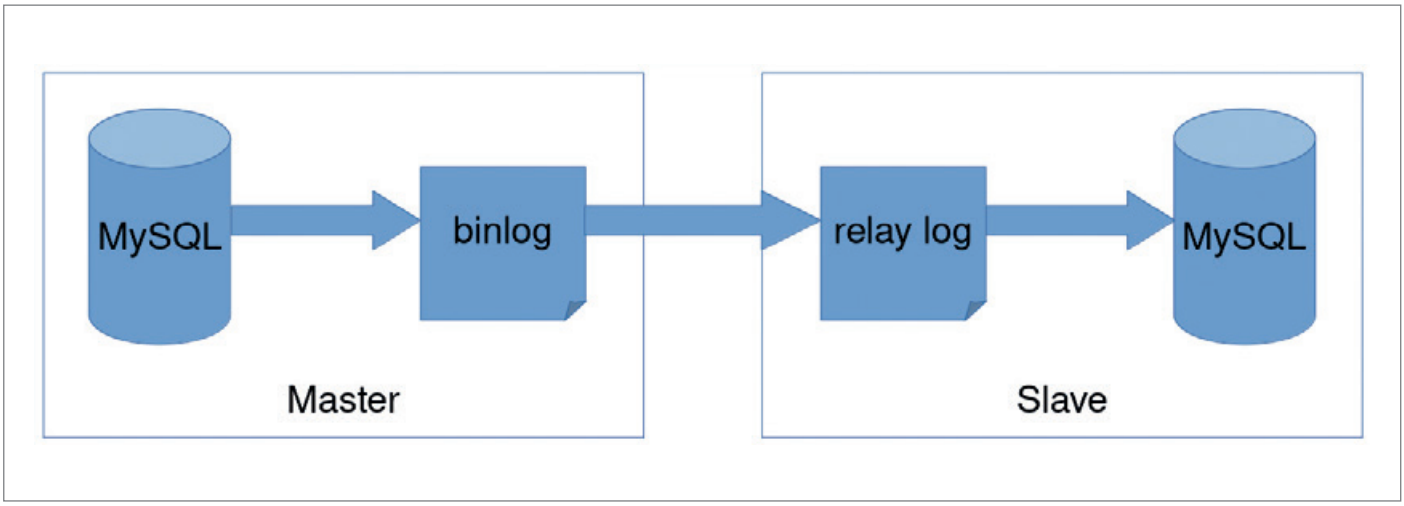


Abbildung 1: Asynchrone Replikation

Schon mit Einführung der Replikation vor etwa siebzehn Jahren konnte man zwischen einer synchronen und einer asynchronen Variante wählen. Während die synchrone Variante mit der „ndb"-Engine sehr spezielle Voraussetzungen bezüglich Hardware und an die eingesetzte Anwendung hat, funktioniert die asynchrone Replikation grundsätzlich mit allen Engines.

In der jüngeren Vergangenheit hat Codership Oy die Galera Cluster entwickelt. Damit stand erstmals eine synchrone Variante der Replikation zur Verfügung, die sich durch Unterstützung der InnoDB-Engine und eine recht einfache Konfiguration auszeichnet. Die Einschränkungen hinsichtlich der zugreifenden Anwendung sind im Gegensatz zu „ndb" nicht mehr existent, auch ist ein Betrieb mit schwächerer Hardware möglich.

Mit MySQL 5.7 führte Oracle ebenfalls eine zusätzliche Cluster-Variante (Group Replication, GRE) ein. Diese integriert sich besser in die bisher bekannten Abläufe, so kann sie zum Beispiel analog zur asynchronen Replikation mit „START GROUP_REPLICATION" gestartet werden. Um aber eine Anwendung an die Datenbank anbinden zu können, kann man (mit Ausnahme von MySQL Fabric) auf eine Software von Drittanbietern zurückgreifen. Die meisten dieser Varianten haben allerdings den Nachteil, dass sie nicht von sich aus hochverfügbar sind. Bei einigen Varianten vermindert das die Ausfallsicherheit nicht wesentlich, bei anderen jedoch enorm. Welche Maßnahmen dagegen ergriffen werden können, ist abschließend aufgezeigt.

Asynchrone Replikation

Diese Art der Replikation zeichnet sich dadurch aus, dass sie unabhängig von der verwendeten Datenbank-Engine arbeitet. Es sind sogar unterschiedliche Engines auf Master und Slave für eine Tabelle möglich. Außerdem besteht die Möglichkeit, Datenbanken und/oder Tabellen von der Replikation auszuschließen, diese nur auf bestimmten Replikaten zu replizieren oder auf dem Replikat einen anderen Namen zu verwenden.

Die asynchrone Replikation verwendet „binlog" auf dem Master und „relaylog" auf dem Slave, um die Änderungen zu übertragen. Dabei wird nach erfolgreichem Anwenden einer Änderung in der Datenbank diese auf dem Master in das „binlog" geschrieben. Der Slave schreibt nun diese Änderungen in sein „relaylog" und wendet sie lokal an.

Dies kann über das ausgeführte SQL (Statement-based) geschehen oder es werden die Zeilen vor und nach der Änderung (Row-based) übertragen. Generell benötigt die Statement-Methode weniger Platz in den Logs, ist aber fehleranfällig, wenn „LIMIT"- oder „GROUP"-Funktionen zum Einsatz kommen. Hier besteht die Möglichkeit, mittels „MIXED" derartige Statements Row-based zu replizieren. Die Row-based Replikation benötigt zwar mehr Platz in den Logs, jedoch muss der Slave das SQL nicht mehr ausführen und kann die Änderungen direkt anwenden *(siehe Abbildung 1)*.

Aus dieser Art der Replikation ergeben sich hauptsächlich Probleme mit der Anwendung der Änderungen auf dem Slave. So kann es hier zu Verzögerungen kommen, da der Slave nur einen Thread zum Anwenden der Änderungen benutzen kann, der Master jedoch mit mehreren gleichzeitig Änderungen durchführt. Hier kann man sich zwar mit der Option „slave_parallel_workers" behelfen, jedoch nutzt das nur dann, wenn mehrere Datenbanken repliziert werden.

Auch ist es per Default möglich, auf einen Slave zu schreiben. Dies kann dazu führen, dass der Slave die Replikation unterbricht und der Administrator entscheiden muss, welcher Datenstand der richtige ist. Allerdings kann dies auch unbemerkt passieren, wenn dieser Datensatz nicht wieder angefasst wird. Um dies zu verhindern, sollten alle Slaves „read_only" (ab Version 5.7 auch „super_read_only") aktiviert haben.

Synchrone Replikation

Obwohl die „ndb"-Engine die älteste Form der synchronen Replikation von MySQL ist, führt sie eher ein Nischendasein. Dies liegt vor allem an den speziellen Anforderungen, die sie an Hardware und Anwendung stellt, um noch performant zu sein. Die Daten sind über mindestens zwei Knoten verteilt, wovon einer immer eine Kopie der Daten hält.

Die „ndb"-Engine partitioniert die Daten automatisch auf die vorhandenen Knoten. Zusätzlich ist noch mindestens ein System für das Management erforderlich sowie mindestens ein SQL-Knoten, der die Anfragen der Anwendung entgegennimmt. Konnten die Daten ur-

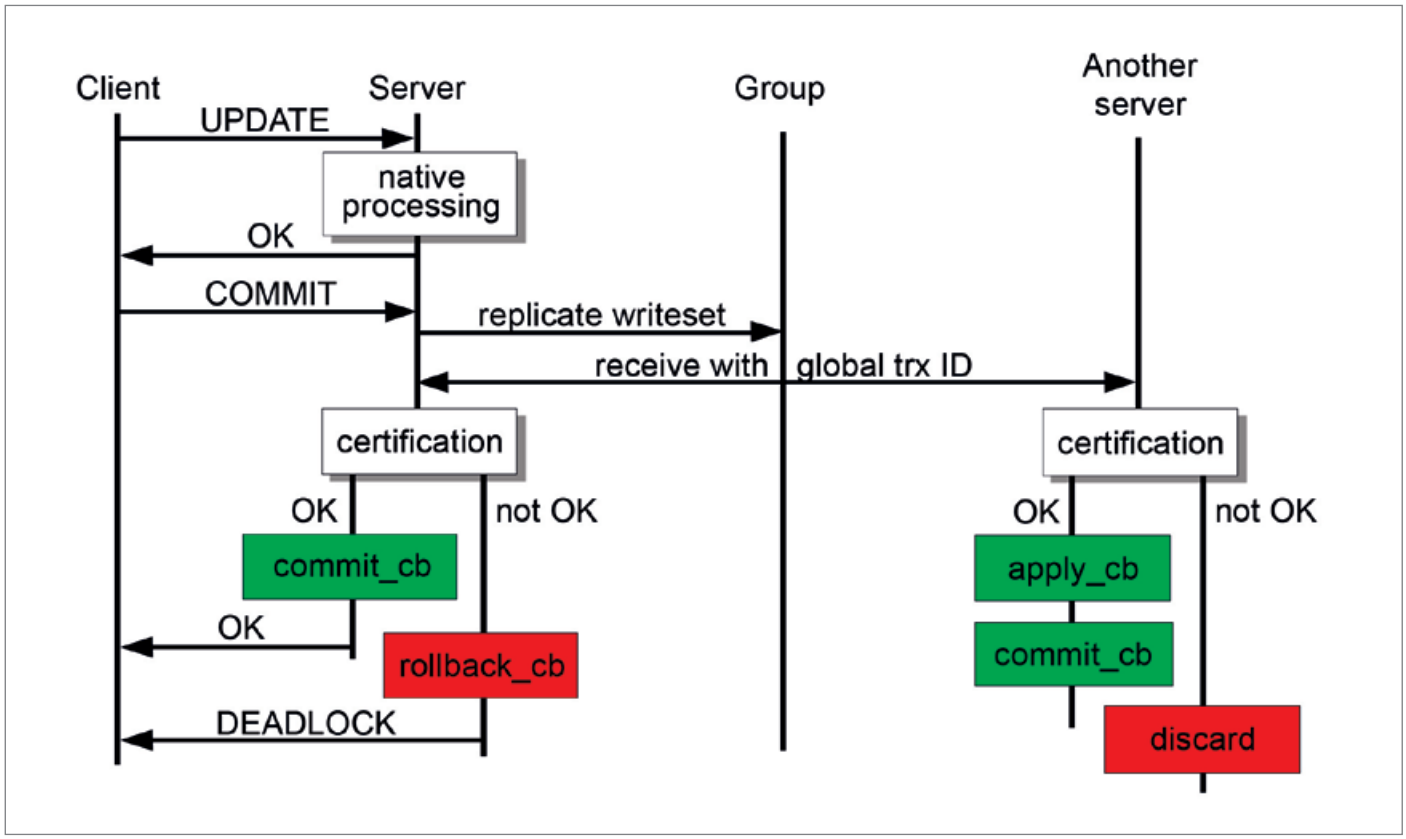


Abbildung 2: Schreib-Prozess bei Galera Cluster

sprünglich ausschließlich im RAM der Datenknoten abgelegt werden, ist das mittlerweile auf Kosten der Performance auch auf Festplatte möglich.

Bei der „ndb"-Engine bestehen meist Performance-Probleme. Gerade bei vielen Datenknoten sind die Daten stark partitioniert, was zum Beispiel Joins extrem teuer macht. Diese Operationen werden von den Datenknoten an den SQL-Knoten hochgereicht, der dann das endgültige Ergebnis noch berechnen muss. Dies lässt sich nur durch entsprechende Anpassung des Schemas und der Anwendung verhindern. Ebenfalls problematisch sind Schema-Änderungen, die erst seit der „ndb"-Version 7.5 online erfolgen können. Verglichen mit InnoDB dauern diese auch deutlich länger.

Seit etwa fünf Jahren gibt es mit Galera Cluster eine stabile synchrone Replikation, die auf InnoDB basiert. Oracle hat mit MySQL 5.7 die Group Replication (GRE) eingeführt. Beide Methoden unterscheiden sich in den technischen Details, die Konzepte sind jedoch so ähnlich, dass beide zusammen betrachtet werden können.

Die Anwendung der Änderung im Cluster erfolgt in einem mehrstufigen Prozess. Dabei werden die Änderungen erst lokal auf dem Server verarbeitet. Sobald der Commit erfolgt, findet auf den anderen Servern ein Zertifizierungsprozess statt. Dieser prüft, ob die Änderungen anwendbar sind, und sendet das Ergebnis zurück. Da hier tatsächlich keine Daten geschrieben werden, ist er sehr schnell.

Lassen sich die Änderungen auf allen Servern durchführen, werden dem Client der Commit bestätigt und die Daten endgültig geschrieben, im Fehlerfall die Daten nicht geschrieben und der Client bekommt eine entsprechende Meldung. Hier zeigt sich der Unterschied zwischen Galera und GRE deutlich: Während Galera nur ein generisches DEADLOCK zurückliefert, konnte Oracle entsprechende Fehlercodes implementieren *(siehe Abbildung 2)*.

Ob sich am Ende Galera oder GRE durchsetzen oder beide eine Koexistenz führen werden, ist im Moment noch unklar. Galera profitiert durch den früheren Start. So konnte Codership bereits mehr Erfahrungen sammeln und seine Implementierung weiter verbreiten. MySQL punktet durch die direkte Implementierung im Code, was sich durch eine angenehmere Bedienung über „mysql cli" zeigt. Es ist davon auszugehen, dass Oracle fehlende Features in naher Zukunft nachrüsten wird. So kann man bei Galera mit einem Arbitrator Ressourcen sparen. Auch gibt es hier Möglichkeiten, die Kommunikation des Clusters über WAN-Verbindungen zu beschleunigen. Allerdings ist bei GRE die Kommunikation des Clusters untereinander deutlich Ressourcen-schonender und wird mit mehr Servern auch nicht spürbar langsamer.

Synchrone Replikation sollte immer mit einem dedizierten Server für Schreibzugriffe betrieben werden. Obwohl die Cluster-Software das zeitgleiche Schreiben auf unterschiedlichen Cluster-Nodes unterstützt, bringt dies am Ende keine Vorteile. Da die Daten auf jedem Node geschrieben werden müssen, werden Schreib-Operationen nicht schneller. Je mehr gleichzeitig auf unterschiedliche Nodes geschrieben wird, desto höher ist die Chance, dass der Zertifizierungsprozess fehlschlägt. Auch sollte darauf geachtet werden, dass die zu schreibenden Änderungen möglichst klein sind, da sonst der Zertifizierungsprozess und die Anwendung der Änderung sehr langsam werden, was unter Umständen den kompletten Cluster blockieren kann. Die bei Galera empfohlenen Grenzen liegen hier

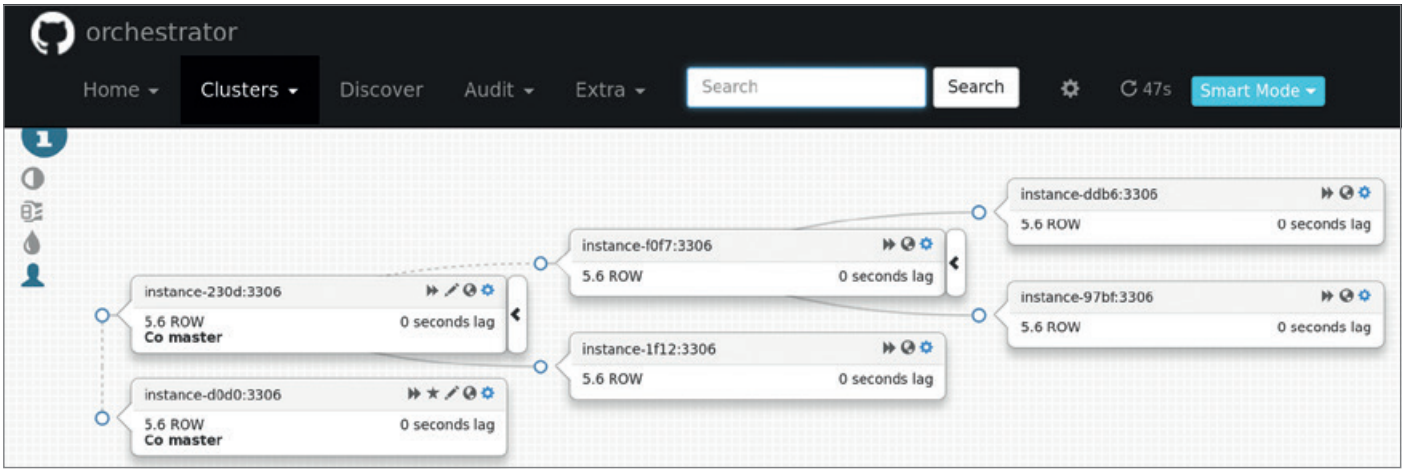


Abbildung 3: Orchestrator-Web-Oberfläche

bei 128 KB pro Zeile beziehungsweise 1 GB pro Transaktion.

Ein wichtiger Faktor beim Einsatz von Cluster-Systemen ist die Hardware. Der gesamte Cluster ist nur so schnell wie sein schwächstes Mitglied. Es wird auch eine ungerade Anzahl von Nodes (mindestens drei) empfohlen, um Split-Brain-Szenarien vorzubeugen. Dabei handelt es sich um Situationen, in denen die eine Hälfte der Nodes nicht mehr mit der anderen kommunizieren kann. Dabei wird der Cluster funktionsunfähig, da die Nodes keine Mehrheit für die Masterwahl mehr herstellen können.

Failover mittels MySQL Fabric

Die Hochverfügbarkeit auf der Datenbank-Ebene ist einfach herzustellen und arbeitet stabil. Um jedoch die Anwendung anzubinden, ist in den meisten Fällen Drittanbieter-Software oder eigene Programmierarbeit notwendig.

Oracle selbst bietet mit MySQL Fabric eine eigene Lösung an. Diese erfordert allerdings einen speziellen Treiber für die Anwendung und ist nur für asynchrone Replikation geeignet. Auch der Management-Server, der die Koordination übernimmt, ist selbst nicht hochverfügbar. Dessen Ausfall würde den alten Status beibehalten. Ein zusätzlicher Ausfall des Masters würde zu einer Downtime führen, was jedoch recht unwahrscheinlich ist.

Neben dem Routing kümmert sich Fabric auch um die Wiederherstellung der Replikation. So wird im Fehlerfall aus einem Pool von Servern ein neuer Master bestimmt und alle verbliebenen Server als dessen Slave konfiguriert. Unschön ist an dieser Stelle, dass der ausgefallene Server nach seiner Wiederherstellung manuell in Fabric wieder aktiviert werden muss. Damit ist man zwar auf der sicheren Seite, wenn man auf diesem Server Daten wiederherstellen musste; wenn jedoch nur ein Neustart etwa aufgrund von Softwareaktualisierungen stattgefunden hat, ist das Ganze recht umständlich.

Failover mittels MHA oder Orchestrator

Einen ähnlichen Weg gehen Master High Availability Manager (MHA) und Orchestrator. Beide Tools können asynchrone Topologien managen, also im Fehlerfall einen neuen Master bestimmen und die Slaves entsprechend konfigurieren. Beide benötigen eine Management-Instanz. Nicht per Default integriert ist allerdings, die Anwendung über die Topologie-Änderung zu informieren (wie bei Fabric) beziehungsweise auf eine Art den Betrieb sicherzustellen. Dies liegt an den verschiedenen Möglichkeiten, mit Fehlersituationen umzugehen. So kann man beispielsweise die Anwendung per API über die Änderung informieren, sofern sie dies unterstützt, oder einfach deren Konfiguration austauschen. Falls ein Proxy benutzt wird, kann dieser die Änderung abfragen.

Die am häufigsten verwendete Methode ist jedoch, den Wechsel mithilfe einer virtuellen IP zu vollziehen. Dabei ist die IP-Adresse, die die Anwendung zur Verbindung nutzt, nicht fest an einen Datenbank-Server gebunden, sondern wird je nach Bedarf und Topologie zusätzlich vergeben.

MHA und Orchestrator haben bei unterschiedlichen Ereignissen (wie Master up/down) Hooks integriert, über die man dann passende Skripte ausführen kann. Dies überlässt dem Administrator die volle Kontrolle über die zu treffenden Maßnahmen; so kann hier beispielsweise auch der ausgefallene Master nach der Wiederherstellung automatisch als Slave weiterlaufen. Dazu liefern beide Tools Beispiel-Skripte mit, die schnell an die eigenen Anforderungen angepasst sind.

Wie auch bei Fabric führt der Ausfall der Management-Instanz nur dazu, dass im Fehlerfall kein automatischer Failover stattfinden kann.

Während MHA einzig auf der Kommandozeile zu bedienen ist, bietet Orchestrator ein Web-Interface, über das man einen Überblick über die aktuelle Situation erhalten kann. Auch Änderungen der Topologie sind mit „Drag&Drop“ möglich, zudem lassen sich Server-Parameter einsehen und ändern. Zusätzlich sind diese Funktionen auch über ein Web-API zugänglich *(siehe Abbildung 3)*.

Als einziges vorgestelltes Tool kann Orchestrator von sich aus Hochverfügbarkeit herstellen. Dazu genügt es, mehrere Instanzen zu installieren und ein gemeinsames Backend zu nutzen.

Failover mittels Proxy

Eine weitere Möglichkeit, eine Anwendung an die Datenbanken anzubinden, besteht darin, einen Proxy zu verwenden.

Dabei wird in der Anwendung der Proxy als Ziel konfiguriert und dieser kümmert sich um die Weiterleitung der Verbindung. Hier bieten sich zum Beispiel „haproxy“ oder ProxySQL an. Allerdings ist bei einer Proxy-Lösung auch dafür zu sorgen, dass diese selbst hochverfügbar ist. Weitverbreitet ist die Nutzung von „keepalived“ für diesen Zweck, jedoch ist auch die Kombination aus „heartbeat“ und „pacemaker“ möglich. Entsprechende Anleitungen sind in der Dokumentation der jeweiligen Proxy-Software verfügbar. Mit Proxy lassen sich auch synchron replizierende Systeme anbinden.

Haproxy ist ein generischer Proxy, der für alle Dienste verwendet werden kann. In der einfachsten Variante wird lediglich überprüft, ob die Datenbank erreichbar ist. Dies ist allerdings in Replikations-Szenarien nicht ausreichend, da auch überprüft werden muss, ob der Server richtig repliziert und dessen Daten genügend aktuell sind. Realisieren kann man das mit eigenen Skripten, die von „haproxy“ via „xinetd“ gestartet werden.

Die bessere Wahl ist ProxySQL. Wichtig ist dabei, dass ProxySQL nicht allein arbeiten kann. Standardmäßig wird überprüft, welche Datenbank in der Topologie schreibbar ist. Auf diese werden dann die Schreib-Operationen geleitet, die anderen werden zum Lesen genutzt. Dies funktioniert sehr gut, wenn GRE benutzt wird. Standardmäßig wird nur ein Node des Clusters zum Schreiben freigegeben und die Cluster-Software kümmert sich selbst um die Topologie. Bei einer asynchronen Replikation gibt es eine solche Funktionalität jedoch nicht. Hier helfen MHA oder Orchestrator, die nach einem Failover die entsprechenden Einstellungen vornehmen. Zusätzlich bietet ProxySQL Query Routing und Firewall. So können Anfragen auf den für sie besten Server umgeleitet oder problematische Anfragen unterdrückt werden.

Fazit

Welches die richtige Replikationsmethode ist, hängt von der jeweiligen Anwendung ab. Grundsätzlich ist aufgrund der Robustheit eine Cluster-Lösung zu empfehlen. Hier empfiehlt sich die Group Replication, da sie sich besser in das bestehende System integriert. Auch dass hier Hosts automatisch auf „read_only“ gesetzt werden, hilft bei der Anbindung von Anwendungen mittels ProxySQL.

Wer über mehrere Rechenzentren ein Cluster bilden muss, sollte momentan noch zu Galera greifen. Hier überwiegen die Vorteile der von Codership implementierten WAN-Optimierungen deutlich. Man kann aber hier auch überlegen, ob man in den Rechenzentren GRE nutzt und diese untereinander durch asynchrone Replikation anbindet. Die asynchrone Replikation bietet sich an, wenn man wenig Hardware zur Verfügung hat oder große Datensätze schreiben muss.

Welches Tool zum Anbinden der Anwendung genutzt wird, richtet sich hauptsächlich nach der Replikationsmethode. Bei synchroner Replikation empfiehlt sich ProxySQL. Bei asynchroner Replikation bevorzugt der Autor aufgrund der besseren Übersicht im Web-Interface Orchestrator.

Matthias Klein
matthias.klein@innogames.com

Wenn Daten historisch wachsen

Antoniya Kuhlmeyer, Awin AG

MySQL gehört zu den am meisten verbreiteten Open-Source-Datenbank-Systemen. Es lässt sich mit wenigen Handgriffen installieren und auch die Replikation ist sehr einfach aufzusetzen. Mit stets wachsender Datenmenge kann allerdings ein kleines Missgeschick im Design oder in der Konfiguration eines MySQL-Servers zu einem späteren Zeitpunkt eine echte Herausforderung werden.

Wer mit relationalen Datenbanken und SQL vertraut ist, erhofft sich einen schnellen Einstieg in die MySQL-Technologie. Kommt man allerdings ungeplant und unvorbereitet zu deren produktiven Einsatz, läuft man Gefahr, auf einige Fallen zu stoßen. Der Artikel zeigt gängige Fallen im Design und Betrieb von MySQL-Datenbanken auf. Im Mittelpunkt stehen praktische Erfahrungen rund um Datenbank-Design, Anwendung verschiedener Storage Engines und deren Auswirkung auf Performance und Betrieb, High Availability und Replikation.

MySQL und relationale Datenbank-Systeme

Die Grundlagen für relationale Datenbanken wurden bereits in den 1960er-Jahren gelegt. Heutzutage gelten sie als eine bewährte Technik, um Daten zu organisieren, und bilden eine unentbehrliche und zentrale Komponente vieler IT-Systeme.

Auch wenn MySQL in die Kategorie der relationalen Datenbanksysteme fällt, bedeutet das nicht zwingend, dass MySQL die ACID-Eigenschaften erfüllt. Das wohl wichtigste und ungewöhnlichste Merkmal von MySQL ist seine Engine-Architektur, die Abfrage-Verarbeitung von der eigentlichen Datenspeicherung trennt. Die MySQL Storage Engines unterscheiden sich wesentlich in der Art, wie sie Daten intern ablegen. Die Wahl der Storage Engine bestimmt somit, ob zum Beispiel Fremdschlüssel, Transaktionen und welche Art von Indizes unterstützt werden.

Das Produktions-System

Die nachfolgend aufgeführten Szenarien basieren auf einem Legacy-Datenbank-System mit zwei Mastern in MySQL Version 5.5 und einer Datengröße von jeweils zehn Terabyte. Darüber hinaus verfügt das System über einen Mix von rund zwanzig Slaves, die unterschiedlichste Replikationsfilter einsetzen und in MySQL 5.5 oder 5.6 laufen. Die Replikation vom Master zu den Slaves erfolgt über Statement-basierte Logs, es werden also nur die SQL-Statements in die Logs geschrieben und diese dann lokal auf dem jeweiligen Server ausgeführt. Der Datenbank-Betrieb muss rund um die Uhr sichergestellt sein und jegliche Wartungsarbeiten, die den Betrieb unterbrechen, sind auf minimaler Dauer zu halten.

Die richtige Wahl der Storage Engine

Die Struktur der Daten in einer relationalen Datenbank spielt später eine entscheidende Rolle für Performance und Betrieb. Genauso wichtig ist eine möglichst präzise Kapazitätsplanung, basierend auf dem erwarteten Datenvolumen. Der Entwicklungsprozess von relationalen Datenbanken folgt dem Wasserfall-Modell und aus diesem Grund ist es umso wichtiger, dass man grobe Fehler von Beginn an ausschließt. Spätere Änderungen in der Struktur der Daten gestalten sich oft als schwierig, wenn die Datenbank bereits im produktiven Einsatz ist.

Im ununterbrochen laufenden Betrieb größerer Datenbank-Systeme ist es oft sogar unmöglich, gewisse Optimierungen im Nachhinein durchzuführen, ohne die Verfügbarkeit der Datenbank zu unterbrechen. In Umgebungen mit sehr schnellem Datenwachstum beobachtet man immer häufiger auftretende Performance-Probleme und auch die Wiederherstellung der Daten nach einem Ausfall beansprucht mehr Zeit, in der die Datenbank unter Umständen gar nicht verfügbar ist.

Diese Bedenken betreffen nicht nur MySQL-Datenbanken; die Handhabung einer MySQL-Datenbank ist allerdings zum größten Teil davon bestimmt, welche Storage Engine im Einsatz ist. Im folgenden Abschnitt werden reelle Anwendungsfälle für verschiedene MySQL Storage Engines vorgestellt und auf ihre Vor- und Nachteile im Live-Betrieb untersucht.

InnoDB

Seit Version 5.5.5 ist InnoDB die Default Storage Engine in MySQL. Sie ist konform mit den ACID-Eigenschaften, da es intern ein Transaktionsprotokoll gibt, um die Datenkonsistenz auch nach einem Absturz des Datenbank-Servers zu erhalten. InnoDB implementiert Fremdschlüssel und ist die einzige Engine, die Clustered Indizes unterstützt. Darüber hinaus sperrt InnoDB bei Datenzugriffen einzelne Zeilen und nicht die komplette Tabelle, was eine höhere Anzahl von gleichzeitigen Zugriffen auf derselben Tabelle erlaubt. Aus diesem Grund können allerdings auch Deadlocks auftreten, die InnoDB in den meisten Fällen automatisch erkennen und auflösen kann. Aufgrund seiner Robustheit sollte InnoDB in einem produktiven OLTP-System die erste Wahl sein, dennoch gibt es Szenarien, in denen die Performance ausbleibt.

Sehr große Tabellen: Im System der Autorin gab es eine besonders große InnoDB-Tabelle. Die Daten wurden nur eingefügt, aber nie verändert. Über die Zeit wuchs die Tabelle auf 1,9 TB und aufgrund fehlender Indizes war sie nur für einige wenige Anwendungsfälle sinnvoll abfragbar. Genauere Untersuchung zeigte, dass Applikationen nur Daten der letzten 24 bis 48 Stunden abfragten und somit ältere Daten potenziell entfernt werden konnten. Die Tabelle von alten Daten zu befreien, gestaltete sich dennoch schwieriger als erwartet, denn es war weder ein inkrementeller Primärschlüssel noch ein passender Index vorhanden. Die Erstellung eines solchen Index hätte die Tabelle für unbestimmte Zeit gesperrt und war somit auch keine Option im Produktionssystem.

Die Lösung: Hat man Prozesse, die auf Daten der letzten x Stunden/Tage zugreifen, ist es sinnvoll, über Partitionierung nachzudenken. Im Beispiel wurde eine zweite Tabelle erstellt und per MySQL-Replikation befüllt. Diese wurde in „RANGE"-Partitionen auf einer Datetime-Spalte organisiert. Lese-Operationen haben somit effektiv nur 1 bis 2 Tagespartitionen abgefragt statt der kompletten Datenmenge von 1,9 TB über einen nicht Zeitstempel-basierten Index. In Zahlen ausgedrückt bedeutete dies, dass der Index von 800 GB nicht mehr komplett durchsucht wurde, sondern lediglich maximal zwei Partitionen mit jeweiliger Größe von 2 bis 3 GB. Möchte man die Tabelle auf demselben Datenbankserver behalten, kann man alternativ mit einem Insert-Trigger arbeiten, der die neue Tabelle parallel zu der aktuellen inkrementell befüllt. Ist die neue Tabelle ausreichend befüllt, kann man die Replikation / den Trigger entfernen und die alte Tabelle außer Betrieb nehmen. Ein zusätzlicher Vorteil der Partitionierung ist, dass

man ältere Partitionen quasi ausschalten kann, ohne die Daten kopieren zu müssen. Es empfiehlt sich, einen Prozess einzuführen, der die Tages-Partitionen automatisiert im Voraus erstellt und abgelaufene dauerhaft entfernt, um somit unkontrolliertes Datenwachstum zu vermeiden.

Kleine Tabelle, viele Schreibzugriffe: Auch bei viel kleineren Datenmengen können bereits ein schlechtes Datenschema oder Applikationsdesign zu Problemen führen. Ein Beispiel dafür ist eine Tabelle, die als Zwischenablage für Daten eingesetzt wird und auf die viele Prozesse schreibend zugreifen. Es gibt dabei im Wesentlichen zwei Arten von Prozessen – die einen fügen Daten in die Tabelle ein, die anderen lesen diese, verarbeiten sie auf Applikationsebene und löschen sie anschließend aus der Tabelle. Die Tabelle verfügte nur über nicht eindeutige Indizes, um die Leseoperationen zu optimieren. Durch steigende Anzahl der zugreifenden Prozesse und die fein-granularen InnoDB-Sperren kommt es häufig zu Deadlocks, die InnoDB automatisch auflöst, indem es eine der konkurrierenden Transaktionen zurücksetzt.

Die Lösung: In einem OLTP-Datenbanksystem sollte man nicht auf geeignete Primärschlüssel verzichten. Verfügt die Tabelle über einen Clustered Index, der einen nummerischen Datentyp verwendet und inkrementell wächst, können konkurrierende Insert- und Delete-Operationen geschickt verteilt werden, sodass keine Deadlock-Situationen entstehen. Wenn die Prozesslogik es zulässt, kann man für Leseoperationen zusätzlich Gebrauch von geeigneten Transaction Isolation Levels machen, die steuern, welche Datenänderungen ab wann für parallel laufende Prozesse sichtbar sind. Generell sind Deadlocks auf Applikationsebene zu lösen, da sie durch die zugreifenden Prozesse ausgelöst werden.

MyISAM

Vor Version 5.5.5 war MyISAM die Standard Engine für Tabellen in MySQL. Im Gegensatz zu InnoDB wird für MyISAM-Tabellen kein Transaktions-Protokoll geführt, was letztendlich bedeutet, dass die Storage Engine nicht ACID-konform ist. Das macht MyISAM sehr anfällig für Daten-Korruptionen, wenn der Datenbank-Server unerwartet stoppt, etwa bei einem Absturz. Ist der Server wieder online, muss die Tabelle von halbgeschriebenen Daten bereinigt werden.

MyISAM unterstützt weder Fremdschlüssel noch Transaktionen. Um parallele Zugriffe auf dasselbe Datenset zu koordinieren, werden nur Sperren auf Tabellen-Ebene benutzt. Diese Faktoren schränken die parallele Abarbeitung von Abfragen enorm ein, wenn Schreibzugriffe involviert sind.

MyISAM-Daten sind in mehreren Files organisiert, die sich bei gestopptem MySQL-Prozess ohne Weiteres vom Filesystem kopieren lassen. Das kann vor allem beim Wiederherstellen von Daten oder bei der Migration von ganzen Datenbanken sehr hilfreich sein. Vor MySQL 5.6 war MyISAM die einzige Storage Engine, die Full-Text-Indizes unterstützte.

MyISAM-Tabellen in der Praxis: In einer produktiven Datenbank hat man selten gute Gründe, um MyISAM als Default Storage Engine einzusetzen. Wer große Datenmengen in MyISAM lagert, kann mit langen Wartezeiten und potenziellem Datenverlust rechnen, wenn diese Tabellen nach Absturz repariert werden müssen.

Logische Backups von MyISAM kollidieren mit Schreib-Operationen und können somit die schreibenden Zugriffe enorm verzögern. Ist der Datenbank-Server, auf dem die MyISAM-Korruption entstanden ist, zusätzlich als Replikat konfiguriert, kann die Tabellensperre während der Repair-Operation unter Umständen die Replikation blockieren. Eine MyISAM-Tabelle von ca. 1 TB benötigte in der genannten Umgebung dafür 12 bis 16 Stunden. Währenddessen war es unmöglich, Daten vom Master zu replizieren, was insgesamt mehr als 24 Stunden beanspruchte, bis alle Tabellen auf dem Slave-Server wieder verfügbar und synchronisiert waren.

Die Lösung: Für Tabellen ab einer Größe von mehreren Gigabyte ist es dringend empfohlen, diese in InnoDB zu konvertieren. Logische Backups von InnoDB-Tabellen blockieren Schreib-Operationen nicht und man kann durch einen bewussten Einsatz von Isolation-Level die Anzahl der parallelen Transaktionen erhöhen. Handelt es sich um größere Tabellen, kann man zusätzliche Tools wie Percona „pt-online-schema-change“ einsetzen, um eine nicht blockierende Konvertierung während des normalen Betriebs zu ermöglichen. Dabei wird intern eine neue Tabelle parallel aufgebaut und per Trigger aktuell gehalten, während Clients weiterhin auf die bestehende Tabelle zugreifen können. Mit diesem Ansatz konnte die Autorin eine MyISAM-Tabelle von 180 GB in Produktion in InnoDB konvertieren. Wichtig zu beachten ist, dass das Tool keine Änderung auf einer Tabelle vornehmen wird, wenn diese keinen Primärschlüssel oder zumindest keinen eindeutigen Schlüssel aufweist.

Merge MyISAM

In MySQL ist es möglich, eine Ansammlung von identischen Tabellen unter deren Beibehaltung zu einer logischen Tabelle zusammenzufassen. In diesem Fall spricht man von einer „Merge-Tabelle“. Dieses Konzept kann nur für MyISAM-Tabellen verwendet werden, die in Bezug auf Reihenfolge und Datentypen der jeweiligen Spalten identisch sind. Vorteile bringt das meist für die Clients, die auf solch eine Tabelle zugreifen, da sie nicht genau wissen müssen, in welcher der zugrunde liegenden Tabellen sich die gefragten Daten befinden.

Merge-Tabellen in der Praxis: Merge-Tabellen basieren auf MyISAM, daher bringen sie alle deren Vor- und Nachteile mit. Aus operativer Sicht ist der Einsatz dieser Engine jedoch nur bedingt sinnvoll.

Praktische Erfahrungen mit einer Merge-Tabelle von über 100 GB und mehr als 220 zugrunde liegenden Tabellen zeigen, dass vor allem auf Replikaten Konflikte zwischen Schreib- und Lese-Operationen entstehen, die den Replikationsprozess blockieren.

Die Lösung: MySQL unterstützt partitionierte Tabellen nativ seit Version 5.1. – ein Feature, das effizientere Abfragemöglichkeiten bietet und flexibler bei der Storage-Engine-Wahl ist. Ein weniger invasiver Ansatz, eine Merge-Tabelle zu ersetzen, wäre, eine View zu definieren, die die MyISAM-Tabellen vereint. In einer MySQL-5.5-Umgebung erwies sich das aufgrund der eher großen Datenmenge als unpraktikable Lösung.

Weitere Storage Engines in Einsatz

MySQL bietet einige weitere Engines, die für speziellere Anwendungsfälle geeignet sind:

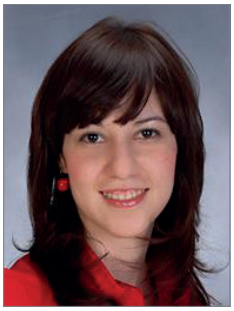
- *Archive*
  Diese Engine eignet sich, wie der Name auch schon sagt, für archivierte Daten. Spalten von Archive-Tabellen können nicht indiziert werden, was aber die hoch komprimierte und somit speichereffiziente Lagerung der Daten unterstützt.

- *Federated*
  Federated-Tabellen ermöglichen den Zugriff auf Daten, die auf einer anderen MySQL-Instanz oder entfernten Servern liegen. Für Clients ist der tatsächliche Speicherort der angefragten Daten transparent. In der Praxis ist die Anwendung dieser Engine nicht empfehlenswert, da selbst bei einer kleinen Anzahl von ausgegebenen Zeilen die Ausführungszeit um mehr als das Hundertfache steigt. In manchen Fällen führen Abfragen auf Federated-Tabellen zum Absturz der MySQL-Instanz, auf der die Clients die Abfragen absetzen.

- *Blackhole*
  Diese Engine eignet sich für komplexere MySQL-Replikations-Topologien, in denen ein oder mehrere Slaves selbst als Master agieren, um weitere Slaves mit Daten zu befüllen. Die Besonderheit von Blackhole-Tabellen ist, dass sie Schreib- und Lese-Zugriffe akzeptieren, aber Datenänderungen nicht persistieren. Auf diese Weise werden lediglich die MySQL-Binary-Logs befüllt, die bei der Replikation zu anderen Slaves weitergereicht werden. Der Vorteil bei diesem Ansatz ist, dass der Zwischen-Master selbst keine Verzögerung in der Replikationskette verursacht, da die Schreib-Operationen nur geloggt, aber nicht auf reellen Daten ausgeführt werden. Voraussetzung für solche Setups ist, dass der Zwischen-Master das Statement-Format für seine Binary-Logs benutzt. Potenzielle Probleme können auftreten, wenn versucht wird, persistierte Tabellen zu erstellen, die auf eine Blackhole-Tabelle referenzieren.

Fazit

MySQL bietet eine Vielzahl von Möglichkeiten und Features für die Speicherung von Daten. Die Wahl der Storage Engine beeinflusst nicht nur die interne Struktur der zu persistierenden Daten, sie bestimmt auch wesentliche Aspekte der Abfrage-Performance, der Backup-Strategien und der Replikations-Konfiguration. Mit steigender Datengröße ist es nicht immer einfach, bestehende Datenstrukturen zu verändern und zu verbessern. Der Artikel hat anhand einiger Beispiele aus der Praxis gezeigt, wie man Storage Engines kombinieren kann, um eine bessere Performance zu erzielen oder weniger Speicherplatz zu belegen.



Antoniya Kuhlmeyer
antoniya.kuhlmeyer@awin.com

Keine Angst vor Key-Value-Stores: Einblicke in die Oracle NoSQL DB

Karin Patenge, ORACLE Deutschland B.V. & Co. KG

Das Portfolio an Daten-Management-Systemen von Oracle bietet schon seit langer Zeit mehr Lösungen als nur die Oracle-Datenbank. Mit MySQL, der Berkeley DB, TimesTen, Oracle Rdb, Oracle Essbase oder der Oracle NoSQL DB werden nicht nur die wichtigsten Plattformen unterstützt, sondern auch die unterschiedlichsten Anforderungen für die Datenhaltung in unternehmenskritischen Anwendungen.

Die Oracle NoSQL DB rückt immer dann in den Fokus der Betrachtung, wenn es um Anwendungen geht, die insbesondere Flexibilität bei der Schema-Definition erwarten („Schema-less"), die höchste Verfügbarkeit sowie Performance versprechen und die flexibel und kostengünstig je nach auftretenden oder erwarteten Lasten skaliert werden können.

Anforderungen aus dem Bereich relationaler Datenbanksysteme wie die ACID-Konformität (Atomicity, Consistency, Isolation and Durability) für Transaktionen treten in den Hintergrund zugunsten einer „Eventual Consistency". Diese besagt, dass konsistente Zustände unter Inkaufnahme vorübergehender inkonsistenter Zustände erreicht werden. Die Oracle NoSQL DB schafft dabei den Spagat, auf programmatischem Weg feingranular Konsistenz in der Bandbreite von BASE bis ACID zu ermöglichen. Neben dieser Eigenschaft ist für die meisten eine voll funktionsfähige, kostenfreie Community-Edition Ausgangspunkt für die Evaluierung oder Nutzung dieser Datenbank.

Die wichtigsten Informationen im Überblick

Die Oracle NoSQL DB gehört zur Gruppe der horizontal verteilten und skalierbaren

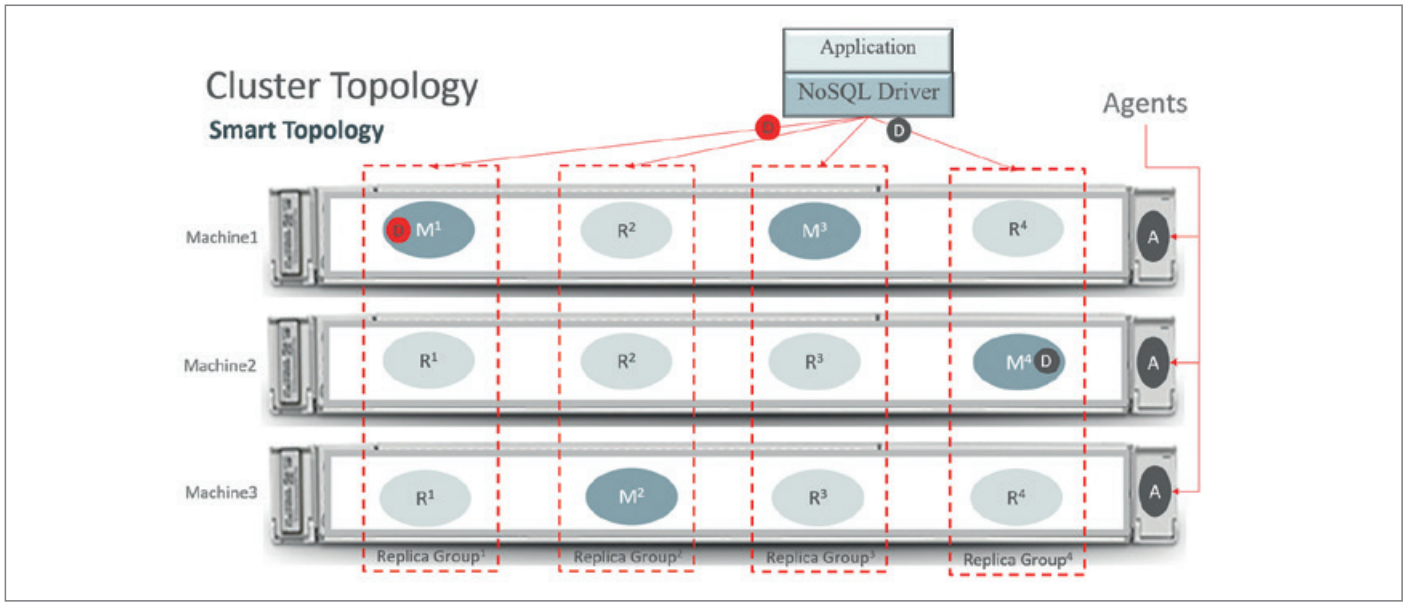


Abbildung 1: Oracle NoSQL DB – Cluster-Topologie

„Key-Value-Stores“. Diese stellen das derzeit einfachste Datenmodell bereit. Datensätze sind Schlüssel-Werte-Paare. Die Menge aller Paare wird in der Datenbank (im „Store“) gespeichert. Sowohl Schlüssel als auch Werte können beliebigen Inhaltes sein.

Schlüssel, wie die Verwendung des Begriffs schon nahelegt, dienen dazu, effizient auf den zugeordneten Wert zuzugreifen. Als Werte können auch JSON-Dokumente gespeichert sein. Die Oracle NoSQL DB unterstützt nativ JSON. Wird die Datenbank um die Software-Komponente „Oracle Big Data Spatial and Graph“ erweitert, ist sie als Graph-Datenbank nutzbar.

Horizontal verteilte Datenbank bedeutet, dass die eingehenden Datensätze verteilt (typischerweise über eine Hash-Funktion, die auf den Schlüssel angewendet wird) auf verschiedenen Speicherknoten geschrieben werden. Es liegt also eine „Shared Nothing“-Architektur zugrunde. Schreibvorgänge sind umso performanter, je höher der Grad der Parallelisierung ist. Letzterer wiederum wird durch die Anzahl der Speicherknoten bestimmt.

Skalierbarkeit wird erreicht, indem bei Bedarf online zusätzliche Speicherknoten zum bestehenden Datenbank-Cluster hinzugefügt werden. Wird auf diese Weise die Topologie der Datenbank verändert, erfolgt ein automatisches Ausbalancieren beziehungsweise Umverteilen der Daten, um so den oder die neuen Clusterknoten einzubeziehen.

Die Ausfallsicherheit beziehungsweise die höchste Verfügbarkeit wird vereinfacht gesagt dadurch gewährleistet, dass jeder Datensatz nicht nur einmal, sondern n-fach im Datenbank-Cluster abgelegt ist. Die Replikate liegen dabei auf unterschiedlichen Speicherknoten. Der Grad der Replikation ist frei wählbar. Es empfiehlt sich, mit einem Replikationsfaktor von mindestens drei zu arbeiten, also ein Masterknoten und zwei Replikaknoten. Welche Strategie beim Schreiben und Lesen der Daten gewählt wird, ist durchaus unterschiedlich in Bezug auf die verschiedenen Anbieter von NoSQL-Datenbanken. Für die Oracle NoSQL DB gilt, dass Schreibvorgänge immer nur auf den Masterknoten erfolgen und von dort auf die Replikaknoten geschrieben werden *(siehe Abbildung 1)*.

Für Lese-Operationen kann sowohl auf den Master- als auch auf einen der Replikaknoten zugegriffen werden. Mit diesem Vorgehen erklärt sich auch die „Eventual Consistency“, denn zwischen dem Schreiben auf dem Masterknoten und von dort auf die Replikaknoten gibt es eine, wenn auch geringfügige, Zeitspanne. Soll Konsistenz im Sinne von ACID gewährt werden, braucht es mindestens die Rückbestätigung der Replikaknoten, bis auch sie die jeweils aktuelle Version eines Datensatzes festgeschrieben haben.

Anhand der Topologie in *Abbildung 1* lässt sich leicht ableiten, wie Ausfallsicherheit beziehungsweise Hochverfügbarkeit garantiert werden kann. Fällt ein Speicherknoten, etwa die Maschine 1, aus, wird innerhalb der Replika-Gruppen (Gruppe = ein Master und dessen Replikas) auf den verbleibenden Speicherknoten (Maschinen 2 und 3) jeweils ein neuer Master bestimmt. Somit ist sichergestellt, dass weiter Daten geschrieben werden. Lesevorgänge verteilen sich ebenso auf die übrigen Knoten, die im beschriebenen Fall ja noch mindestens zwei Kopien der Daten vorhalten.

Wie funktioniert das alles zusammen? Auf jedem der Speicherknoten wird die Software installiert, vorzugsweise in das gleiche Verzeichnis. Teil der Software sind die sogenannten „Agents“, die für den Betrieb gestartet werden müssen. Für die Entwicklung der Anwendung stehen zwei APIs (Key/Value, Table) und Treiber für die gängigen Programmiersprachen *(siehe Abbildung 2)* zur Verfügung. Sofern keine eigene Präferenz vorliegt, empfiehlt Oracle die Benutzung des Table-API. Darüber sind die meisten Programmiersprachen unterstützt *[1]*. Die Treiber werden, wie für die jeweilige Programmiersprache üblich, in die Anwendung eingebunden. Über diese erfolgt die Kommunikation mit den Agents auf den Speicherknoten.

Wie die Anwendung schließlich die Daten verteilt und wieder auf sie zugreift, ist also Aufgabe des Treibers. Programmatisch beeinflussbar ist allerdings das Konsistenzverhalten, das in Abhängigkeit von den Anforderungen feingranular im Bereich von „None“ (kann nicht aktuelle Daten lesen) bis „Absolute“ (liest immer aktuelle Daten) eingestellt werden kann.

Betriebsvarianten der Oracle NoSQL DB

Wo und wie die Datenbank aufgesetzt wird, entscheidet sich an ein paar wenigen Fragen:

- Soll oder kann die Bereitstellung als Cloud-Services erfolgen?
- Wer übernimmt das Management der Datenbank?

Die aktuell einfachste Form der Bereitstellung ist die über den Oracle NoSQL DB Cloud-Service *[2]*. Damit wird insbesondere die Anwendungsentwicklung angesprochen, da außer der Einrichtung von Nutzer-Accounts keinerlei administrativer Aufwand anfällt.

Für die Entwicklung der Anwendung stehen sowohl Treiber als auch zusätzliche Werkzeuge wie ein Cloud-Simulator zur Verfügung. Die Treiber setzen alle auf dem Table-API auf. Der Cloud-Simulator ist Teil des Oracle NoSQL Cloud Software Development Kit (SDK) und simuliert für Testzwecke die Anwendung so, als ob diese auf dem Cloud-Service liefe. Beispielcode steht im Installationsverzeichnis für das SDK, um für die ersten Tests genutzt zu werden. Sind alle Tests erfolgreich abgeschlossen, kann mit wenigen Änderungen an den Verbindungsparametern sowie mit dem Anlegen einer lokalen Credential-Datei der Umstieg auf den Cloud-Service erfolgen. Die Credential-Datei enthält vier Einträge: Benutzername, Passwort, Client-ID und Client-Secret.

Ab dem Moment, wenn eine Anwendung erstmalig mit dem Oracle NoSQL DB Cloud-Service verbunden wird, erscheint ein entsprechender Eintrag in der Applikationsübersicht. Im Beispiel, das in *Abbildung 3* dargestellt ist, wurden zwei Anwendungen erzeugt. Eine davon verwaltet Nutzerdaten, die andere Profile von Bankkunden.

Alle, die an der Anwendungsentwicklung beteiligt sind, benötigen also einen Cloud-Account. Dieser kann für einen bereits abonnierten Cloud-Service von derjenigen Person eingerichtet werden, die als Cloud-Administrator registriert ist. Den Nutzerprofilen muss dann zusätzlich noch eines der folgenden Rechte zugeordnet sein *(siehe Abbildung 4)*.

Alternativ kann für Evaluierungszwecke auch mit einem Trial-Account gear-

Oracle NoSQL Database Drivers

The Oracle NoSQL Database Driver are licensed pursuant to the Apache 2.0 License (Apache 2.0) and are used with both the Community Edition (CE) and Enterprise Edition (EE). The Apache License and third party notices for the Oracle NoSQL Database Drivers may be viewed at this location or in the downloaded software.

Below is a matrix of the supported combination of released Server version and released Driver version.

Oracle NoSQL Database Drivers

Oracle NoSQL Database Server	Java Driver	Java Script	C-Key Value	C-Table	Python	C# Driver
	18.1.16	18.1.1	4.5.11	18.1.1	18.1.1	18.1.12
18.1.16	✓	✓	✓	✓	✓	✓
4.5.12	✓	✓	✓	✓	✓	✓

Abbildung 2: Verfügbare Treiber

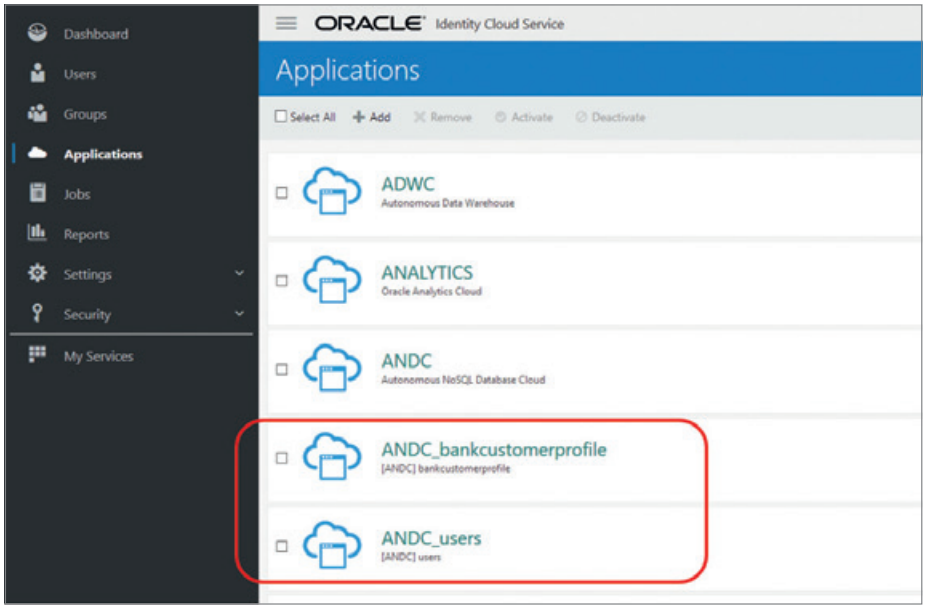


Abbildung 3: Übersicht der dem Account zugeordneten Cloud-Applikationen

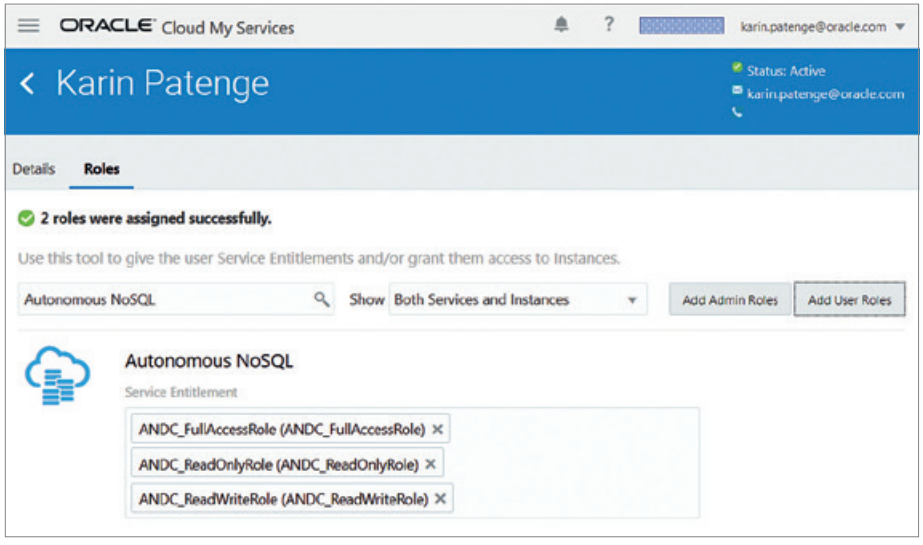


Abbildung 4: Verfügbare Rollen für Nutzung des Oracle NoSQL DB Cloud-Service

```
/* Create a new TableLimits object, setting three values:
 *   read, write, storage units
 */
TableLimits newLimits = new TableLimits(100, 200, 300);

/* Create a new TableRequest.setTableLimits, parsing the TableLimits object
 * and calling TableRequest.setTableName, with the name of the table.
 */
treq = new TableRequest().setTableLimits(newLimits).
    setTableName(tableName);
tres = handle.tableRequest(treq);
System.out.println("Altering table limits");

/* Waiting for the table to become active */
TableResult.waitForState(
    handle,
    tableName,
    TableResult.State.ACTIVE,
    60000,
    60000);

GetTableRequest gtr = new GetTableRequest().setTableName(tableName);
tres = handle.getTable(gtr);
System.out.println(
    "New table limits: " +
    "read units=" + tres.getTableLimits().getReadUnits() +
    ", write units=" + tres.getTableLimits().getWriteUnits() +
    ", table size=" + tres.getTableLimits().getStorageGB());
```

Listing 1: Programmatische Anpassung auf Arbeitslasten (programmiert in Java)

beitet werden. Das Einrichten eines solchen Accounts geht sehr einfach und verursacht keinerlei Kosten, da automatisch ein Budget zugeordnet wird, das nach eigenem Belieben verbraucht werden kann.

Bei der Eingabe der Account-Details ist für den NoSQL DB Cloud-Service einzig zu beachten, dass dieser zum Zeitpunkt der Entstehung des Artikels nur über das Generation-2-Cloud-Rechenzentrum in den USA verfügbar ist. Demzufolge ist als Default Data Region „North America“ anzugeben. Andere Cloud-Rechenzentren wie das in Frankfurt sollen zeitnah folgen. Die Besonderheit dieses Cloud-Service besteht darin, dass über den Programmcode gesteuert werden kann, wie viel Arbeitslast voraussichtlich entsteht (*Listing 1*). Über die einfachen Angaben zu Reads, Writes und Storage Units werden vollautomatisiert im Hintergrund die notwendigen CPU- und Speicher-Ressourcen zugeordnet beziehungsweise sowohl nach oben als auch nach unten skaliert.

Hilfestellung für diese Angaben gibt es über ein kleines, Browser-basiertes Werkzeug, den Capacity Estimator *[3]*. Dessen Parametrisierung ist beispielhaft in *Abbildung 5* gezeigt, das Ergebnis in *Abbildung 6*.

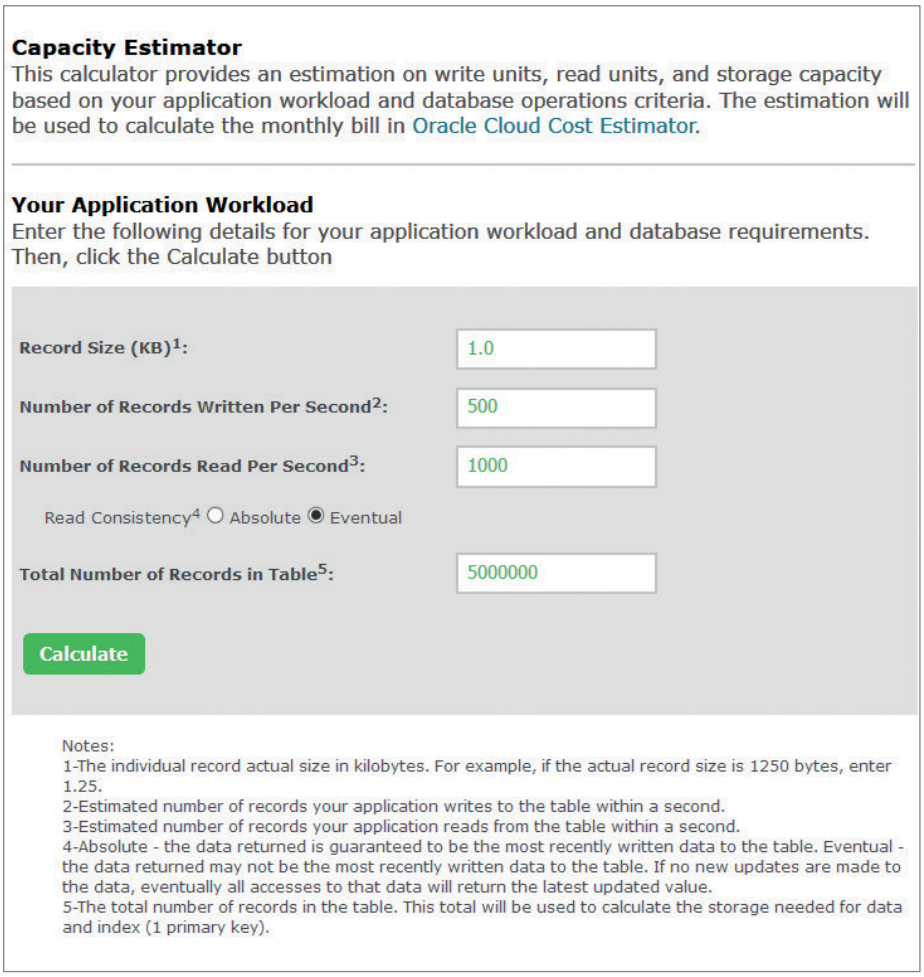


Abbildung 5: Capacity Estimator – Parameter

Für den Betrieb der Oracle NoSQL DB in der Oracle-Cloud gibt es eine weitere Option, nämlich als „Do-it-yourself"-Cluster auf den Infrastructure Services (IaaS). Das Vorgehen dafür ist sehr ausführlich in einem Whitepaper *[4]* beschrieben, auf das über den Oracle NoSQL DB Blog *[5]* verwiesen wird. Für diese Option ist zu beachten, dass im Unterschied zu dem „fully managed" Oracle NoSQL DB Cloud-Service hier die administrativen Aufgaben wie Installation, Konfiguration, Deployment, Patchen etc. auf der Nutzerseite liegen. Andererseits bietet diese Option das Betreiben der NoSQL DB auf Bare-Metal-Cloud-Ressourcen. Diese werden dediziert zugeordnet und ermöglichen daher vorhersagbare Latenzzeiten und Durchsatz. Wer also höchste Performance braucht, entscheidet sich für diese Option.

Wichtig ist zudem, auch die eingangs erwähnte Community-Edition auf diese Art nutzen zu können. Dadurch fallen Kosten nur für die sehr kostengünstigen Storage- und Compute-Ressourcen an. Netzwerk-Ressourcen sind laut aktueller Preisliste bis zu einem Volumen von 10 TB pro Monat frei *[6]*. Dies bezieht sich sowohl auf den ein- als auch auf den ausgehenden Datenverkehr.

Estimated Capacities for Cost Estimator

Enter the following write units, read units, storage GB in Oracle Cloud Cost Estimator to estimate your monthly billing.

Write Units:	**500**
Read Units:	**1000**
Storage(GB):	**5.38**

Notes:
-Rounded record size for Write and Read Units estimation: 1(KB)
-Actual record size for storage estimation: 1.0(KB)
-Read consistency selected: Eventual
-Per record storage overhead: 76 bytes. Storage capacity includes the raw data size plus the index for primary key.
-Number of decimal points used for storage estimation: 9 (Tip: Round it up based on your requirements.)
-The estimation assumes the same workload over the entire month.

Abbildung 6: Capacity Estimator – Ergebnis

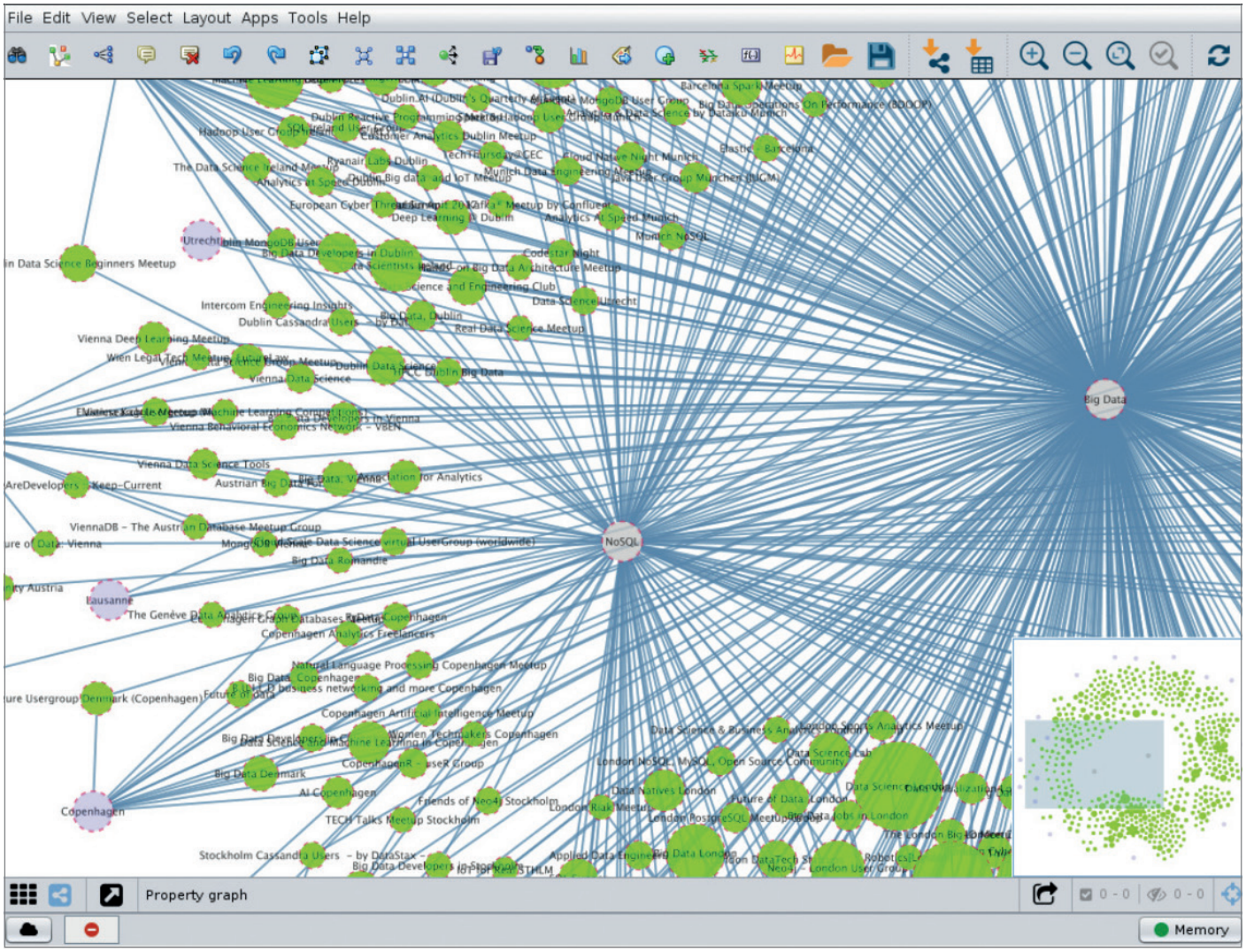


Abbildung 7: Ausgewählte Knoten und Kanten eines Graphen, mit Cytoscape visualisiert

Kommt der Betrieb über Cloud-Services nicht infrage, lässt sich die Oracle NoSQL DB auf eigenen Hardware-Ressourcen installieren. Auch hier kann selbst entschieden werden, mit welcher Topologie das Datenbank-Cluster aufgesetzt werden oder ob etwa ein zweites Rechenzentrum den Ausfall zusätzlich absichern soll.

Wer einfach nur die Oracle NoSQL DB evaluieren möchte, dafür aber weder Cloud-Services abonnieren kann noch den Aufwand für die Installation betreiben möchte, kann sich kostenfrei eine der beiden „Prebuilt Virtual Machines" aus dem Oracle Technology Network herunterladen:

- Oracle VM mit der NoSQL DB Community Edition *[7]*
- Oracle Big Data Lite Virtual Machine *[8]*

Obwohl voll funktionsfähig, sind beide Varianten nicht für den produktiven Einsatz vorgesehen. Die Big Data Lite Virtual Machine arbeitet gar nur auf einem Cluster-Knoten und daher ohne Replikation. Um sich gegen Ausfall abzusichern, muss also mit Snapshots entweder der NoSQL DB oder der virtuellen Maschine gearbeitet werden.

```
server = new ArrayList();
server.add("bigdatalite.localdomain:5000");

// Graph configuration
// Graph name: "meetup"
// KV store name: "kvstore"

cfg = GraphConfigBuilder.forPropertyGraphNosql() \
.setName("meetup") \
.setStoreName("kvstore") \
.setHosts(server) \
.setMaxNumConnections(2) \
.hasEdgeLabel(true) \
.setLoadEdgeLabel(true) \
.addVertexProperty("type", PropertyType.STRING, "NA") \
.addVertexProperty("city_name", PropertyType.STRING, "NA") \
…
.setLoadVertexLabels(true) \
.setUseVertexPropertyValueAsLabel("type") \
.setPropertyValueDelimiter(",") \
.build();

opg = OraclePropertyGraph.getInstance(cfg);
opg.getKVStoreConfig();

// Prepare for data load
opg.setClearTableDOP(2);
opg.clearRepository();

opgdl=OraclePropertyGraphDataLoader.getInstance();
vfile="/home/oracle/Documents/Meetup/data/meetup.opv";
efile="/home/oracle/Documents/Meetup/data/meetup.ope";

// Load data
opgdl.loadData(opg, vfile, efile, 2);
…
```

Listing 3: Laden von Graph-Daten in die Oracle NoSQL DB

Anwendung als Graph-Datenbank

Die Big Data Lite Virtual Machine eignet sich übrigens in besonderer Weise, um zu sehen, wie die Oracle NoSQL DB als Graph-Datenbank arbeitet. Denn auch Big Data Spatial and Graph ist als zusätzliche Software-Komponente bereits vorinstalliert. *Listing 2 (siehe http://www.doag.org/go/redstack201806listings)* gibt Einblicke unter anderem in die Topologie, mit der die Datenbank aufgesetzt wurde.

Die Datenbank kann jetzt genutzt werden, um Daten als Graph abzuspeichern und diese zu analysieren. Die Analyse kann dabei sowohl Kommandozeilen-basiert als auch über Werkzeuge wie Jupyter Notebook oder Cytoscape erfolgen. Für Jupyter Notebook gibt es einen entsprechenden Interpreter, für Cytoscape ein Plug-in. *Listing 3* zeigt einen Skript-Ausschnitt zum Laden von Graph-Daten, die in Form von zwei Dateien übergeben werden:

- Knoten-Datei („Vertices" oder „Nodes") mit der Endung „.opv"
- Kanten-Datei („Edges") mit der Endung „.ope"

Im Ergebnis entsteht ein Graph mit rund 90.000 Knoten und 120.000 Kanten, beispielhaft in *Abbildung 7* mit Cytoscape und dem Plug-in für die Oracle NoSQL DB visualisiert.

Fazit

Wer eine NoSQL-Datenbank sucht, die sowohl mehrere Datenmodelle als auch Programmiersprachen unterstützt, zudem für höchste Performanz ausgelegt ist und sowohl in der Cloud als auch auf eigener Hardware betrieben werden kann, wird im Datenbank-Portfolio von Oracle fündig. Die Oracle NoSQL DB Community Edition kann kostenfrei über die Apache-2.0-Lizenz genutzt werden.

Referenzen

[1] *http://tinyurl.com/nosqldbdoc*
[2] *http://cloud.oracle.com/nosql*
[3] *http://tinyurl.com/nosqldbcapacityestimator*
[4] *http://tinyurl.com/nosqldbbaremetal*
[5] *http://tinyurl.com/nosqldbblog*
[6] *http://tinyurl.com/oracloudpricing*
[7] *http://tinyurl.com/nosqldbdownload*
[8] *http://tinyurl.com/bigdatalitevm*

Karin Patenge
karin.patenge@oracle.com

NoSQL, NewSQL und Cloud-native Datenbanken

Andreas Buckenhofer, Daimler TSS

Die ersten NoSQL-Datenbanken sind um das Jahr 2000 entstanden. Unternehmen wie Google, Amazon oder Twitter entwickelten eigene Datenbanken für ihre spezifischen Anforderungen. Im Laufe der Zeit stehen viele dieser Datenbanken als Open Source zur Verfügung.

In den 2000er-Jahren entstanden eine Vielzahl von Datenbanken, die als NoSQL zusammengefasst sind. Sie zeichnen sich dadurch aus, dass die Datenspeicherung verteilt erfolgt, Replikation zum Einsatz kommt, hohe Verfügbarkeit besteht und die Daten nicht in relationalen Tabellen gespeichert sind. Durch den Verzicht auf SQL soll der Impedance Mismatch vermieden werden, der auftritt, wenn man Objekte einer objektorientierten Programmiersprache in einer relationalen Datenbank speichert.

NoSQL-Datenbank-Kategorien

NoSQL basiert auf speziellen Datenmodellen, die im Gegensatz zum relationalen Datenbankmodell nicht mehr universell einsetzbar sind, sondern einen bestimmten Anwendungszweck erfüllen. Es wird zwischen vier verschiedenen NoSQL-Datenbank-Kategorien unterschieden:

- Key-Value Stores
- Document Stores
- Wide Column Stores
- Graph Stores

Key-Value Stores bieten ein leicht verständliches Datenmodell. Daten lassen sich einfach verteilen und somit hohe Skalierbarkeit erreichen. Key-Value Stores werden häufig als Caches eingesetzt, entweder als eigenständige Datenbank oder als Caching-Layer für bestimmte Daten zwischen der Anwendung und einer relationalen Datenbank. *Abbildung 1* zeigt das Datenmodell.

Key-Value Stores bestehen aus einem eindeutigen Zugriffsschlüssel (Key) und den Nutzdaten (Value). Die Nutzdaten können vielfältig sein, etwa ein einzelner Wert oder komplexe Strukturen. Als Operatoren stehen „put“, „get“ und „delete“ zur Verfügung. Typischerweise kann nur der Key indexiert werden, da über diesen der Zugriff erfolgt.

Oracle NoSQL, Redis, Amazon Dynamo und Riak sind typische Vertreter dieser Kategorie. Der Übergang zu Document Stores ist fließend. Oracle NoSQL hat beispielsweise auch Eigenschaften von Document Stores.

Document Stores basieren auf einem Datenmodell, in dem zu einem Schlüssel komplex strukturierte Daten („Dokumente“) als Wert zugeordnet sind *(siehe Abbildung 2)*. Solche Dokumente sind typischerweise JavaScript Object Notation (JSON) oder XML-Strukturen. Der Zugriff auf Daten erfolgt nicht nur über den Schlüssel im Vergleich zu Key-Value Stores, sondern auch über Daten im Dokument. Dazu ist eine Indexierung der Zugriffsschlüssel nötig, um eine performante Suche zu ermöglichen.

JSON wird zur Serialisierung von vielen Datenbank-Systemen als primäres (etwa bei MongoDB, CouchDB, Couchbase) oder zusätzliches Datenformat unterstützt. JSON-Strukturen erlauben die Einbettung vieler Datenelemente in ein Dokument, beispielsweise Personendaten einschließlich Adressen und Blog-Artikel. Diese Normalisierung vermeidet Joins. Besonders aufwendig sind dagegen Updates, da aufgrund der Denormalisierung unter Umständen viele Datensätze aktualisiert werden müssen.

Eine Standard-Abfragesprache wie SQL sucht man für Document Stores vergeblich. Proprietäre Sprachen, die teilweise SQL-artige Elemente übernehmen, müssen zum Erzeugen, Ändern und Löschen von Dokumenten genutzt werden; so gibt es beispielsweise Operationen wie „insertOne“ oder „insertMany“ zum Einfügen eines oder mehrerer Dokumente.

Wide Column Stores verwenden ein Datenmodell, das aus beliebig vielen Spalten besteht. Einzelne Zeilen können dabei verschiedene Spalten aufweisen *(siehe Abbildung 3)*. Die Zellen von Spalten können atomare Daten oder komplexe Strukturen wie Dokumente beinhalten; der Zugriff erfolgt über den

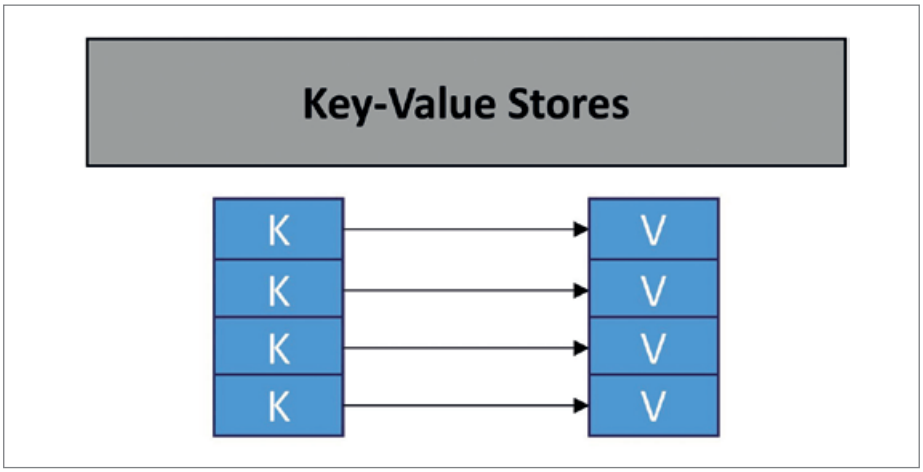


Abbildung 1: Key-Value Stores

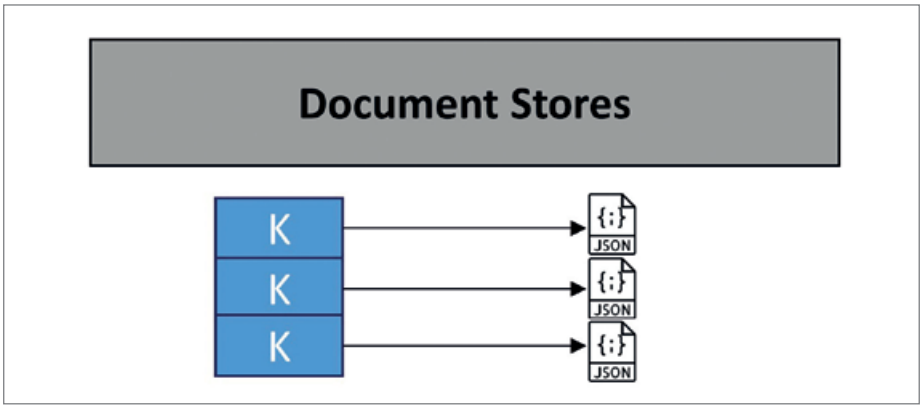


Abbildung 2: Document Stores

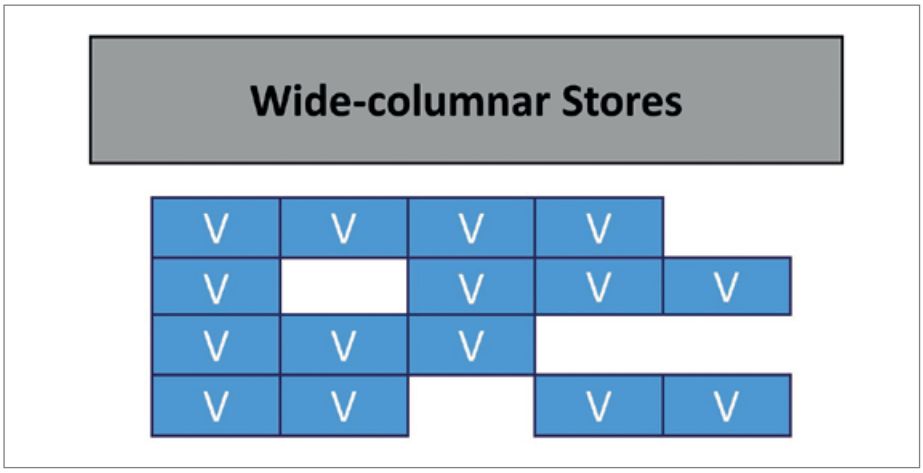


Abbildung 3: Wide Column Stores

Schlüssel. Diese Datenbanken sind hochskalierbar beim Schreiben sowie beim Lesen weniger Datensätze über den Schlüssel. Bekannteste Vertreter dieser Kategorie sind Cassandra und HBase als Bestandteil des Hadoop-Stacks. Ein Einsatz-Szenario von Cassandra ist die Speicherung von Ausstattungsmerkmalen bei der individuellen Zusammenstellung eines Fahrzeugs in einem webbasierten Online-Fahrzeugkonfigurator.

Graph-Datenbanken *(siehe Abbildung 4)* zählen ebenfalls zur NoSQL-Kategorie, auch wenn Skalierbarkeit und Verteilung nicht zu den Stärken dieser Datenbanken gehören. Sie speichern Daten als Knoten und Kanten ab. Haupt-Anwendungsgebiet der Graph-Datenbanken ist

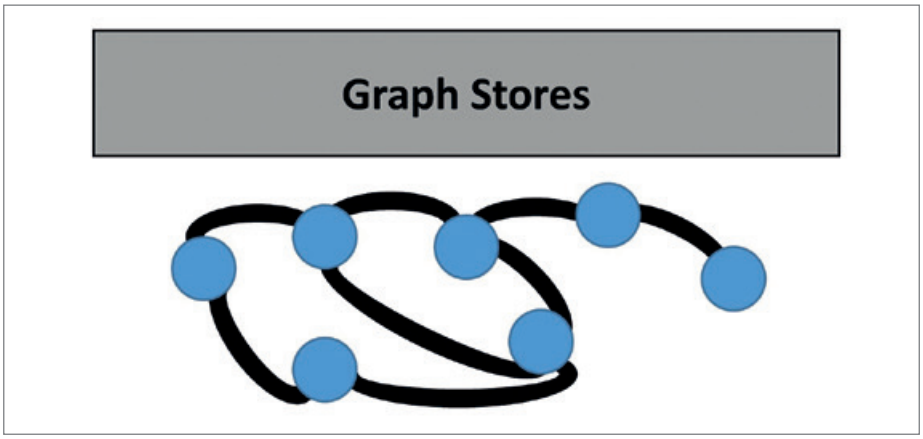


Abbildung 4: Graph Stores

die Abbildung von Beziehungen. Diese Datenbanken ermöglichen ein sehr performantes Traversieren von Kanten. Bekannteste Datenbank dieser Kategorie ist Neo4j.

ACID, BASE und CAP

ACID ist eine bekannte Eigenschaft von Transaktionen in relationalen Datenbanken, insbesondere zur Sicherstellung von Konsistenz. Konsistenz wird bei NoSQL sehr häufig aufgeweicht. Laut dem CAP-Theorem von Brewer kann ein verteiltes System zwei der folgenden Eigenschaften annehmen: Konsistenz (C = consistency), Verfügbarkeit (A = availability) und Partitionstoleranz (P = partition tolerance). AP-Systeme geben strenge Konsistenz auf zugunsten von Verfügbarkeit und Ausfallsicherheit.

Für diese Datenbanken gilt das BASE-Prinzip (Basically Available, Soft state, Eventual consistency). Die Konsistenz stellt sich im Laufe der Zeit ein, sobald alle verteilten Datenbankknoten die Änderung erhalten haben. Das Verhalten einer Anwendung ohne strenge Konsistenz lässt sich beispielsweise auf Twitter regelmäßig beobachten: Neue Follower werden namentlich gelistet, der Zähler für die Anzahl der Follower wird oft jedoch erst später aktualisiert.

Das folgende Beispiel zeigt die Auswirkungen von BASE. Angenommen, der Wert der Variable X ist 1 und es werden die Operationen X = 5 und danach auf einem anderen Netzwerkknoten Y = X + 2 durchgeführt. Ein System mit strenger Konsistenz wird den Wert 7 liefern. Ein BASE-System dagegen liefert 3 oder 7 beziehungsweise nach einiger Zeit nur noch 7. Ein sehr lesenswerter Artikel zu CAP und BASE stammt von Daniel Abadi *[1]*.

NewSQL-Datenbanken

Das Fehlen strenger Konsistenz in der Persistenz-Schicht macht eine Anwendung sehr komplex. Die Verlagerung der Konsistenz-Sicherung in die Anwendung erweist sich als zu große Herausforderung, zu aufwendig und viel zu fehleranfällig. In einem White Paper über die NewSQL-Datenbank F1 bringt Google die Problematik auf den Punkt: „We also have a lot of experience with eventual consistency systems at Google. In all such systems, we find developers spend a significant fraction of their time building extremely complex and error-prone mechanisms to cope with eventual consistency and handle data that may be out of date. We think this is an unacceptable burden to place on developers and that consistency problems should be solved at the database level. Full transactional consistency is one of the most important properties of F1“ *[2]*.

Auch SQL als standardisierte Anfragesprache wurde schnell als wichtig erkannt. Datenbanken wie VoltDB versprechen die Flexibilität von NoSQL sowie wichtige Features wie strenge Konsistenz und SQL.

Cloud-native Datenbanken

Cloud-native ist der neueste Trend und basiert wie NoSQL und NewSQL auf Verteilung, Replikation und Hochverfügbarkeit. Cloud-native Datenbanken sind idealerweise bereits für die Nutzung in der Cloud entwickelt. Eigenschaften solcher Anwendungen sind von der Cloud-native Computing Foundation (CNCF) definiert:

- *Ubiquitous and flexible*
  Die Datenbank muss in beliebigen Container-Technologien in beliebigen Cloud-Umgebungen lauffähig sein
- *Resilient and scalable*
  Der Ausfall von Knoten darf nicht durch hohe Verfügbarkeit, Redundanz und automatischen Fail-Over bemerkbar sein
- *Dynamic*
  Upgrades erfolgen automatisch
- *Automatable*
  Alles ist im Sinne einer Infrastructure as Code automatisiert
- *Observable*
  Loggin, Tracing, Metriken müssen auch noch verfügbar sein, wenn der Container nicht mehr existiert
- *Distributed*
  Nutzung der Vorteile einer verteilten Cloud

Beispiele für Cloud-native Datenbanken sind Google Spanner sowie die Open-Source-Variante CockroachDB.

Ausblick

Wie reagieren die relationalen Datenbank-Hersteller auf die Herausforderungen durch NoSQL, NewSQL und Cloud-native Datenbanken? In der Vergangenheit gab es bereits Ergänzungen, die längst in den SQL-Standard eingeflossen sind:

- In den 1990er-Jahren wurden objektorientierte Konzepte als Antwort auf objektorientierte Datenbanken integriert
- In den 2000er-Jahren wurden XML-Konzepte als Antwort auf XML-Datenbanken integriert

Wird der Einsatz einer NoSQL-, NewSQL- oder Cloud-ready Datenbank in Erwägung gezogen, sind Security-Features kritisch zu betrachten („Enterprise-readiness“). Relationale Datenbanken verfü-

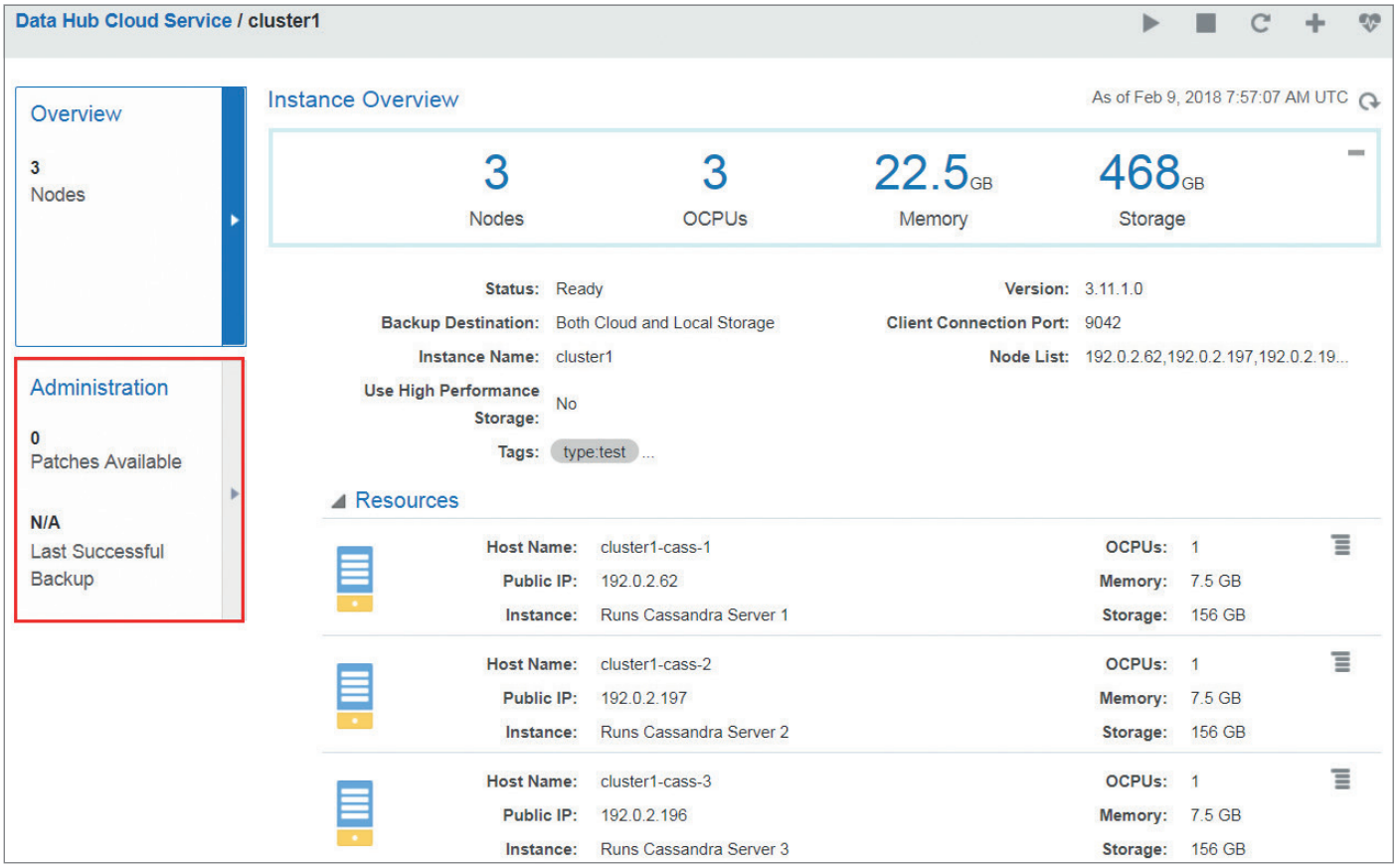


Abbildung 5: Apache Cassandra Cluster im Oracle Data Hub Cloud Service [3]

gen über umfangreiche und ausgereifte Features. Neue Datenbanken haben noch Nachholbedarf, sodass ein benötigtes Security Feature noch fehlen kann. Auch auf die Herausforderungen durch NoSQL-, NewSQL- und Cloud-native Datenbanken gibt es Antworten:

- Flexibles Datenmodell für Schema-on-read durch JSON-Unterstützung (SQL-Standard 2016)
- Nutzung von Oracle OWF zur Speicherung Graph-basierter Daten
- MemOptimized RowStore als Alternative zu Key-Value-Stores
- Autonomous Datenbanken für bestimmte Szenarien zur Verringerung von Tuning-Maßnahmen und betriebliche Tätigkeiten
- Weitere längst bekannte Maßnahmen, etwa durch Informational (Deferred) Constraints, Partitionierung, Parallelisierung etc.

In der Oracle Cloud sind NoSQL-Datenbanken als „Managed Services" verfügbar:

- Oracle NoSQL wird als autonome Datenbank angeboten
- Cassandra kann im Oracle Data Hub Cloud Service konfiguriert werden *(siehe Abbildung 5)*
- Mit Bitnami besteht eine Partnerschaft, um MongoDB in der Oracle Cloud verfügbar zu machen *[4]*
- Apache HBase ist innerhalb des Big Data Cloud Service nutzbar
- Container-Technologien wie Docker und Kubernetes können genutzt werden

Fazit

Im Gegensatz zu den objektorientierten beziehungsweise XML-Datenbanken werden NoSQL-, NewSQL- und Cloud-ready Datenbanken bestehen bleiben. Diese haben sich bereits einen Anteil des Gesamtmarktes gesichert und sind für bestimmte Use Cases sinnvoll.

Referenzen

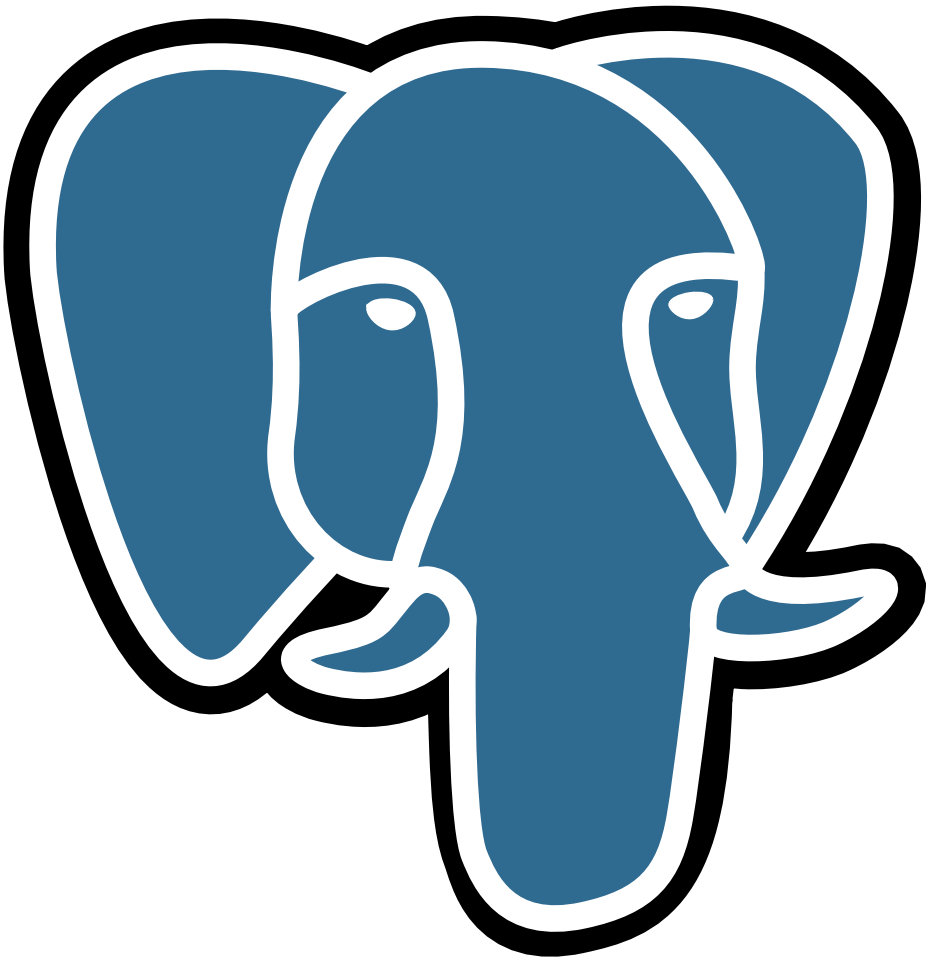
[1] NewSQL database systems are failing to guarantee consistency, and I blame Spanner: *http://dbmsmusings.blogspot.com/2018/09/newsql-database-systems-are-failing-to.html*

[2] F1, A Distributed SQL Database That Scales: *https://db.disi.unitn.eu//pages/VLDBProgram/pdf/industry/p769-shute.pdf*

[3] *https://www.oracle.com/webfolder/technetwork/tutorials/obe/cloud/dhcs/getstarted/dhcs.html*

[4] *https://bitnami.com/stack/mongodb/cloud/oracle*

Andreas Buckenhofer
andreas.buckenhofer@daimler.com



PostgreSQL – der neue Standard für Allzweck-Datenbanken

Jan Karremans, EnterpriseDB

Jeder hat den Namen wahrscheinlich schon mal irgendwo gehört: PostgreSQL oder kurz: Postgres. Aber was ist das eigentlich? Das hat doch irgendetwas mit Open Source zu tun? Bei einer Datenbank erscheint das zunächst ein bisschen merkwürdig. So etwas Hochkompliziertes wie eine hochverfügbare, geschäftskritische Fünf-Terabyte-Datenbank, gebaut von irgendwelchen dubiosen Hoodie-Trägern in schummerigen Kellern, darauf soll sich ein Unternehmen verlassen?

Mitte der siebziger Jahre arbeiteten die Herren Edgar Codd und Christopher Date bei IBM an der Umsetzung des von Codd im Jahr 1970 vorgestellten relationalen Datenmodells. Innerhalb dieses Forschungsprojekts entstand die erste tatsächliche Implementierung von SQL. Wikipedia meldet dazu *[1]*: „System R ist ein von IBM im San Jose Research Center (heute IBM Almaden Research Center) entwickeltes Datenbank-Managementsystem. Das System wurde in den 1970er Jahren im Rahmen eines Forschungsprojekts entwickelt. Es ist historisch von großer Bedeutung, da es das erste relationale Datenbank-Managementsystem war und die Abfragesprache SEQUEL (= Structured English Query Language) definierte, aus der die SQL-Abfragesprache hervorging. Durch die Erfahrungen mit dem System R wurde von der IBM das System SQL/DS entwickelt und ab dem Jahr 1981 bei Kunden eingesetzt. SQL/DS ist der Vorgänger des relationalen Datenbank-Managementsystems DB2."

Es ist interessant zu sehen, dass die relationale Theorie erfunden wurde, um die Probleme mit NoSQL-Datenbanken, die es damals nämlich schon gab, zu beheben. Basierend auf dieser Theorie wurden unterschiedliche Projekte gestartet, von denen Relational Software, im Jahr 1977 gegründet – jetzt besser bekannt als „Oracle" –, das erste richtige kommerziell erfolgreiche Produkt war.

Im Jahr 1973 veröffentlichte IBM das erste Research Paper über das System-R-Projekt. Michael Stonebraker und Eugene Wong von der Berkeley University waren daran stark interessiert und haben darauf basierend das Projekt „University Ingres" oder „Berkeley Ingres" begonnen. Nach der Fertigstellung des ersten Prototyps im Jahr 1974 wurde das Ingres-Projekt noch bis zum Jahr 1985 an der Uni weitergeführt. Michael Stonebraker startete im Jahr 1985 das Post-Ingres-Projekt mit dem Ziel, die Probleme von derzeitigen Datenbank-Systemen zu beseitigen.

Der erste Prototyp erschien im Jahr 1987. Auf Version 1 im Juni 1989 folgten genau ein Jahr später Version 2 und im Jahr 1991 Version 3. Das Universitätsprojekt endete am 30. Juni 1994. Die Jahre 1994 bis 1996 wurden genutzt, um SQL als Abfragesprache zu implementieren. Die Postgres-Community nahm am 8. Juli 1996 den ersten Entwicklungsserver außerhalb einer Universität in Betrieb.

Postgres ist nicht nur ein Open-Source-Projekt, sondern sogar ein Community-Open-Source-Projekt. Die Entwicklung von Postgres erfolgt völlig offen und unabhängig, anders als etwa MySQL oder MongoDB, deren Weiterentwicklung von Unternehmen getrieben und gesteuert wird *[2]*. Seit dem Jahr 1996 steuert ein „PostgreSQL Core Team" das Projekt. „The PostgreSQL Global Development Group" erhält tatkräftige Unterstützung von Major- und Regular-Contributors *[3]*.

Open-Source-Wellen

Abbildung 1 sieht schön bunt aus. Aber was bedeutet das? Kommerzielle Anwen-

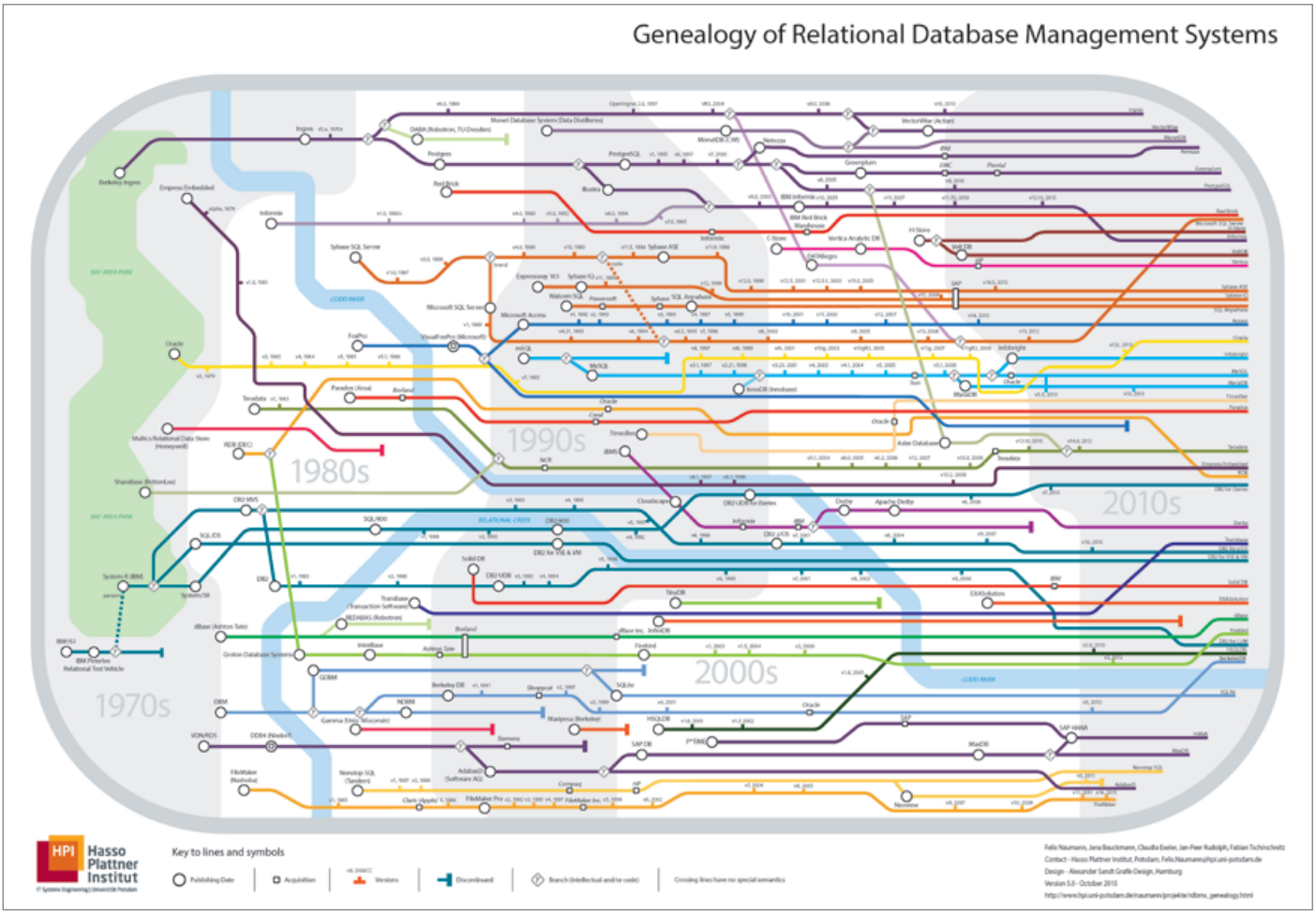


Abbildung 1: Stammbaum der relationalen Datenbank-Systeme

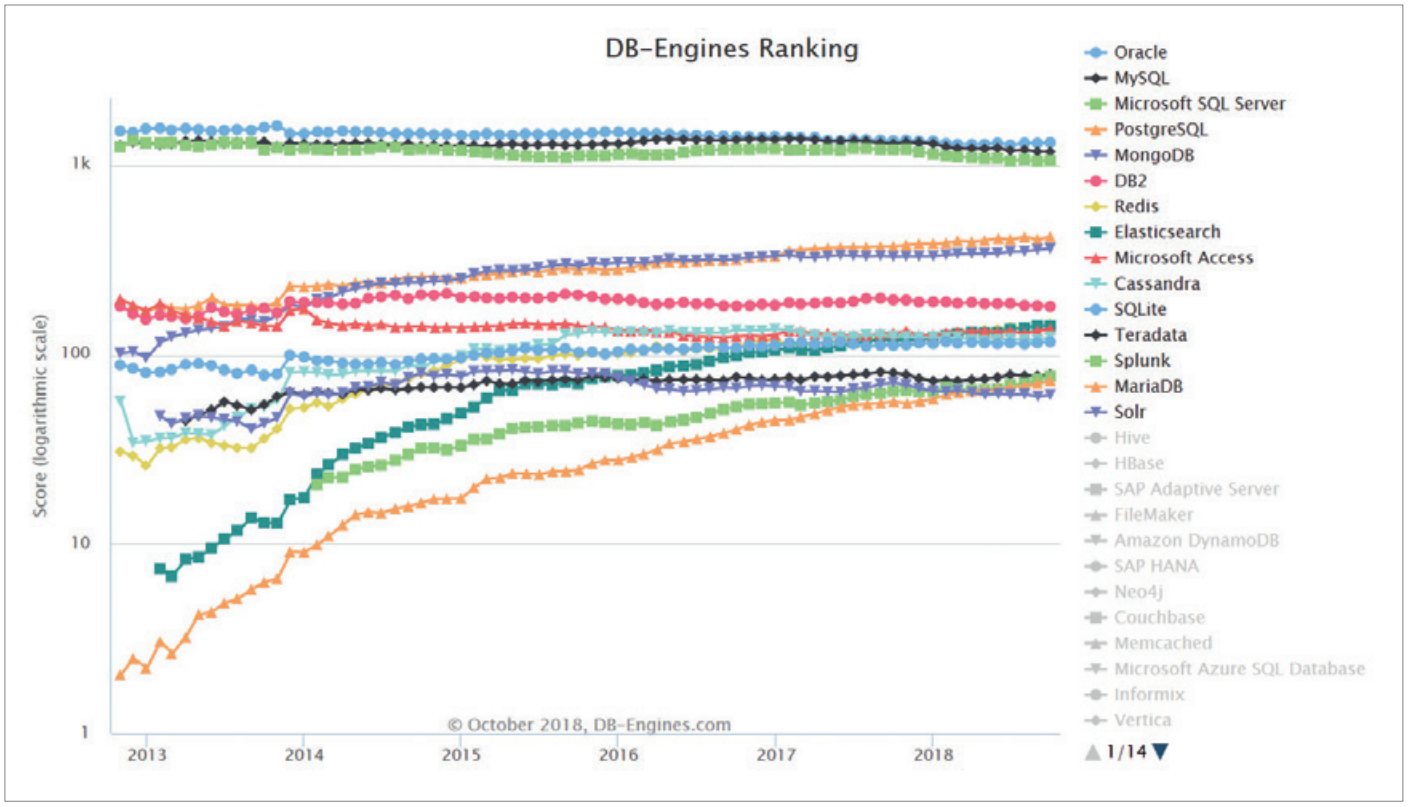


Abbildung 2: Die Verbreitung der Datenbanken

dungen sind seit vielen Jahren sehr erfolgreich. Sie werden auch in Zukunft erfolgreich sein, aber immer mehr aus dem Mainstream verdrängt. Die erste Open-Source-Welle hat gezeigt, worum es geht: Heute wird der größte Teil aller Server mit Linux, einem Open-Source-Betriebssystem, gefahren. Traditionelle Betriebssysteme, die Unternehmen bislang erfolgreich beim Wachstum unterstützt haben, wie die unterschiedlichen Varianten von Unix und Microsoft Windows Server, wurden aus dem Mainstream verdrängt und überleben in der Nische.

Das Gleiche passiert gerade ein weiteres Mal: Die zweite Open-Source-Welle rollt – Datenbanken werden mehr und mehr zum Mainstream. Dieselbe Bewegung, die zuvor Linux groß gemacht hat, bringt jetzt Open-Source-Datenbanken hervor *[4]*.

Der Blick auf die Geschichte hat gezeigt, dass RDBMS entwickelt wurden, um die Probleme der hierarchischen NoSQL-Datenbanken zu lösen. Letztendlich werden Open-Source-Anwendungen für Mainstream-Problembereiche immer mehr an Kraft und Einfluss gewinnen.

PostgreSQL wird unter der PostgreSQL-Lizenz veröffentlicht, einer liberalen Open-Source-Lizenz, ähnlich der BSD- oder MIT-Lizenz. Die Verbreitung von Postgres steigt insbesondere in Deutschland enorm. Postgres-Nutzer brauchen sich allerdings nirgends zu registrieren.

Deshalb ist leider unbekannt, wie viele Postgres-Instanzen es tatsächlich gibt. Ein schöner Indikator ist die Webseite „DB-Engines.com“. Sie ermittelt regelmäßig ein Ranking über die Verbreitung von Datenbanken *(siehe Abbildung 2)*.

Der von DB Engines verwendete Algorithmus basiert auf folgenden Faktoren:

- Anzahl von Erwähnungen auf Webseiten
- Generelles Interesse
- Anzahl der technischen Diskussionen
- Menge an Stellenangeboten
- Menge an Referenzen in Bewerbungen
- Relevanz in sozialen Netzwerken

Postgres wurde von DB Engines im Jahr 2017 zur Datenbank des Jahres gekürt. Daraus kann jeder seine eigenen Schlüsse ziehen.

Was hinter Postgres steckt

Postgres ist ein relationales Datenbank-Management-System (RDBMS). Es ist genau wie beispielsweise Oracle, Microsoft SQL Server oder IBM DB2 eine Multi-Purpose-Datenbank. Man kann also mit einer solchen Datenbank viele verschiedene Anwendungsszenarien abdecken. Dieser Datenbank-Typ ist am weitesten verbreitet.

Postgres verwaltet neben relationalen Daten auch „key-value pairs“. Es kann als Data Warehouse und Document Store eingesetzt werden und hat spezielle Datentypen und eine umfangreiche Unterstützung für geografische (Spatial-) Daten.

Postgres ist als erweiterbare Datenbank entworfen worden. Michael Stonebraker hat schon bei der Entwicklung von University Ingres festgelegt, dass seine relationale Datenbank erweiterbar sein muss. PostgreSQL ist so konzipiert, dass es leicht erweiterbar ist. Aus diesem Grund können in die Datenbank geladene, selbst geschriebene Erweiterungen genauso funktionieren wie eingebaute Features.

Extensions können in verschiedenen Programmiersprachen entwickelt werden. Solche Erweiterungen vergrößern die Funktionalität des Datenbank-Kerns. Da sich eine große Gruppe von Postgres-Entwicklern mit solchen Erweiterungen beschäftigt, sind unzählige Erweiterungen fix und fertig verfügbar.

```
select distinct product_type
     , data->>'brand'        as Brand
     , data->>'available'    as Availability
  from json_data
  join products
    on (
       products.product_type = json_data.data->>'name'
       )
 where json_data.data->>'available' = true
;
```

Listing 1

Postgres ist einfach. Die Postgres-Entwickler haben sich zum Ziel gesetzt, alle notwendige Komplexität in der Datenbank zu verstecken. Das führt dazu, dass Installation und Betrieb einer Postgres-Datenbank sehr einfach sind. So einfach, dass man vielleicht im Anfang denkt: „Das ist so einfach, das kann doch gar nichts sein." Trotzdem funktioniert eine Postgres-Datenbank mindestens genauso gut wie die direkte Konkurrenz.

Diese Einfachheit führt dazu, dass sich Postgres hervorragend in die aktuellen Trends im IT-Sektor einfügt: Die Postgres-Datenbank verhält sich außergewöhnlich gut in einem DevOps-Umfeld. Postgres scheint wie gemacht zu sein für den Einsatz in einer Container-Infrastruktur, da die Installationsgröße angenehm gering ist. Auch für den Einsatz in der Cloud ist Postgres sehr gut geeignet.

Postgres vermischt relationale und NoSQL-Daten. Die native Erweiterbarkeit von Postgres hat den Einbau von NoSQL-Funktionalität sehr einfach gemacht, als diese zu Beginn der 2000er Jahre wieder populär wurde. Mit dem Aufkommen von JSON wurden ab dem Jahr 2010 für Postgres die JSON- und JSONB-Dateitypen entwickelt. Die Erweiterbarkeit von Postgres hat es ermöglicht, dass zusätzlich Operatoren und Funktionen entstanden sind, um die Benutzung von JSON innerhalb der Datenbank zu vereinfachen. Es können Indizes auf JSON-Datentypen gebaut werden, die einzelne Felder indexieren. So lässt sich JSON mit relationalen Daten in einer einzigen Abfrage verknüpfen *(siehe Listing 1)*.

Postgres im Produktiv-Einsatz

Vor dem Einsatz von Postgres in „Mission Critical"-Umgebungen gibt es noch eine letzte Hürde. Eine „Community Open Source"-Lösung hat ihre Vorteile. Sie hat aber auch, so wie die meisten Open-Source-Lösungen, eine Schwachstelle: den Support; insbesondere dann, wenn es sich um einen 24/7-Support für geschäftskritische Systeme handeln soll.

Für den Einsatz von Open-Source-Lösungen im Unternehmen ist diese Hersteller-ähnliche Unterstützung von kritischer Wichtigkeit. Im Linux-Umfeld haben Unternehmen wie Red Hat oder Suse diese Rolle übernommen und bieten den passenden Support an. Im europäischen Raum bieten die Unternehmen „2nd Quadrant" und „EnterpriseDB" diese Unterstützung im Postgres-Umfeld. Die Firma „2nd Quadrant" ist auf den Support von Postgres spezialisiert; sie unterstützt und führt die Postgres-Community. „EnterpriseDB" ist zusätzlich auf Migrationen spezialisiert und begleitet Kunden, die Postgres als Ersatz für Legacy-Datenbank-Umgebungen einsetzen wollen, auf ihrem Weg von Oracle, MS SQL oder IBM zu Postgres. Dazu entwickelt EnterpriseDB die Oracle-Kompatibilitätsschicht, eine Erweiterung von Postgres.

Eine Kombination aus nativen und integrierten Tools ermöglicht den hochverfügbaren Betrieb von Postgres, ähnlich wie die Active-Data-Guard-Lösung von Oracle. Für Backup und Recovery ist ein skriptierbares Tool mit ähnlicher Funktionalität verfügbar, vergleichbar mit RMAN bei Oracle. Monitoring und Wartung der Postgres-Datenbank können komplett über Tools wie den Postgres Enterprise Manager erfolgen. So kann mit Fug und Recht behauptet werden: „Es gibt eigentlich keinen Grund mehr, Postgres nicht zu benutzen!"

Hinweis: Redaktionelle Bearbeitung von Sabine Heimsath und Robert Marz, DOAG Development Community

Referenzen

[1] *https://de.wikipedia.org/wiki/IBM_System_R*
[2] *https://en.wikipedia.org/wiki/PostgreSQL*
[3] *https://www.postgresql.org/community/contributors*
[4] *https://momjian.us/main/writings/pgsql/forever.pdf*

Jan Karremans
jan.karremans@enterprisedb.com

Warum PostgreSQL momentan so erfolgreich ist

Daniel Westermann, dbi services sa

Obwohl PostgreSQL bereits seit mehr als zwanzig Jahren auf dem Markt ist, und zwar durchaus erfolgreich, hat es in den vergangenen Jahren, und speziell im Jahr 2018, enorm an Bedeutung gewonnen. Wird sich dieser Trend die kommenden Jahre fortsetzen oder werden sich die bisherigen Platzhirsche ihre Marktanteile wieder zurückholen? Um diese Frage beantworten zu können, muss man einerseits die Community hinter PostgreSQL verstehen, andererseits aber auch ergründen, warum denn gerade jetzt die Zeit für PostgreSQL gekommen scheint und wie das alles mit anderen Entwicklungen zusammenhängt.

PostgreSQL ist durch und durch Community-getrieben, doch was soll das heißen? Schaut man sich andere Open-Source-Datenbanken wie MariaDB und MySQL an, dann gibt es im Vergleich zu PostgreSQL einen riesigen Unterschied: Bei PostgreSQL gibt es keine Firma, die hinter dem Projekt steht. PostgreSQL kann man also nicht kaufen, weil es keine Firma „PostgreSQL" gibt. Es gibt allerdings etliche Firmen, die Mitarbeiter einstellen, um nur an PostgreSQL zu arbeiten, etwa EnterpriseDB oder Fujitsu. Ganz klar versuchen diese Firmen auch ihre eigenen Wünsche in PostgreSQL einzubringen, sie können das Projekt jedoch nicht kontrollieren.

Egal was in PostgreSQL Einzug halten soll, es ist immer die Community, die entscheidet, ob es schlussendlich integriert wird oder nicht. Man kann natürlich anmerken, dass eine Firma so viele Entwickler anstellen könnte, dass sie die Mehrheit der Community bilden und somit Einfluss nehmen können. In der realen Welt ist das allerdings schon dadurch ausgeschlossen, dass sich so viele Firmen auf dem ganzen Globus an PostgreSQL beteiligen, dass das sehr wahrscheinlich niemals passieren kann. Das hört sich erst einmal alles sehr theoretisch an, wird aber schnell klar, wenn man sich ansieht, welchen Weg ein Patch oder ein neues Feature in PostgreSQL zurücklegt, bis es dann wirklich in PostgreSQL landet.

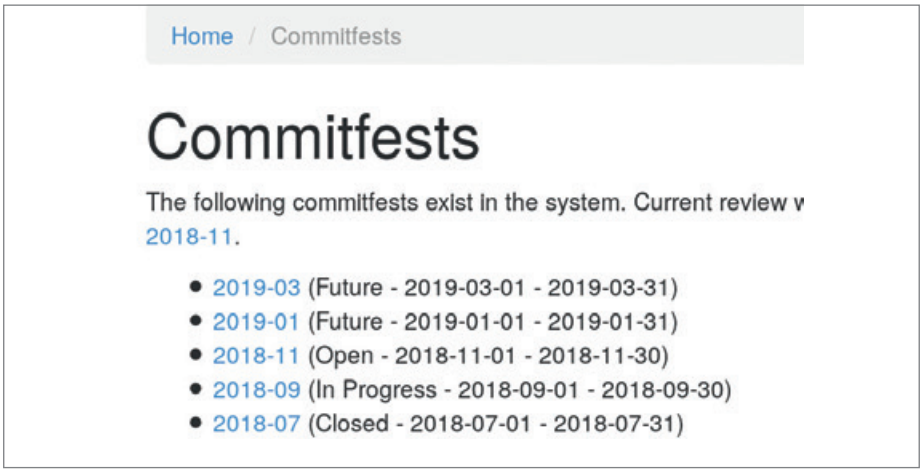


Abbildung 1: Daran arbeitet die PostgreSQL-Community aktuell

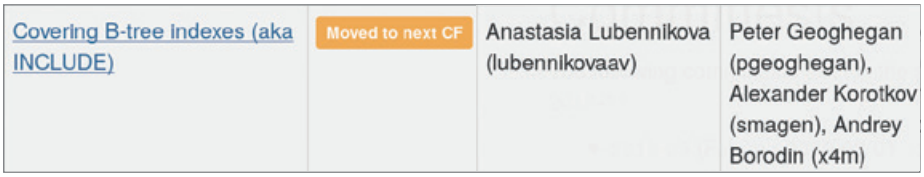


Abbildung 2: So arbeitet die PostgreSQL-Community

Patches/neue Features in PostgreSQL

Um sich zu informieren, woran die PostgreSQL-Community momentan arbeitet, gibt es mehrere Wege. Für Außenstehende führt der einfachste Weg über die sogenannten „Commitfests". Diese sind für alle unter *„https://commitfest.postgresql.org"* einsehbar *(siehe Abbildung 1)*.

Commitfests haben entweder den Status „Closed", „In Progress", „Open" oder „Future". Diejenigen, die mit „Future" gekennzeichnet sind, kann man getrost ignorieren, da man dort nichts finden wird. Commitfests im Status „Open" oder „In Progress" beinhalten das, was momentan entwickelt wird, und unter Status „Closed" befindet sich alles, was in der Vergangenheit war.

Möchte man sich also darüber informieren, was momentan geschieht, sind „2018-09" und „2018-11" von Interesse. Für das Beispiel ist allerdings „2018-01" das Commitfest der Wahl, da es einen Patch enthält, an dem man sehr schön sehen kann, wie die PostgreSQL-Community arbeitet *(siehe Abbildung 2)*.

Jeder Patch oder jedes neue Feature braucht einen Titel (Spalte 1). Allerdings gibt es immer auch einen Status (Spalte 2), eine oder mehrere Personen, die den Patch oder das Feature entwickeln (Spalte 3), und es braucht immer sogenannte „Reviewer", die sich ansehen, was gemacht wurde und ob es den PostgreSQL-Standards entspricht. Reviewer testen, geben Feedback, machen Verbesserungsvorschläge und können durchaus am Ende zu der Entscheidung kommen, dass ein Patch oder Feature nicht weiter verfolgt wird. Um nun die komplette Entwicklung eines Patches oder neuen Features zu verstehen, kann man sich zu diesem Details anzeigen lassen, indem man den Titel auswählt *(siehe Abbildung 3)*.

Anhand des „Status" kann man sehr schnell sehen, in welchem Zustand der Patch momentan ist. In diesem Fall wurde zweimal in ein folgendes Commitfest übertragen, um dann schließlich in „2018-03" in PostgreSQL zu landen (Status „Committed").

Jeder kann sich als „Reviewer" für einen Patch oder ein neues Feature registrieren. Alles was es dafür braucht, ist ein Community Account, den man sich unter *„https://www.postgresql.org/account"* erstellen kann. Schon das Testen, ob ein Patch sauber kompiliert, ist ein Beitrag zum Review-Prozess und kann ein erster Schritt sein, sich an PostgreSQL zu beteiligen.

Das Allerwichtigste an dieser Detail-Seite ist allerdings der Link zur ersten E-Mail an die Mailing-Liste, die zu diesem Patch oder neuem Feature geführt hat. In der PostgreSQL-Welt ist mehr oder weniger alles E-Mail-basiert und es gibt diverse Mailing-Listen, bei denen man sich registrieren kann, um an den Diskussionen teilzuhaben *(siehe „https://www.postgresql.org/list")*. Im Falle eines neuen Features oder eines Patches geht es um die „pgsql-hackers"-Mailing-Liste, und genau diese wird hier referenziert. Folgt man diesem Link, kommt man zum Start der Diskussion zu diesem neuen Feature *(siehe Abbildung 4)*.

Das mag nun nicht weiter bedeutend erscheinen, allerdings hat das einen riesigen Vorteil gegenüber Herstellern von proprietärer Software: Der komplette Entwicklungsprozess ist für alle jederzeit einsehbar und vor allem auch nachvollziehbar. Schaut man sich die Archive der Mailing-Listen genauer an (in diesem Fall die „pgsql-hackers"-List unter *„https://www.postgresql.org/list/pgsql-hackers"*),

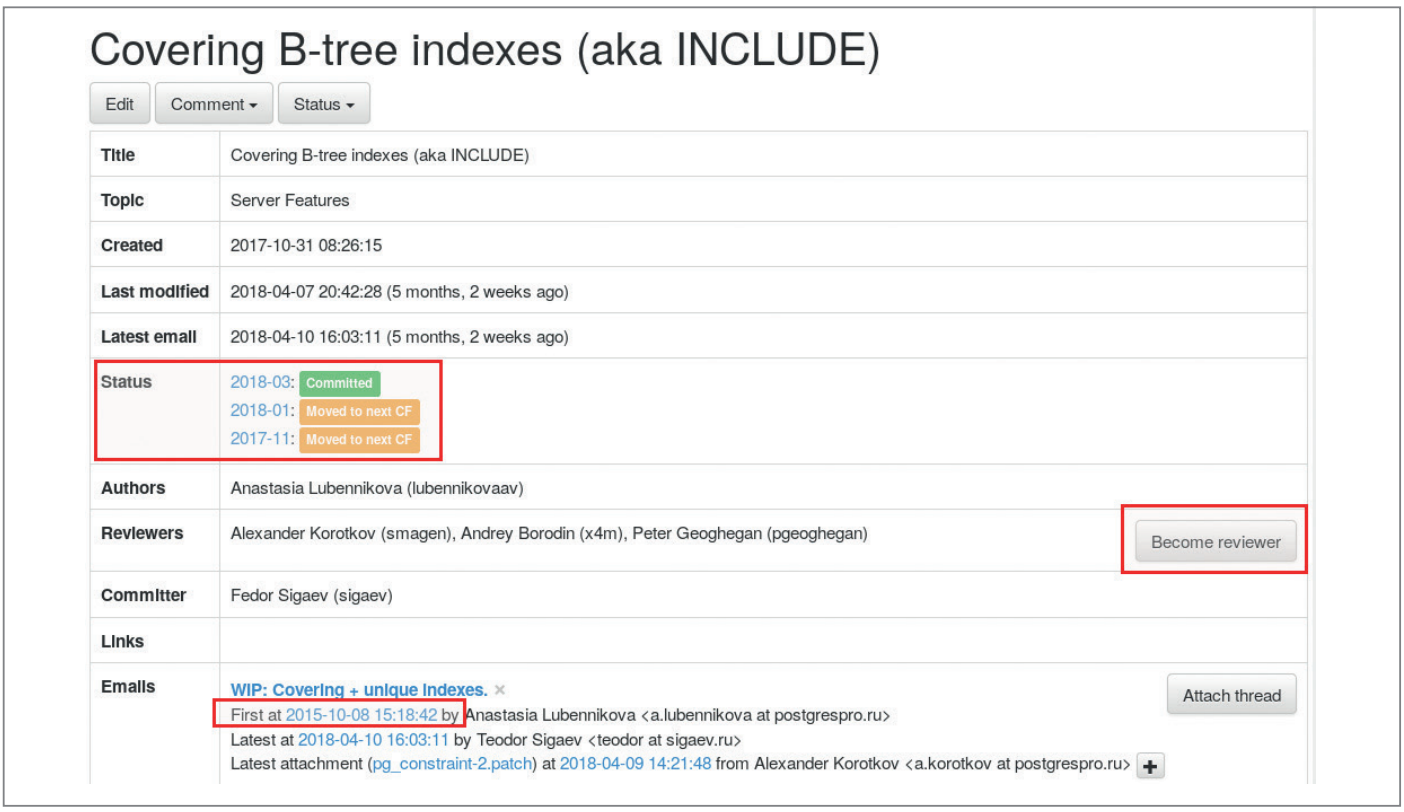


Abbildung 3: Der Verlauf eines Patches

wird man feststellen, dass alle Diskussionen bis zurück ins Jahr 1997 nachgelesen werden können – alle und öffentlich im Internet.

Die volle Transparenz trägt auch zum momentanen Erfolg von PostgreSQL bei, und sei es nur deswegen, weil es ein gutes Gefühl gibt, wenn man dabei sein oder zumindest jederzeit nachvollziehen kann, warum es zu gewissen Entscheidungen kam oder eben nicht. Zudem kann man auch immer abschätzen, was in der nächsten Version von PostgreSQL vorhanden sein wird und was nicht. Zu guter Letzt kann das auch jederzeit von jedem selbst getestet werden, denn auch die Patches an sich sind öffentlich.

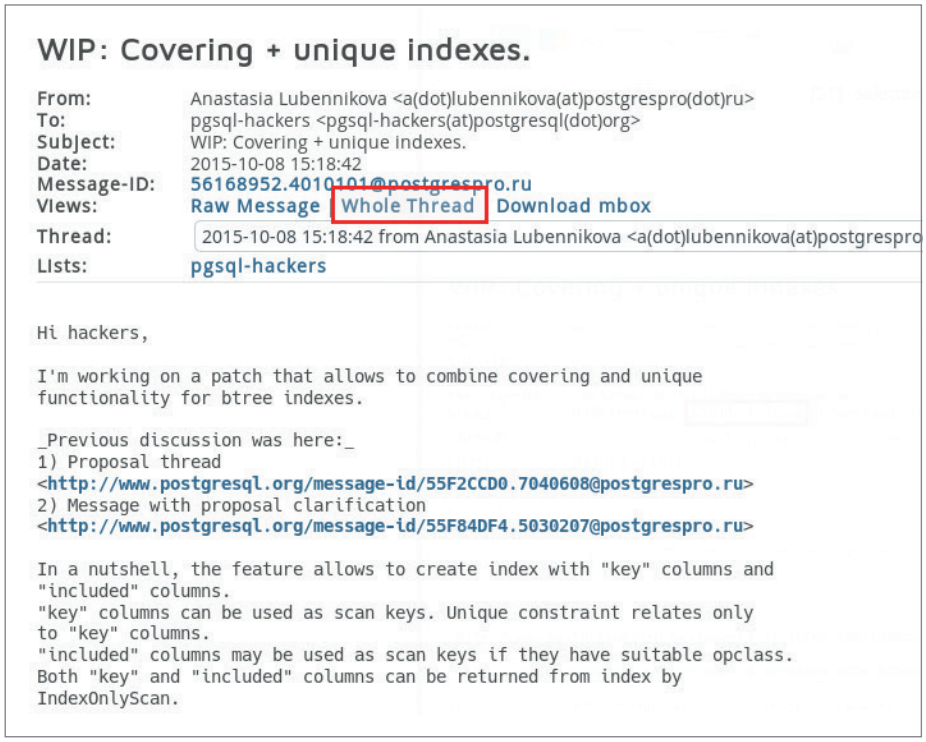


Abbildung 4: Neues Feature einrichten

Die PostgreSQL-Lizenz

Wenn nun aber alles öffentlich zugänglich ist, wie schützt dann die PostgreSQL-Community ihr Produkt? Die Antwort ist so einfach wie bestechend: Gar nicht. Kann nicht sein? Die PostgreSQL-Lizenz steht unter *„https://opensource.org/licenses/postgresql“*. Der wichtigste Auszug daraus ist: „Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.” Was immer man auch mit dem Quellcode machen möchte, man darf es machen. Das geht so weit, dass es auch explizit erlaubt ist, ein eigenes Produkt aus dem Quellcode zu erstellen und dieses dann zu vermarkten, auch wenn es kein Open Source mehr ist. Mehr Freiheit geht nicht.

```
pg_basebackup --checkpoint=fast --write-recovery-conf -D /u02/pgdata/10/DEMOR
pg_ctl -D /u02/pgdata/10/DEMOR start
```

Listing 1

Firmen setzen schon heute ganz bewusst auf Open-Source-Produkte, auch wenn es sich meistens noch um Produkte rund um Linux handelt. Firmen wie Red Hat, SUSE oder Ubuntu haben es geschafft, ein Business-Modell um Open-Source-Technologien zu erstellen und davon leben zu können. Im Datenbank-Umfeld passiert momentan das Gleiche wie mit Linux in den letzten Jahren: Firmen suchen Anbieter, die Services und Produkte um Open-Source-Projekte anbieten und diese dann auch unterstützen.

Heute will kaum einer mehr auf Nicht-Open-Source-Produkte bauen, da die Entscheidungsprozesse nicht nachvollziehbar sind und Lizenz-Modelle sich entweder sehr schnell ändern oder erst gar nicht zu verstehen sind. PostgreSQL ist aus den genannten Gründen perfekt dafür gemacht, weil es keine Einschränkungen gibt. Das würde zwar alles nichts helfen, wenn PostgreSQL keine Features hätte, die in der heutigen Zeit zwingend erforderlich sind. Hat es aber.

PostgreSQL-Features

PostgreSQL bietet nahezu alle Features, die im Enterprise-Umfeld erforderlich sind. Das sind nicht genau die gleichen Features wie bei den Datenbank-Systemen der großen Hersteller, das gilt jedoch auch andersherum. Für 90 Prozent der heutigen Anforderungen bietet PostgreSQL alles, was es braucht – seien es zum Lesen geöffnete Replika-Datenbanken, seien es Backup und Point-in-time-Recovery, parallele Ausführung von SQL über mehrere Cores, Partitionierung, logische Replikation oder die schiere Anzahl von Datentypen *(siehe https://www.postgresql.org/docs/current/static/datatype.html")*. Eine Übersicht aller Features steht unter *„https://www.postgresql.org/about/featurematrix"*. Wichtiger als die Anzahl der Features ist jedoch der generelle Ansatz der PostgreSQL-Entwickler: Alle Features sollen benutzerfreundlich und einfach anzuwenden sein.

```
create table demo ( a int, b text, c timestamp );
with generator as
( select a.*
       , md5(a::text)
       , now()
    from generate_series (1,1000000) a
)
insert into demo
select * from generator;
```

Listing 2

```
copy demo to '/var/tmp/demo.out';
\! vi /var/tmp/demo.out
```

Listing 3

```
create table demo2 (like demo);
\h copy
copy demo2 from '/var/tmp/demo.out';
```

Listing 4

Am deutlichsten wird dieser Ansatz, wenn man sich ansieht, wie viele Schritte es braucht, um eine zum Lesen geöffnete Replika-Datenbank zu erstellen. Davon ausgehend, dass das auf dem gleichen Knoten ausgeführt wird, auf dem die Master-Datenbank läuft, sind exakt drei Schritte notwendig *(siehe Listing 1)*.

Der erste Schritt erstellt ein physisches Abbild der Master-Datenbank in einem neuen Verzeichnis, der zweite Schritt startet die Instanz. Mehr ist nicht notwendig und die Replika ist fertig und fährt alle Änderungen der Master-Datenbank nach. Um die Replika auf einem anderen Server zu erstellen, braucht es nur einen einzigen Schritt mehr: die Konfiguration der Master-Datenbank so anzupassen, dass von einem anderen Server auch eine Verbindung aufgebaut werden kann.

Ein weiteres Beispiel zeigt, wie einfach es ist, Daten in PostgreSQL hinein- beziehungsweise herauszuladen. Man gehe von einer simplen Tabelle mit 1.000.000 Einträgen wie dieser aus *(siehe Listing 2)*. Um diese Daten aus PostgreSQL in eine Datei zu schreiben, ist ein einziger Schritt notwendig *(siehe Listing 3)*. Diese Daten in eine andere Tabelle wieder einzulesen, ist ebenso einfach *(siehe Listing 4)*.

Ein anderes Beispiel sind innovative Daten-Typen wie Range Types. Eine häufige Anforderung von Applikationen besteht darin, einen Gültigkeits-Bereich zu definieren und auch abzufragen. In den meisten Datenbanken wird das über zwei Felder („gültig von" und „gültig bis") und einen Trigger gelöst. Der Trigger vergleicht die Werte, abhängig davon wird der neue Eintrag entweder zurückgewiesen oder erlaubt. In PostgreSQL lässt sich das mit einem Range-Datentyp und sogenannten „Exclusion Constaints" sehr einfach realisieren *(siehe „https://www.postgresql.org/docs/current/static/ddl-constraints.html#DDL-CONSTRAINTS-EXCLUSION")*. *Listing 5* zeigt ein Beispiel für ein einfaches Reservierungs-System für Sitzungszimmer.

Die zweite Tabelle verwendet einen Datentyp „tsrange" (Range von Timestamps) und einen Exclusion Constraint. Diese Kombination erlaubt es nun, Datums-Perioden zu vergleichen, wie es die

```
create table meeting_rooms ( id int primary key
                           , mname varchar(20)
                           , location varchar(10)
                           );
create table meeting_rooms_booked ( mid int references meeting_rooms(id)
                                  , booking_range tsrange
                                  , exclude using gist (mid with =,booking_range with &&)
                                  );
insert into meeting_rooms ( id, mname, location)
       values ( 1, 'meetingsouth', 'south' )
            , ( 2, 'meetingnorth', 'north' )
            , ( 3, 'meetingwest', 'west' )
            , ( 4, 'meetingeast', 'east' );
insert into meeting_rooms_booked ( mid, booking_range )
       values ( 1, '[2018-01-01 15:00, 2018-01-01 18:30]' )
            , ( 1, '[2018-01-01 08:00, 2018-01-01 08:30]' )
            , ( 2, '[2018-03-01 17:00, 2018-03-01 18:30]' )
            , ( 1, '[2018-03-01 05:00, 2018-03-01 08:30]' )
            , ( 3, '[2018-02-01 15:00, 2018-02-01 18:30]' )
            , ( 4, '[2018-02-01 19:00, 2018-02-01 20:30]' )
            , ( 4, '[2018-03-01 15:00, 2018-03-01 18:30]' );
select booking_range && '[2018-02-01 16:00,2018-02-01 16:30)'::tsrange
  from meeting_rooms_booked where mid = 3;
select booking_range && '[2018-02-01 18:45,2018-02-01 19:15)'::tsrange
  from meeting_rooms_booked where mid = 3;
```

Listing 5

```
du -sh $PGHOME
28M    /u01/app/postgres/product/10/db_3
```

Listing 6

beiden Statements am Ende bespielhaft demonstrieren. Das sind nur drei Beispiele und es gäbe noch etliche mehr, doch sie verdeutlichen sehr gut, wie PostgreSQL entwickelt wird: Egal, welches Feature aufgenommen wird, es soll einfach sein, es anzuwenden.

Klein, aber fein

Installiert man PostgreSQL vom Source Code, benötigt man auf der Platte schlussendlich zwischen 25 und 30 MB, je nach PostgreSQL-Version und je nachdem, was alles hineinkompiliert wurde. Das Beispiel in *Listing 6* ist ohne Dokumentation für PostgreSQL 10.3.

Das mag für DBAs, die mit Datenbank-Systemen anderer Hersteller arbeiten, lächerlich aussehen. Fakt ist aber, dass es besonders im Container-Umfeld eine große Rolle spielt. Es macht einen Unterschied, ob Container im Megabyte- oder im Gigabyte-Bereich verteilt werden. Container sind heute überall und es ist keine Überraschung, dass PostgreSQL in diesem Umfeld besonders häufig zum Einsatz kommt. Die Installation ist klein, Lizenz-Probleme gibt es nicht, es ist durch Erweiterungen extrem einfach auszubauen und die Features für die meisten Anwendungsgebiete sind alle vorhanden.

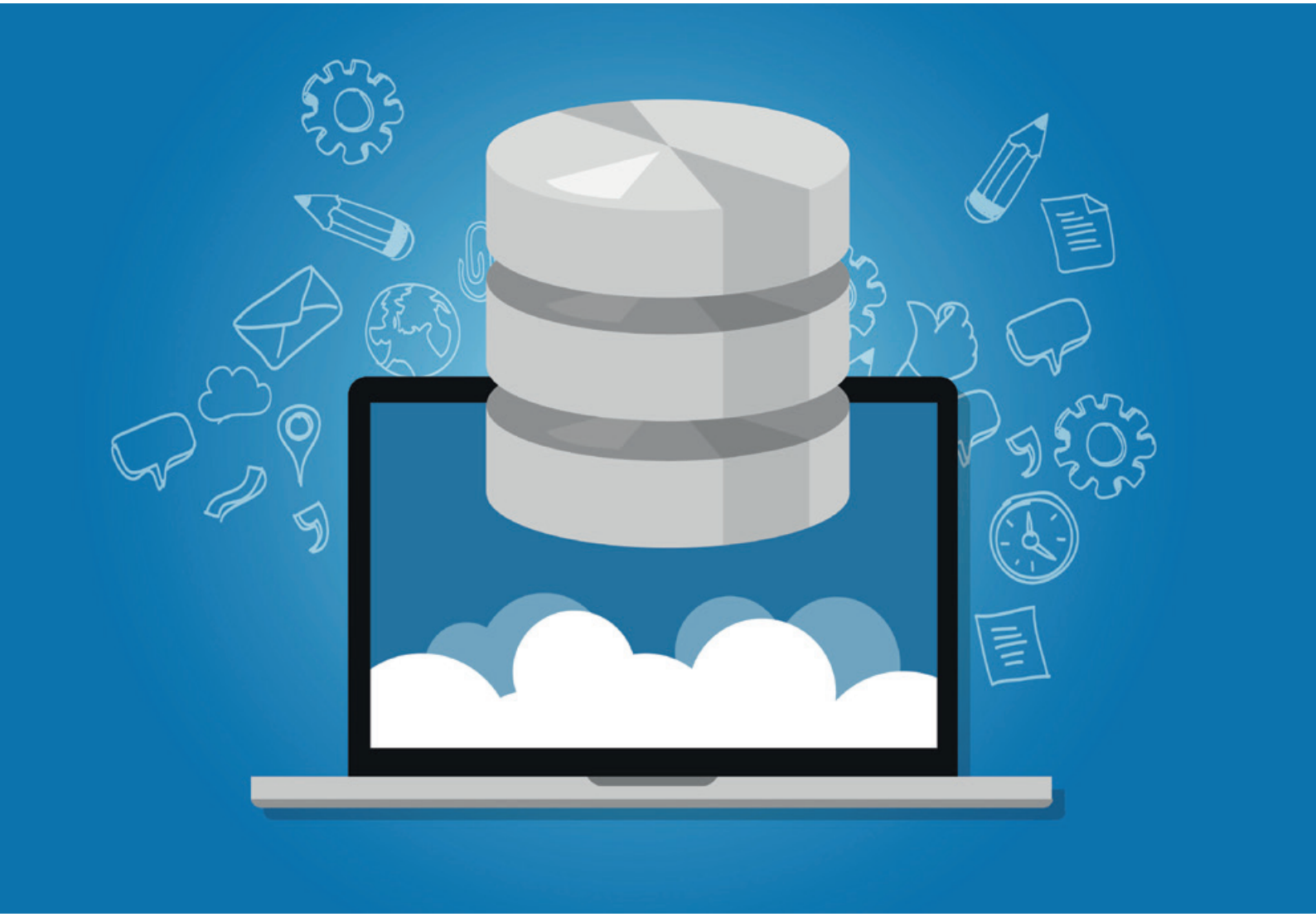
Fazit

Die momentane Richtung ist klar: Open-Source-Produkte kombiniert mit professionellem Support sind das, wo die meisten Firmen heute hinwollen. Das hat allerdings nicht nur Kostengründe, sondern auch viel mit der Qualität der Produkte zu tun. Zumindest bei PostgreSQL wird extrem viel Wert auf sauberen, wiederverwendbaren und vor allem wartbaren Code gelegt. Die Größe der Installation ist ein eindeutiger Beleg dafür. Diese Stabilität und Schlankheit wird heute mehr und mehr geschätzt und bei Container-Deployments auch in Zukunft eine große Rolle spielen.

Selbst, wenn man auf professionellen Support verzichten möchte (aus welchen Gründen auch immer), bietet die PostgreSQL-Community mit ihren Mailing-Listen sehr guten Support *(siehe „https://www.postgresql.org/list“)*. Wer zweifelt, möge sich bitte registrieren und eine Frage stellen.

Die Antwort auf die am Anfang des Artikels gestellte Frage „… und wird das auch so bleiben” lautet eindeutig „ja“. Das gilt nicht nur für PostgreSQL, sondern für alle Community-Projekte, die sich so entwickeln können, wie es die PostgreSQL-Community über die letzten zwanzig Jahre getan hat: Teilhabe einfach gestalten, jährliche Releases, eine aktive Community, immer hilfsbereit, aufgeschlossen und brandaktuell sowie mit einem Code of Conduct *(siehe „https://www.postgresql.org/about/policies/coc“)*. Dieses Projekt wird auch in Zukunft Erfolg haben.

Daniel Westermann
daniel.westermann@dbi-services.com



Oracle, PostgreSQL, Docker und Kubernetes bei der Mobiliar

Hans Eichenberger, Mobiliar Versicherungen, und Daniel Westermann, dbi-services

„Agile Development", „Docker", „Cloud", „DbaaS" – diese und ähnliche Begriffe werden auch bei der Mobiliar Versicherung seit ein paar Jahren immer häufiger genannt. Kann Oracle als strategisches RDBMS-Produkt dies alles abdecken oder gibt es Gründe, auch Open-Source-Datenbank-Systeme genauer anzuschauen?

In der Mobiliar werden bereits diverse Applikationen als Microservices in einem Docker-/Kubernetes-Umfeld entwickelt und betrieben. Vor zwei Jahren hat die Mobiliar-IT eine REST-Schnittstelle für die automatisierte Bereitstellung von Pluggable Databases (PDB) gebaut. Diese erlaubt es dem Unternehmen, während des Deployments einer Applikation bei Bedarf in Kubernetes auch die Datenbank vollautomatisch bereitzustellen. Diese Lösung funktioniert inhouse zuverlässig und erstellt beziehungsweise löscht die PDBs ohne manuelle Eingriffe durch den Datenbank-Administrator.

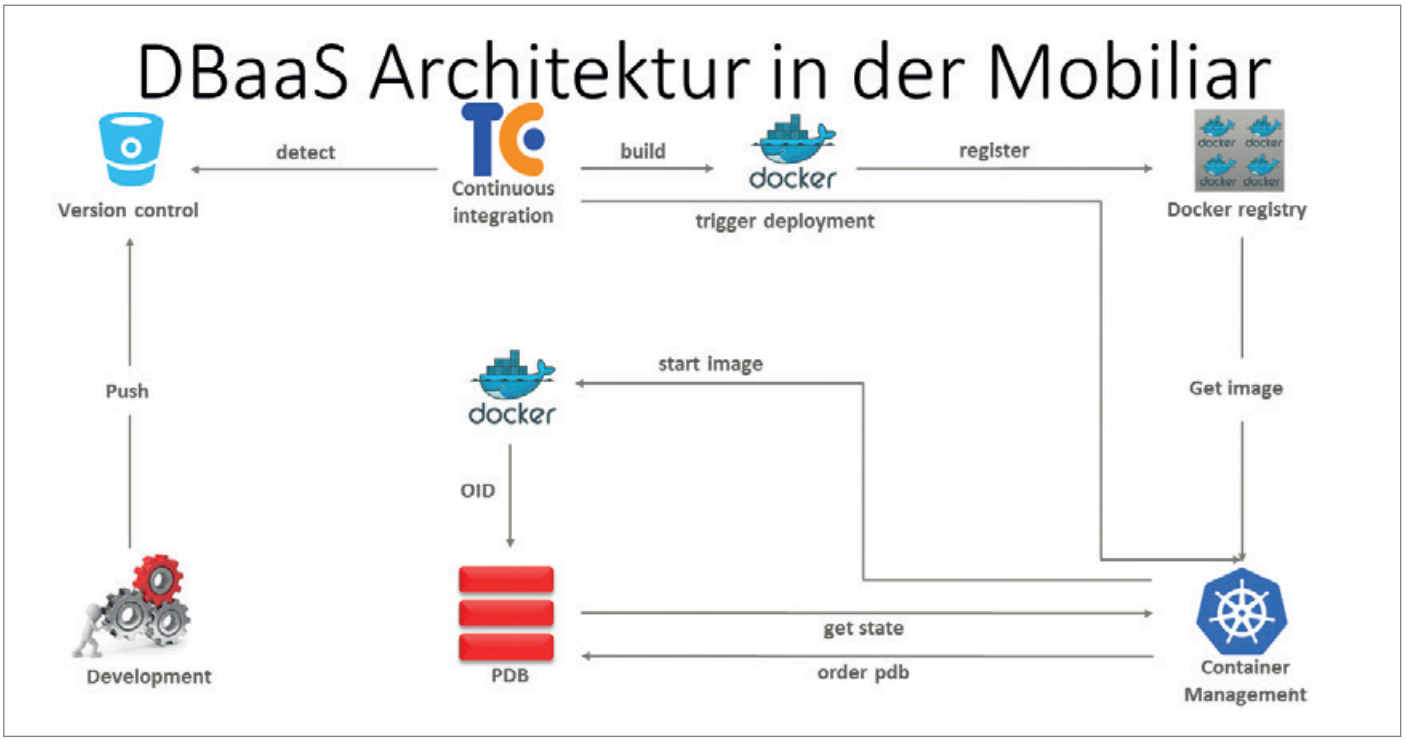


Abbildung 1: Schematische Darstellung der DBaaS-Architektur in der Mobiliar

Auch in der Mobiliar wird die Anforderung immer wichtiger, dass etwa für Test-Installationen sowohl Inhouse- als auch Cloud-Infrastrukturen einsetzbar sein sollen. Die Bedürfnisse der Entwicklung sollen vollumfänglich abgedeckt werden können. Ein zentraler Punkt dabei sind neben der Geschwindigkeit vor allem auch kalkulierbare Kosten. Mit der aktuellen PDB-Order-Schnittstelle erzwingt dieser Punkt, dass beispielsweise für Tests auf den Entwickler-Notebooks nicht eine EE-Installation genutzt, sondern stattdessen die lange nicht mehr aktualisierte XE-Version eingesetzt werden muss.

Technisch gesehen verwenden Microservices meist nur die Grund-Features eines Datenbank-Systems. Damit sind die Persistenz und der schnelle Zugriff auf die Daten sichergestellt. Das Deployment in Kubernetes lässt sich vereinfachen, wenn alle notwendigen Komponenten direkt von Kubernetes gemanagt und keine Schnittstellen zu externen Ressourcen gebaut werden müssen, beispielsweise zum Erstellen von PDBs. Darum musste man eine Lösung mit der Datenbank im Docker-Container suchen. Die folgenden Abschnitte zeigen den Weg von der aktuellen Lösung bis zur Entwicklung der Architektur der PostgreSQL-Docker-Implementierung, wie sie aktuell bei der Mobiliar im Einsatz ist.

Automatisierte PDB-Bereitstellung via REST-Schnittstelle

Damit das agile Entwickeln nicht beim manuellen (sprich langsamen) konventionellen Deployment scheitert, hat die Mobiliar vor etwa zweieinhalb Jahren eine Lösung gesucht, um Datenbanken möglichst automatisiert und schnell bereitstellen zu können. Es stellte sich relativ rasch heraus, dass die Multitenant-Option, die seit Oracle 12.1 verfügbar ist, die Anforderungen an die Datenbanken im agilen Umfeld am besten abdecken kann.

Mit der Modularisierung der Applikation, also dem Schneiden der bestehenden Monolithen in mehrere Microservices, wurde seitens der Architektur definiert, dass Microservice zu Datenbank eine „1:1"-Beziehung darstellen soll. Das bedeutet auf der einen Seite eine Entflechtung der Schnittstellen auf Datenbank-Ebene; auf der anderen Seite steigt allerdings die Zahl der notwendigen Datenbanken stark an. Mit der Multitenant-Option kann diese Anforderung abgedeckt werden, ohne die Server-Ressourcen aufstocken zu müssen.

Wie eine solche Datenbank auszusehen hat, war also klar. Jetzt ging es noch darum, diese quasi in „no time" zur Verfügung zu stellen. Als Lösungsansatz baute man eine Container-Datenbank (CDB) und erstellte darin eine Template-PDB, die bereits alle standardmäßigen Komponenten enthält (Applikations-Tablespace, Applikations-User-Rolle etc.).

Via REST-Schnittstelle wird auf eine zentrale Verwaltungs-Datenbank verbunden und eine Funktion aufgerufen, die im Hintergrund per Datenbank-Link auf die entsprechende CDB verbindet. Dort ruft sie mit „scheduled-job" die Host-Skripte „clone_pdb_4_app", „add_oid_entry" und „register_pdb_in_oem" auf. Die aufrufende Schnittstelle erhält zum Schluss das Resultat, also die Meldung, ob die PDB fehlerfrei angelegt werden konnte, beziehungsweise den aufgetretenen Fehler. Die *Abbildung 1* zeigt, wie die automatische PDB-Erstellung in den Build-/Deployment-Prozess integriert wurde.

Oracle oder PostgreSQL für Docker Kubernetes

Um diese Frage beantworten zu können, hat man sich überlegt, was die wichtigsten Vorgaben für den Betrieb von Datenbanken in einem Docker-Container sind, der von Kubernetes gemanagt wird:

- Sicherstellung der Persistenz (Backup/Recovery, Desaster-fähig)

```
ENV PG_MAJOR=10 \
PG_VERSION=10.4 \
PG_SHA256="1b60812310bd5756c62d93a9f93de8c28ea63b0df254f428cd1cf1a4d9020048" \
PGDATA=/u02/pgdata \
PGDATABASE="" \
PGUSERNAME="" \
PGPASSWORD="" \
REGBACKUP="yes" \
LANG=en_US.utf8
```

Listing 1

- Kalkulierbare Kosten unabhängig von der eingesetzten Infrastruktur
- Performant und skalierbar
- Support durch Community und Spezialisten
- Security-Vorgaben müssen eingehalten werden können
- Möglichst kleines Docker-Image
- Entwicklung/Test/Produktion mit identischer Software

Zwei Punkte sind mit den aktuellen Oracle RDBMS unter den aktuell gültigen Lizenzbedingungen nur schwierig zu erreichen: kalkulierbare Kosten und die Größe des Docker-Image. Das gilt auch für die anderen etablierten Datenbank-Systeme, die bei der Mobiliar eingesetzt werden. Einzig PostgreSQL deckt die geforderten Punkte alle ab.

Nach mehreren Diskussionen mit den Entwicklern, die PostgreSQL als RDBMS-Alternative zu Oracle bevorzugen, wurde vom DBA-Team ein PoC mit PostgreSQL im Docker-/Kubernetes-Umfeld gestartet, in enger Zusammenarbeit mit dem Container-Solutions-Team und mit externer Unterstützung durch Daniel Westermann von dbi.

Entwicklung der Architektur

Die Entwicklung einer neuen Architektur ist immer eine Herausforderung – aber irgendwo muss man halt einfach mal anfangen. Ausgangspunkt war die nachfolgende Liste, die in einem Brainstorming zusammengetragen wurde:

- Datenbank-Name (Input)
- User-/Schema-Name und Passwort (Input)
- Extensions
- Parameter
- Backup ja/nein (abhängig vom Backup-Konzept)
- Public Schema Permissions
- Backup point in time
- Init/Upgrade, alles da
- Monitoring User
- Ad User und Rollen

Das sieht ziemlich simpel aus, fasst aber schon quasi alles zusammen, über was man sich Gedanken machen muss:

- Soll der Name der Datenbank als Parameter in den Container gegeben werden?
- Sollen Benutzername und Passwort als Parameter in den Container gegeben werden?
- Wie behandelt man die PostgreSQL-Parameter? Soll es ein Standard-Set für alle geben?
- Sind verschiedene Templates für verschiedene Anforderungen notwendig?
- Wie macht man in der Container-Welt ein Backup und vor allem einen Restore der Instanz? Braucht man das überhaupt?
- Was ist mit PITR? Wohin will man archivieren?
- Was macht man mit Minor- und Major-Versionsupgrades von PostgreSQL?
- Soll das automatisch laufen?
- Wie überwacht man in der Container-Welt?
- Verwendet man einen vorgefertigten PostgreSQL-Container oder baut man ihn selbst?

Die Entscheidung, den PostgreSQL-Container selbst zu bauen, wurde aus unterschiedlichen Gründen sehr schnell getroffen: Einerseits war wichtig, die volle Kontrolle über das Container-Image zu haben, damit nicht benötigte Optionen auch nicht in den Container kommen. Andererseits soll das Container-Image so klein wie möglich sein und man wollte vor allem den Upgrade-Prozess selbst nach eigenen Vorstellungen steuern können.

Da man PostgreSQL problemlos selbst vom Source-Code kompilieren kann, war das initiale Container-Image schnell gebaut. Die vereinfachte Prozedur dazu ist:

- Installation der benötigten Betriebssystem-Pakete
- Download des PostgreSQL-Source-Codes
- Makefile erzeugen
- Kompilieren und installieren
- Deinstallation aller Pakete, die es nicht mehr braucht, um das Image klein zu halten

Zusammengefasst gehen folgende Variablen in den Container *(siehe Listing 1)*. Sobald der Container startet, werden sie verarbeitet und bestimmen die Version von PostgreSQL, den Ort, an dem die Datenbank-Dateien gespeichert werden, den Namen der Datenbank, den Benutzernamen und das zugehörige Passwort. Zuletzt gibt es noch die „REGBACKUP"-Variable, die steuert, ob für diesen Container Datenbank-Backups erstellt werden sollen. Da ein PostgreSQL-Minor-Upgrade nichts anderes bedeutet, als die neuen Binaries zu installieren und die Instanz mit diesen zu starten, ist dieses Thema schon erledigt: Sobald ein neues PostgreSQL-Minor-Release herauskommt, baut man den Container mit diesem Release neu. Sobald ein Datenbank-Container neu startet, wird er die neue Version des Image anziehen und der Minor-Upgrade ist erledigt.

Major-Version-Upgrades gestalten sich schwieriger: Sobald ein neues Major-Release zur Verfügung steht, wird das entsprechende Container-Image mit dieser Version gebaut. Zusätzlich enthält dieses Image aber auch das letzte Minor-Release der Vorgänger-Version. Somit lässt sich beim Starten des Containers prüfen, ob ein neues Major-Release zur Verfügung steht und der Upgrade dann automatisiert durchgeführt wird.

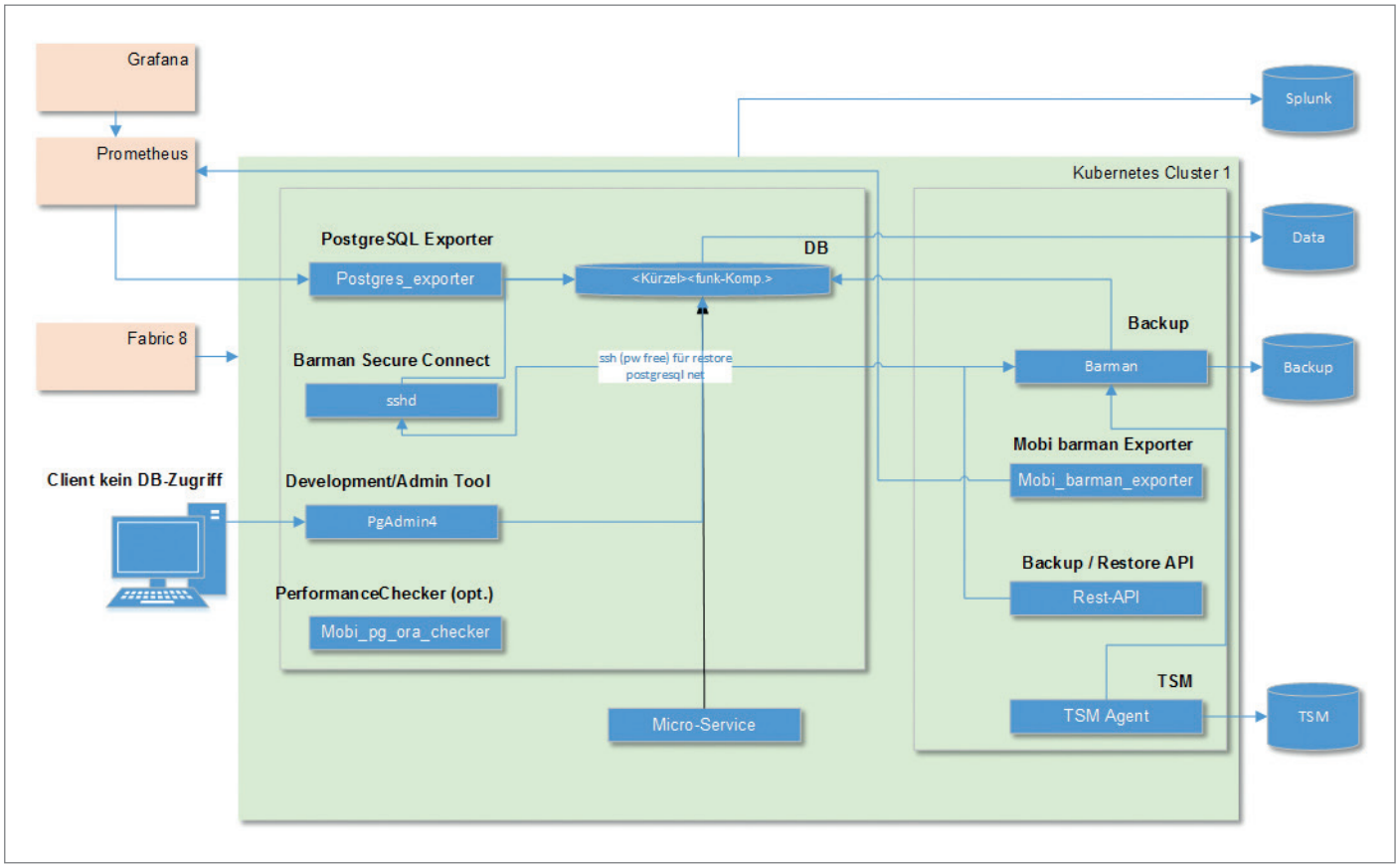


Abbildung 2: Die neue Architektur

Das führt zum nächsten Punkt: Wie will man die Instanzen sichern und wiederherstellen? Da voll auf Open-Source gesetzt wird, war man sich ziemlich schnell einig, Barman zu verwenden. Barman ist ein Community-Produkt, das in der PostgreSQL-Welt sehr verbreitet und beliebt ist. Die Lösung besteht darin, einen eigenen Barman-Container pro Kubernetes-Cluster laufen zu lassen, der dann auf alle Datenbank-Container Zugriff hat, und so ein zentrales Backup-Repository aufzubauen.

Zusätzlich brauchte es noch etwas, womit die Business-Analysten einfach Abfragen gegen die Datenbank machen können. Ein eigener „pgadmin4"-Container ist dafür vollständig ausreichend. Darum wurde pro Datenbank-Deployment ein „pgadmin4"-Container zur Verfügung gestellt.

Da im Docker-Umfeld Log-Meldungen nach „stdout" gehen, lag es auf der Hand, diese von Splunk abgreifen zu lassen. Somit werden alle Log-Meldungen zentral von Splunk verwaltet und auch das Alerting ist sichergestellt.

Blieb das Thema „Monitoring": Das Produkt, das im Kubernetes-Umfeld zum Einsatz kommt, ist Prometheus. Hier musste man nichts neu erfinden, sondern lediglich dafür sorgen, dass Metriken aus PostgreSQL heraus geliefert werden können, die dann von Prometheus weiterverarbeitet werden. Darauf aufbauend kommt dann Grafana zum Einsatz zur Visualisierung der Daten. Die komplette heute eingesetzte Architektur sieht wie in *Abbildung 2* aus.

Fazit

Auch wenn das Oracle-RDMS in Docker betrieben werden kann, liegen die Vorteile doch klar bei Open-Source-Lösungen wie PostgreSQL. Nicht nur ist das Produkt ausgereift, auch das Open-Source-Modell überzeugt, sodass der fehlende Hersteller-Support nicht als Show-Stopper angesehen werden muss. Die besonderen Herausforderungen im Docker-Umfeld wie ein performantes Storage-System sind unabhängig vom eingesetzten Datenbank-System zu lösen.

Hans Eichenberger
hans.eichenberger@mobi.ch

Daniel Westermann
daniel.westermann@dbi-services.com

HDFS vs. NoSQL vs. RDBMS – welcher Datastore für welches Projekt?

Dr. Nadine Schöne, ORACLE Deutschland BV & Co. KG, und Enno Schulte, virtual7 GmbH

Nach welchen Kriterien wählt man den passenden Datastore für ein Projekt aus? Die Auswahl ist groß: Hadoop Distributed Filesystem (HDFS), NoSQL-Datenbanken oder doch lieber eine klassische relationale Datenbank (RDBMS)? Der Artikel zeigt, wie schnell sich Daten in welchem Datastore konsistent abspeichern lassen und wie schnell man wieder auf sie zugreifen kann. Zudem wird die Flexibilität beim Datenschema betrachtet.

Um Informationen aus Rohdaten zu gewinnen, müssen diese gesammelt, vorverarbeitet und analysiert werden. Bei der Batch-Datenverarbeitung werden die Daten erst eingelesen („Ingest"), dann gespeichert und später ausgelesen („Read", siehe *Abbildung 1*). Je nach Anwendungsfall sind hier unterschiedliche Kriterien wichtig, etwa Schreibgeschwindigkeit, Vorverarbeitung, Lesegeschwindigkeit, Speicherpreis, Konsistenz, Redundanz, Zugriffsbeschränkungen, Flexibilität beim

Abbildung 1: Daten werden eingelesen („Ingest"), gespeichert und dann wieder ausgelesen („Read"). Je nach Datastore erfolgen bestimmte Verarbeitungsschritte beim Einlesen oder beim Auslesen der Daten, was die Schnelligkeit beeinflusst, mit der auf dem Datastore geschrieben beziehungsweise gelesen wird. Bei einem Vergleich von Datastores sind also sowohl die Schreib- als auch die Lesegeschwindigkeit zu beachten.

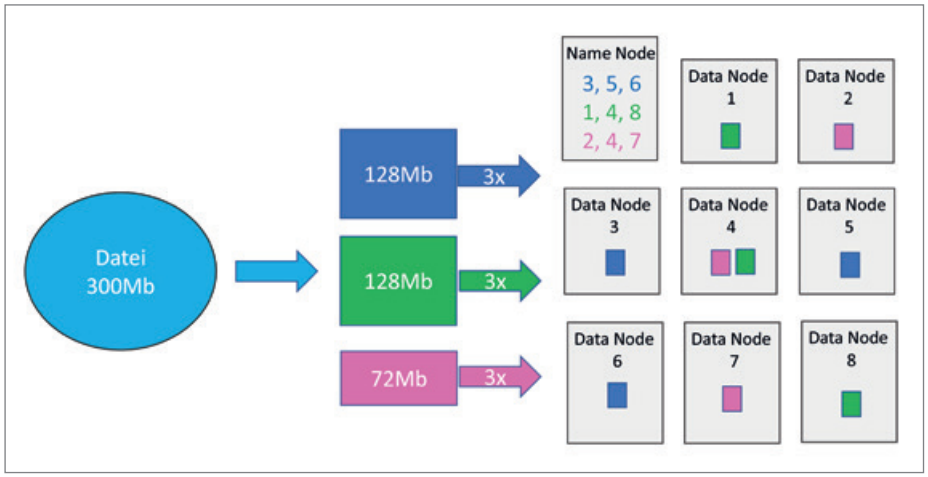


Abbildung 2: Beim Speichern einer Datei in HDFS wird diese in Blöcke geteilt, die dann redundant auf den Data Nodes des Clusters abgelegt sind. Alle Informationen, die zum Zugriff auf die Datei benötigt werden, sind im Name Node des Clusters gespeichert.

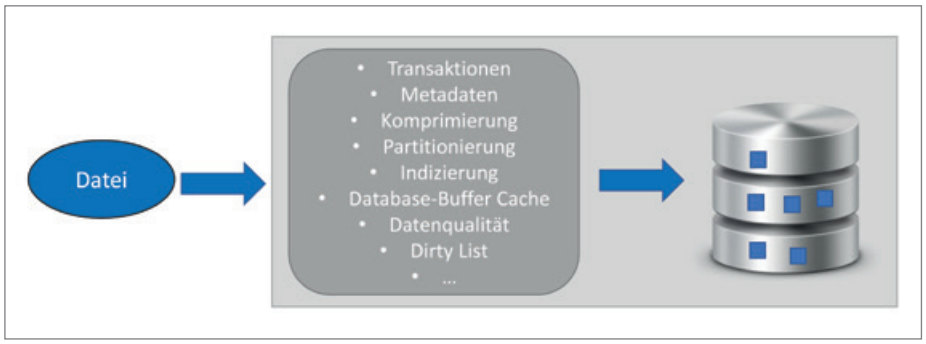


Abbildung 3: Das Speichern einer Datei in der Oracle-Datenbank: Die Datei wird in der Datenbank-Instanz vorverarbeitet und dann in Form von optimierten Database Files in der Datenbank gespeichert.

Datenschema oder auch die Möglichkeit, die Daten im Speicher zu verändern.

Um aus Rohdaten Erkenntnisse zu gewinnen, müssen sie normalerweise vorverarbeitet werden. Diese Vorverarbeitung kann beim „Ingest" erfolgen, also vor dem Schreiben in den Speicher, sodass die bereinigten Daten später direkt nach dem Auslesen analysiert werden können. Generell ist damit der „Ingest" langsamer, der „Read" dafür aber schneller und der zusätzliche Aufwand nach dem „Read" kleiner. Ist ein schneller „Ingest" besonders wichtig, bietet es sich an, die Rohdaten nur minimal vorzuverarbeiten oder ohne Vorverarbeitung zu speichern. Man muss sich damit auch nicht vor dem Speichern für eine bestimmte Vorverarbeitung entscheiden, sondern kann die Daten nach dem Auslesen für unterschiedliche Analysen vorverarbeiten.

Die Eigenschaften der Datastores

Der Artikel ist auf den Vergleich von Datei-basierten, relationalen und Key-Value-Datenspeicher beschränkt. Als Beispiel für einen Datei-basierten Speicher dient das Hadoop Distributed Filesystem (HDFS). Als typische relationale Datenbank wird die Oracle-Datenbank betrachtet und der Key-Value-Store ist mit der Oracle-NoSQL-Datenbank abgebildet.

Das Hadoop Distributed Filesystem ist dafür ausgelegt, große Datenmengen zu verarbeiten. Es „lebt" auf einem Cluster, der aus simpler Hardware besteht; damit ist horizontale Skalierbarkeit gegeben. Die Daten werden redundant (in der Regel in dreifacher Kopie) gespeichert. HDFS ist – wie der Name schon sagt – ein Filesystem. Dateien werden beim Ingest nicht vorverarbeitet, sondern gesplittet und als redundante Blöcke („Data Chunks") definierter Größe gespeichert *(siehe Abbildung 2)*. Damit ist der Ingest sehr schnell und man erreicht hohe Schreibraten für große Dateien, da parallel geschrieben werden kann.

Ein Schreibvorgang ist erst dann erfolgreich, wenn die gesamte Datei geschrieben wurde („synchronous write"), also alle redundanten Data Chunks. Eine Voraussetzung für die Schnelligkeit des Ingest ist jedoch, dass alle Knoten des Hadoop-Clusters im selben Datacenter lokalisiert sind – damit ist vertikale Skalierbarkeit nicht gegeben.

Beim Zugriff auf die Daten kann man keinen Zugriffspfad auswählen, sondern man muss die gesamte Datei lesen. Damit ist der Read (im Gegensatz zum Ingest) langsamer. Zusätzlich müssen sämtliche Vorverarbeitungsschritte vor der Daten-Analyse jedes Mal nach dem Read erfolgen, da Daten beim Ingest nicht vorverarbeitet werden. Wenn regelmäßig auf die Daten zugegriffen wird, kann dies unnötigen Overhead erzeugen.

Eine Modifikation der gespeicherten Daten (Veränderungen, Updates) ist nicht erlaubt, da für jede Änderung alle betroffenen Blöcke komplett neu geschrieben werden müssten. Deshalb sind Änderungen sehr teuer. Eine gewünschte

Änderung erfordert das Neuschreiben aller Kopien des betroffenen Data Chunk. Bei einer Standardgröße von je 128 MB und redundanter Speicherung in drei Kopien würde somit eine gewünschte Änderung von beispielsweise 8 MB insgesamt eine nötige Änderung von 384 MB (3*128 MB) erfordern.

Zusammengefasst eignet sich HDFS als Datenspeicher besonders gut, wenn große Mengen großer Dateien gespeichert werden sollen und die weitere Verarbeitung dieser Daten auf unterschiedliche Arten oder nicht regelmäßig erfolgt.

Die Oracle-Datenbank ist ein Beispiel für ein Relational Database Management System (RDBMS). Dort sind Compute (Instanz) und Storage (Datenbank) getrennt. Daten werden erst einmal in einem einzelnen Datacenter gespeichert. Hochverfügbarkeit kann durch Nutzung von Oracle Real Application Clusters (RAC) – mehrere Datenbank-Instanzen greifen auf denselben Datenspeicher zu – gewährleistet werden. Um vertikale Skalierbarkeit zu erreichen, müssen die Daten in einem weiteren Schritt repliziert werden – mit Anwendungen wie Oracle GoldenGate kann dies in Echtzeit erfolgen. Um Applikationen elastisch skalierbar zu machen, lässt sich Sharding nutzen, sodass für die Applikation ein Pool von Datenbanken wie eine einzelne Datenbank erscheint.

Der Ingest erfolgt in Form von Transaktionen; voneinander abhängige Änderungen werden also zusammengefasst (entweder werden alle Änderungen durchgeführt oder keine) und optimiert ausgeführt. Die Daten werden in der Instanz validiert und vorverarbeitet. Hierbei erfolgen unter anderem eine Indizierung der Daten zur optimierten Suche, das Speichern von Metadaten, die Anwendung von Datenschemata, Partitionierung und Komprimierung. Die aufbereiteten Daten werden dann in mehreren Kopien physikalisch als optimierte Database Files gespeichert *(siehe Abbildung 3)*.

Beim Read werden die zur Beantwortung der Anfrage benötigten Datenbank-Blöcke in den Database-Buffer-Cache der Instanz geladen, falls sie dort noch nicht vorgehalten werden. Es wird ein Zugriffspfad auf die angefragten Daten ausgewählt, es muss also nicht wie bei HDFS der gesamte Datensatz eingelesen werden. Damit ist der Read schnell und konsistent. Die Nutzung von Optionen wie

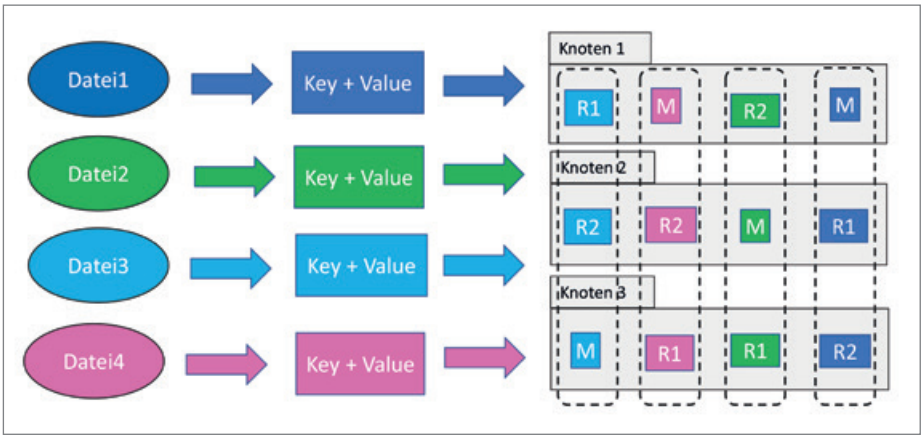


Abbildung 4: Beispiel der Datenspeicherung auf einem Oracle-NoSQL-Datenbank-Cluster mit drei Knoten und einem Replikationsfaktor von drei. Aus einer Datei werden Keys extrahiert und diese zusammen mit der Datei selbst („Value") auf einem der Knoten gespeichert (M). Von diesem Master werden dann Replikas auf anderen Knoten geschrieben.

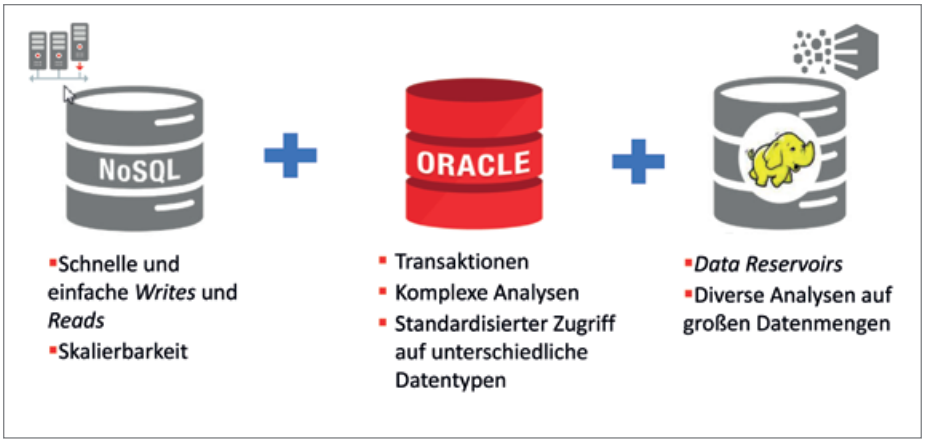


Abbildung 5: Komplementäre Datastores: Die in diesem Artikel betrachteten Datenspeicher decken zusammen den Großteil der gängigen Anforderungen ab.

Partitioning, In-Memory etc. kann das Lesen der Daten weiter beschleunigen.

Spätere Änderungen an den Daten sind möglich. Sie werden erst einmal in der Instanz vollzogen – der Database-Buffer-Cache ist also immer aktuell – und später gesammelt in die Datenbank geschrieben. Typischerweise ist die Oracle-Datenbank beim Ingest langsamer als HDFS oder Oracle NoSQL. Durch die Vorverarbeitung, Indizierung und das Vorhalten von Daten im Database-Buffer-Cache ist der Read jedoch sehr schnell.

Der Begriff „NoSQL" ist eine Abkürzung für „Not-only SQL", der Begriff „NoSQL-Datenbank" beschreibt also eine Vielfalt von Datenspeichern mit unterschiedlichen Strukturen, die für unterschiedlichste Anforderungen entwickelt wurden. In diesem Artikel wird die Oracle-NoSQL-Datenbank (Oracle NoSQL) als Beispiel für einen Key-Value-Store betrachten. Sie basiert auf der Oracle-Berkeley-Datenbank, Java-Edition „High Availability".

Oracle NoSQL wird auf einem Cluster installiert, der mehrere Datacenter überspannen kann. Damit sind sowohl horizontale als auch vertikale Skalierbarkeit gegeben. Die Daten werden redundant (in der Regel in dreifacher Kopie) gespeichert. Der Schreibvorgang ist asynchron und damit „eventual consistent", es wird in der Regel also zeitversetzt auf Master- und Replika-Knoten geschrieben („asynchronous write").

Der Artikel beschränkt sich auf Key-Value-Stores. Diese speichern beliebige Daten („Values") zusammen mit einem oder mehreren Schlüsseln („primary" und „secondary keys"). Die Vorverarbeitung der Daten beim Ingest ist damit minimal – aus den Rohdaten werden die Keys extrahiert und dann zusammen mit den Daten als ein Datensatz gespeichert *(siehe Abbil-*

dung 4). Dabei werden erst eine Kopie auf den Master- und dann weitere Kopien auf den Replika-Knoten geschrieben. Der Ingest ist deshalb sehr schnell. Key-Value-Stores eignen sich besonders gut für das Speichern großer Mengen kleiner Dateien wie beispielsweise Sensordaten.

Beim Read werden die Daten parallel von Master- und Replika-Knoten gelesen. Für komplexe Abfragen ist ein Key-Value-Store im Vergleich mit anderen Datastores am schnellsten, wenn die Keys sinnvoll und passend gewählt wurden. Da die Daten zeitversetzt auf Master- und Replika-Knoten geschrieben werden, kann jedoch dieselbe Abfrage unter Umständen unterschiedliche Ergebnisse liefern, wenn der Ingest noch nicht abgeschlossen ist.

Zusammengefasst sind Key-Value-Stores optimiert für das Speichern von großen Mengen kleiner Dateien, deren Weiterverarbeitung schon soweit definiert ist, dass die Keys zur Suche im Datastore geschickt gewählt werden können.

Ein auf den ersten Blick ähnliches Konzept wie Key-Value-Stores verfolgt der Object Storage. Hier werden Daten zusammen mit einer beliebigen Menge von Metadaten und einem eindeutigen Identifier gespeichert. Object Storage ist jedoch für das Speichern von großen Mengen großer Dateien optimiert. Auch die Konsistenz ist schwächer als bei Key-Value-Stores.

Wie Daten geschrieben werden

Je weniger mit den Daten gemacht wird, wenn sie geschrieben werden, desto schneller ist der Ingest. Für große Dateien ist HDFS somit der schnellste Datastore, da durch die Aufteilung in Blöcke parallel geschrieben werden kann. Sollen viele kleine Dateien geschrieben werden, liegt ein Key-Value-Store wie die Oracle-NoSQL-Datenbank vorn. Aufgrund der Vorverarbeitungsschritte ist der Ingest in die Oracle-Datenbank im Schnitt langsamer als in HDFS oder in die Oracle-NoSQL-Datenbank.

Wie Daten gelesen werden

Beim Lesen der Daten sind vor allem zwei Dinge wichtig: Lesegeschwindigkeit und Read Consistency. Man möchte schnell auf die gespeicherten Daten zugreifen und identische Lesezugriffe sollen auch zu identischem Output führen. Die schnellsten Antworten auf Abfragen liefert ein Key-Value-Store – sofern die definierten Keys geschickt gewählt wurden. Dieselbe Abfrage kann allerdings unter Umständen unterschiedliche Ergebnisse liefern, wenn der Ingest der abgefragten Daten noch nicht abgeschlossen ist.

Die Oracle-Datenbank ist sehr schnell bei strukturierten Daten, bei denen beispielsweise Partitionierung und Indizierung genutzt werden können. Zur Beantwortung von Abfragen muss dann nur ein Teil der Daten gelesen werden. Außerdem werden oft abgefragte Datenblöcke im Hauptspeicher der Instanz im Database-Buffer-Cache vorgehalten, was die Abfragen weiter beschleunigt. Um Daten aus HDFS abzufragen, muss der gesamte ursprüngliche Datensatz gelesen werden. Damit ist HDFS im Vergleich der langsamste Datenspeicher.

Zusammenfassung und Fazit

Die drei hier betrachteten Datenspeicher sind jeweils für unterschiedliche Anforderungen besonders geeignet, in Kombination decken sie die meisten Anforderungen ab *(siehe Abbildung 5)*. Um insbesondere Rohdaten, viele große Dateien oder selten genutzte Daten zu speichern, bietet sich HDFS an. Der Ingest ist sehr schnell und man behält sich die Flexibilität bei der Weiterverarbeitung bis nach dem Read vor. Anwendungsfälle sind zum Beispiel das Sammeln von Social-Media-Daten oder Active Archiving.

Für das Speichern vieler kleiner Dateien in Rohdatenform wie Sensordaten sind Key-Value-Stores wie die Oracle-NoSQL-Datenbank eine gute Lösung. Sowohl Ingest als auch Read sind sehr schnell.

Für strukturierte Daten, die wiederverwendbar gespeichert und mit etablierten Prozessen ausgewertet werden sollen, ist eine RDBMS wie die Oracle-Datenbank weiterhin der Datenspeicher der Wahl. Sie ist für das Speichern von relationalen und objektrelationalen Daten ausgelegt. Die Vorverarbeitung der Daten erfolgt vor dem Speichern, der Ingest ist deshalb langsamer. Dafür stehen die Daten dann aber aufbereitet und konsistent für die weitere Verarbeitung zur Verfügung.

Oft ist eine Kombination verschiedener Datenspeicher sinnvoll. Manchmal sollen die Daten auf unterschiedliche Arten verarbeitet werden, die Daten müssen also unterschiedlich vorverarbeitet werden. Mitunter ist ein Teil dieser Verarbeitungsweisen beim Abspeichern noch nicht definiert. Hier bietet es sich an, Rohdaten oder minimal vorverarbeitete Daten in HDFS oder in einem Key-Value-Store zu speichern. Dies liefert die nötige Flexibilität bei der späteren Aufbereitung der Daten. Gleichzeitig werden die Daten aber auch bereits vorverarbeitet in einer Oracle-Datenbank gespeichert, sodass für Reports etc. der Aufwand der Datenaufbereitung nur einmal betrieben werden muss und nicht bei jeder einzelnen Abfrage.

Hinweis: Die Autoren haben sich für eine lesefreundliche (und damit gegen eine gendergerechte) Schreibweise entschlossen. In diesem Artikel verwenden sie nur die männliche Form – die weibliche Form ist selbstverständlich immer miteingeschlossen.

Dr. Nadine Schöne
nadine.schoene@oracle.com

Enno Schulte
enno.schulte@virtual7.de

ORACLE® auf vmware®

Yvonne Murphy, Vmware

Speicher war schon immer ein kritischer Faktor für die Leistung einer E/A-intensiven Oracle-Datenbank. Der Artikel bringt eine kurze Einführung zum Thema „Oracle unter VMware“ und gibt Hinweise auf einige grundlegende Funktionen in vSphere, die für Oracle problematisch sein können. Darüber hinaus gibt er einen Überblick darüber, wie E/As in einer virtuellen VMware-Umgebung behandelt werden. VMware vSphere bietet eine Vielzahl von Speicher-Optionen und kann einer Oracle-Datenbank als Grundlage dienen, um die Anforderungen jedes Unternehmens zu erfüllen. Zudem werden auch einige Best Practices für die Konfiguration einer Virtual Machine und ESXi-Hosts zur optimalen Unterstützung der Oracle-Datenbank/-Anwendungen vorgestellt.

Die Autorin arbeitet in der VMware-Support-Services-Abteilung in Cork, Irland, und ist auf Oracle spezialisiert. Hat ein Kunde ein Problem, von dem der Oracle-Support denkt, dass es auf den VMware-Technologie-Stack zurückzuführen ist, wendet er sich an VMware und wird ihrem Team zugewiesen. Dieses arbeitet dann mit Oracle zusammen, um das Problem zu beheben. Oracle-Kunden sind eine wichtige Komponente im Geschäft von VMware, sodass jedem Oracle-Fall höchste Priorität eingeräumt wird. Deshalb ist die Autorin in einem Team mit Experten aus anderen Bereichen tätig, auf deren Expertise sie bei Bedarf zurückgreifen kann. Da sie zuvor für zehn Jahre als Oracle DBA beschäftigt war, besitzt sie ein gutes Verständnis für Oracle und VMware sowie dafür, wie beide Produkte optimal zusammenarbeiten.

VMware vSphere

Die VMware-Virtualisierungslösung kann mehrere virtuelle Maschinen auf einer einzigen physischen Maschine betreiben. Dabei teilen sich alle virtuellen Maschinen die Ressourcen dieses einen physischen Servers, diese VMs können dann verschiedene Betriebssysteme und mehrere Anwendungen gleichzeitig ausführen. VMware war übrigens das erste kommerzielle Unternehmen, das die x86-Architektur virtualisiert hat, und ist auch weiterhin Marktführer in diesem Bereich.

Während ihrer Zeit als Oracle DBA hat die Autorin die Zertifizierung in OCA und OCP erfolgreich abgeschlossen, hatte aber immer den Wunsch, auch OCM zu werden. In einer physischen Umgebung und als Produktions-DBA gab es nie genug Zeit oder Ressourcen, um die für diesen Abschluss erforderlichen Tests durchzuführen. Dank der VMware-Funktionen „Klonen“ und „Snapshots“ war die Vorbereitung darauf wesentlich einfacher. „Klonen“ ist der Prozess, um eine Kopie einer virtuellen Maschine zu erstellen. Dies hat den entscheidenden Vorteil, dass ein Betriebssystem nur einmal installiert und eingerichtet werden muss.

„Snapshots“ sind ein Verfahren, um den Zustand der virtuellen Maschinen (einschließlich aller Daten) zu einem bestimmten Zeitpunkt zu speichern. Dies ist besonders beim Testen von DR-Szenarien nützlich, denn wenn eine vollständige Wiederherstellung fehlschlägt, kann man jederzeit einfach zu einem vorherigen Snapshot zurückkehren und alle Änderungen rückgängig machen.

„Snapshots“ ist eine der beliebtesten Funktionen für VMware-Anwender. In ei-

ner Produktionsumgebung sollte man dieses Tool jedoch vorsichtig einsetzen. Alle Änderungen werden in eine Delta-Datei geschrieben und alle Lesevorgänge gehen dann sowohl durch dieses Delta als auch durch die Basisdatei. Dies erhöht die E/As, die von der Festplatte geschrieben und gelesen werden müssen, was sich negativ auf die Performance in einer produktiven Oracle-Datenbank auswirken kann.

Die Delta-Datei wird umso größer, je länger der Snapshot behalten wird. Sobald ein Snapshot entfernt wird, ist diese Delta-Datei zu konsolidieren beziehungsweise zusammenzuführen. Dies kann zu einer kurzfristigen Unterbrechung führen, die virtuelle Maschine reagiert also nicht mehr und der VM-Takt wird gestoppt. Während dieser Zeit sind keine Verbindungen zu Oracle möglich. Im Extremfall kann das dazu führen, dass die Oracle-Datenbank seine SGA löscht. Dies wiederum wirkt sich nach einem erneuten Verbindungsaufbau negativ auf die Performance aus. Die VMware-Entwickler haben hier in vSphere 6.5 und 6.7 entscheidende Vorschritte gemacht, sodass solche Probleme schon seit Längerem nicht mehr beobachtet wurden.

Mit jedem Release wuchs auch die Anzahl der maximal unterstützten Konfigurationen. Eine vollständige Liste mit den Maximalwerten steht auf „vmware. com“ beziehungsweise wenn man nach „VMware Configuration Maximums“ sucht *(siehe Abbildung 1)*.

Maximum Type	Maximum	Recommended Maximum value	Description
Virtual Machine Maximums			
Compute	Virtual CPUs per virtual machine (Virtual SMP)	128	
Memory	RAM per virtual machine	6128 GB	
Memory	Virtual machine swap file size	6128 GB	VMFS3 with 1 MB block maximum swap size is 255 GB. Recommended solution is VMFS5, not VMFS3 with bigger block size
Storage Virtual Adapters and Devices	Virtual SCSI adapters per virtual machine	4	
Storage Virtual Adapters and Devices	Virtual SCSI Targets Per Virtual SCSI Adapter (PVSCSI only)	64	Limit applicable to PVSCSI only. Any combination of disk or VMDirectPath SCSI target.
Storage Virtual Adapters and Devices	Virtual SCSI Targets Per Virtual Machine (PVSCSI only)	256	Limit applicable to PVSCSI only.
Storage Virtual Adapters and Devices	Virtual NVMe adapters per virtual machine	4	
Storage Virtual Adapters and Devices	Virtual NVMe targets per virtual NVMe adapter	15	
Storage Virtual Adapters and Devices	Virtual NVMe targets per virtual machine	60	
Storage Virtual Adapters and Devices	Virtual disk size	62 TB	
Storage Virtual Adapters and Devices	IDE controllers per virtual machine	1	Supports two channels (primary and secondary) each with a master and slave device.

Abbildung 1: VMware-Konfigurationsmaxima für eine VM in vSphere 6.7

Was beim Wechsel von einer physischen zu einer VMware-Lösung zu beachten ist

Aus Oracle-Sicht ist praktisch nichts zu ändern, wenn man Oracle auf einer virtuellen Maschine auf einem ESXi-Host ausführt. Man sollte sich jedoch bewusst sein, dass sich die Oracle-Datenbank oder Anwendung auf einer virtuellen Plattform befindet, und die damit verbundenen Auswirkungen berücksichtigen. Wenn die virtuelle Maschine beispielsweise mit einer anderen virtuellen Maschine im Wettbewerb um Ressourcen steht, wirkt sich dies auf die Oracle-Performance aus, da virtuelle Maschinen auf demselben ESXi-Host die Ressourcen gemeinsam nutzen.

Für Oracle oder andere kritische Workloads empfiehlt sich dringend, eine Reservierung für den Arbeitsspeicher und die CPU der virtuellen Maschine vorzunehmen, um der VM die erforderlichen Ressourcen zu garantieren. Für eine Oracle-Datenbank ist die Speicher-Reservierung so einzustellen, dass sie den Anforderungen des Betriebssystems plus SGA plus PGA plus Background-Prozesse entspricht.

VMware verfügt über eine Vielzahl von Anleitungen dafür, wie man ESXi und die VM einrichtet, um die Leistung von Oracle zu optimieren. Vor der Virtualisierung sollte man sich mit den Vorschlägen im „Oracle on VMware Best Practice Guide“ vertraut machen und diese umsetzen. Zudem sollte man sich auch die Ausführungen zu Oracle im „One Stop Shop“ von VMware ansehen, beides ist auf „vmware. com“ verfügbar.

Wie ein Oracle Redo Log auf einer VMware Virtual Machine funktioniert

VMware hat sich inzwischen weit über die Welt der einfachen Virtualisierung von Servern hinaus entwickelt, es gibt jetzt auch weit genutzte Lösungen zum Virtual Storage (vSAN) und Virtual Networking (NSX). Die Beispiele in diesem Artikel beziehen sich jedoch auf das klassische Disk-Array. Der Artikel geht jetzt nicht durch jede Ebene auf dem Stack, sondern hebt nur gewisse Punkte als Beispiel hervor. Zunächst wird man wie bei einem physischen Server die Log-Files auf vom Datenverkehr der EA-Datei getrennten Festplatten anlegen. Nachfolgend ist zu sehen, was geschieht, nachdem Oracle festgestellt hat, dass in das Log geschrieben werden muss:

Top User Events

Event	Event Class	% Event	Avg Active Sessions
log file sync	Commit	42.35	6.00
CPU + Wait for CPU	CPU	30.20	4.28
buffer busy waits	Concurrency	4.31	0.61
SQL*Net more data from client	Network	3.53	0.50
db file scattered read	User I/O	2.75	0.39

Abbildung 2: Beispiel für ein Oracle-Workload-Repository

- *Schritt eins*
  E/A gelangt über virtuelle SCSI-Controller in eine virtuelle Maschine. Bevor ein Schreibvorgang durchgeführt wird, muss die Berechtigung eingeholt werden. Das heißt, der erste Schreibzugriff ist die Berechtigung zum Schreiben. Diese Anforderung wird über den SCSI-Controller aus der VM heraus übertragen. Man sollte mehrere SCSI-Controller konfigurieren, um die E/As zu trennen, da das automatische Interrupt Coalescing auf einer Controller-Ebene funktioniert.

- *Schritt zwei*
  Der Hypervisor leitet diese Anforderung an den HBA weiter, der an den ESXi angeschlossen ist.

- *Schritt drei*
  Der HBA sendet die Anforderung an die gewünschte Stelle in das Array.

- *Schritt vier*
  Das Array erteilt die Berechtigung, indem es diese Nachricht zusammen mit einer Meldung zurücksendet, die das Senden von Daten erlaubt.

- *Schritt fünf*
  Das Betriebssystem sendet das Log über den SCSI-Controller weiter zur HBA Fibre Disk.

- *Schritt sechs*
  Der Schreibvorgang wird in eine „vmdk" (ein Dateiformat, das Container für virtuelle Festplatten beschreibt, die in virtuellen Maschinen verwendet werden sollen) auf dem Speicher-Array geschrieben. Das Array sendet eine Bestätigung auf umgekehrtem Weg zurück und wenn diese an das Betriebssystem zurückgegeben wird, ist sichergestellt, dass der Schreibvorgang auf der Festplatte festgehalten wurde.

Aus dem Beispiel ist ersichtlich, dass dies ein recht langer Weg für ein einfaches Redo-Log-Schreiben ist, wo eventuell Latenzen entstehen können. Bei einem Problem wird man als Erstes eine hohe Logfile-Synchronisation in seinem ASH-Bericht bemerken *(siehe Abbildung 2)*.

Dies führt uns zu der Annahme, dass es einen Engpass beim Schreiben von E/As gibt. Es wurde bereits erklärt, dass es ein ziemlich langer Weg von der Oracle-Instanz zum Speichergerät ist, bevor ein Schreibvorgang bestätigt werden kann. Es muss also irgendwo auf diesem Weg ein Problem geben. Aber wo ist es?

Esxtop ist das Tool zur Diagnose von Performance-Problemen auf VMware ESXi, es funktioniert auf gleiche Weise wie die ASH-Daten für Oracle. Esxtop kann in Echtzeit verfolgt oder aufgezeichnet und später angesehen werden. Ein wichtiger Punkt ist, dass man Root-Zugriff auf einen ESXi-Host benötigt, um dieses Dienstprogramm auszuführen.

Wer daran interessiert ist, einen Engpass auf seinem System zu identifizieren, sei es im Netzwerk oder im Speicher, für den ist Esxtop die richtige Wahl. Wenn man den VMware-Support mit einem Leistungsproblem anruft, wird dies der erste Bericht sein, der angefordert wird.

Auf einem Server ist „Watch Esxtop" direkt einsehbar; indem man einfach „esxtop" in der Eingabeaufforderung eingibt, wird die Oberfläche angezeigt. Esxtop lässt sich auch aufzeichnen. Eine Aufzeichnung wird über die Befehlszeile oder den vSphere Web Client abgerufen. In VMware KB 1033346 steht, wie man diese aktivieren kann *(siehe Abbildung 3)*. Eine aufgezeichnete Sitzung wird mit „esxtop -R vm-support_dir_path" wiedergegeben *(siehe Abbildung 4)*.

Interpretation von Esxtop-Statistiken

Abbildung 4 ist ein Beispiel dafür, wie die Esxtop-Ausgabe aussieht. Um zu sehen, ob es eine Latenz auf dem HBA gibt, drückt man „d", um die Wartezeiten auf den Host-Bus-Adaptern anzuzeigen. VMware KB 1008205 beschreibt, wie man zwischen den Bildschirmen wechseln kann, um die Metriken zu sehen, die den Speicher be-

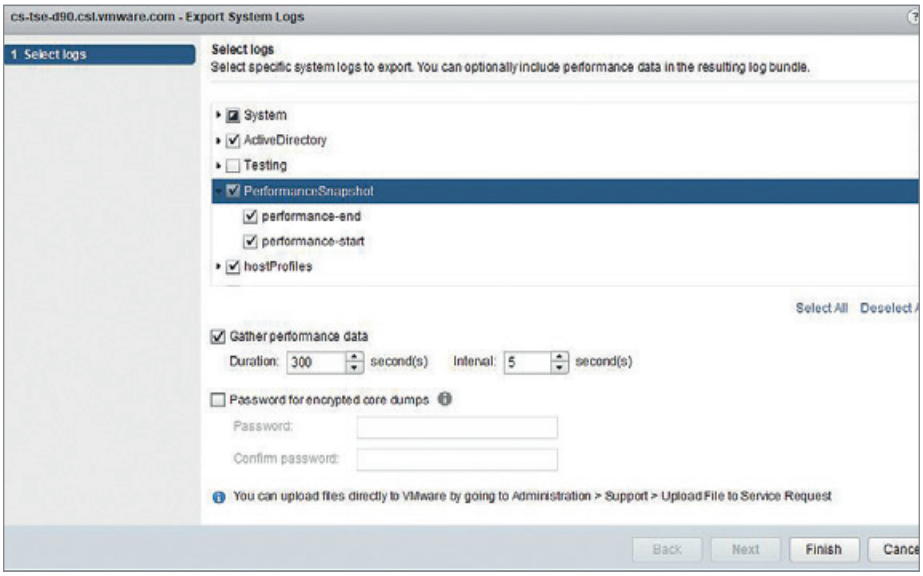


Abbildung 3: So löst man eine Aufzeichnung von Esxtop für 300 Sekunden im Abstand von 5 Sekunden aus

```
11:52:48am up 144 days 35 min, 839 worlds, 9 VMs, 45 vCPUs; CPU load average: 0.15, 0.08, 0.05
SCSI stats: 0 conflicts,   0.00 conflicts/sec, 0.00 usec/cmd
Cache hit: 56.10%, Cache miss: 43.90%, fssRead:  0.00, fssWrite:  1.27
hBuffers: 0, maxFlushQuantum: 1024, flush interval: 0 ms

 ADAPTR PATH                NPTH   CMDS/s  READS/s WRITES/s MBREAD/s MBWRTN/s DAVG/cmd KAVG/cmd GAVG/cmd QAVG/cmd
 vmhba0 -                      1   581.11     4.45   576.66     0.01    54.54     0.21     0.08     0.29     0.03
 vmhba1 -                      0     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
 vmhba2 -                      1     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00     0.00
 vmhba64 -                     2     1.27     0.00     0.95     0.00     0.01     1.58     0.02     1.61     0.00
```

Abbildung 4: Wiedergabe der Esxtop-Aufzeichnung

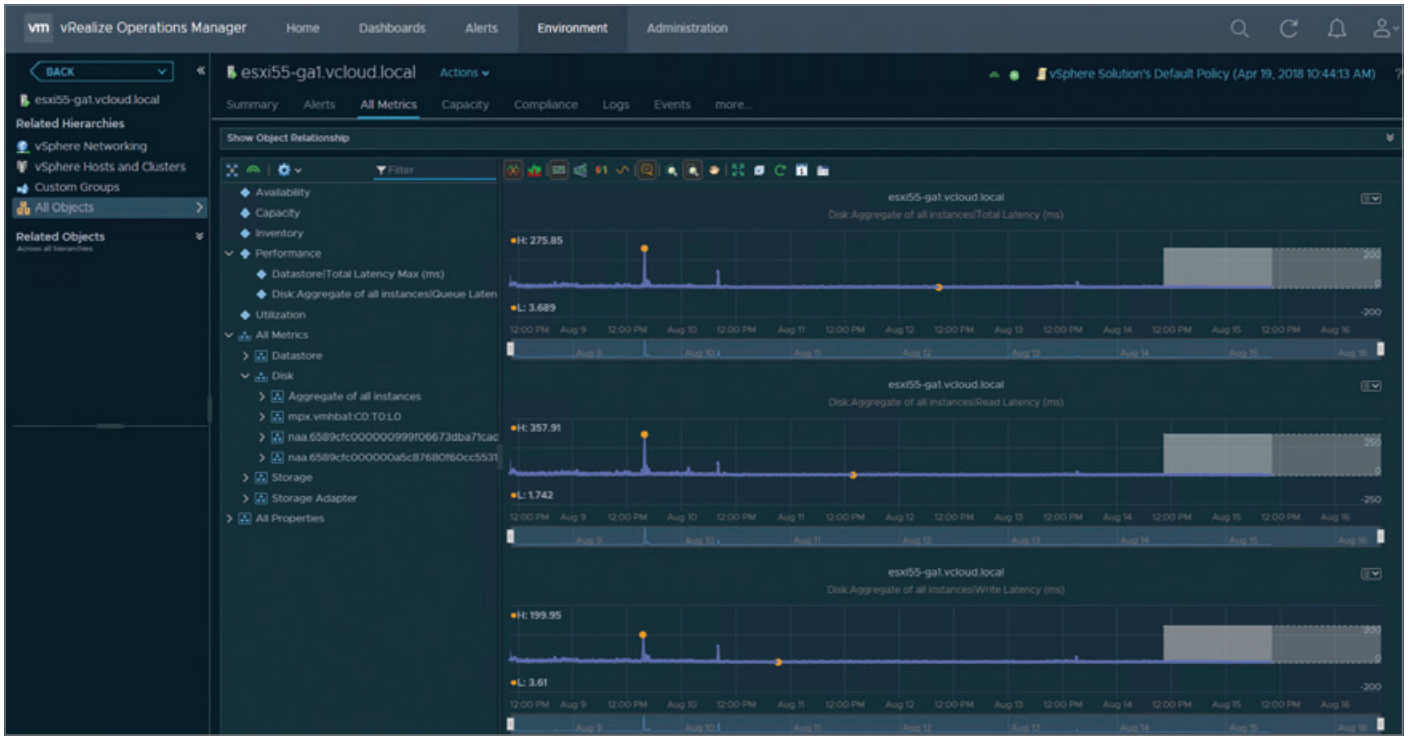


Abbildung 5: Eine Geräte-Leistungsseite von VMware vRealize Operations

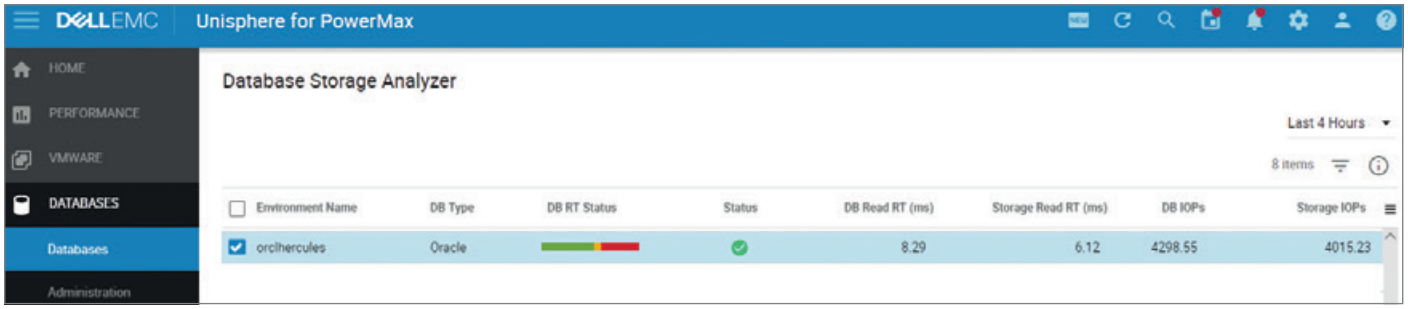


Abbildung 6: Unisphere Oracle-Leistungsstatistik

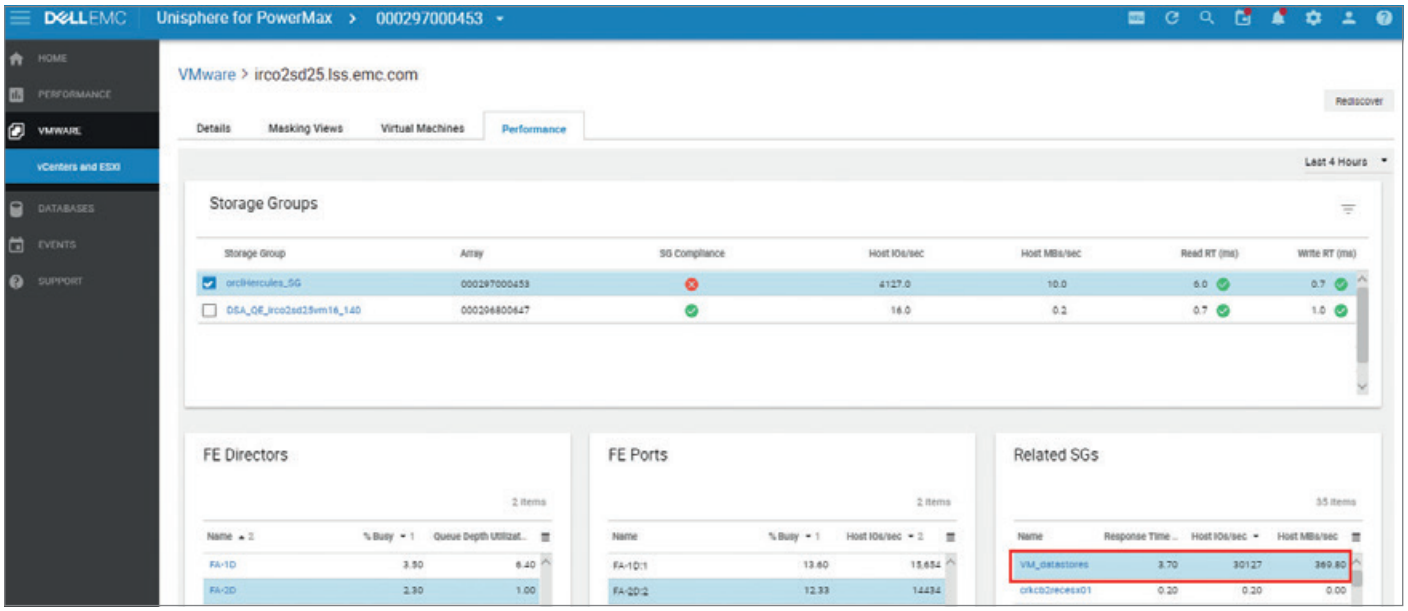


Abbildung 7: Unisphere ESXi-Informationen

einflussen. Auf „vmware.com“ ist mitunter die beste Referenz zu finden, der Titel ist „Interpreting esxtop statistics“.

Wenn man den Esxtop-Bericht über ein Speicherproblem betrachtet, dann schaut man sich die Zähler von DAVG und KVAG an. Allgemeine Richtlinien gelten, wenn die DAVG-Werte hoch sind (mehr als 30 ms gelten als hoch), dann liegt das Problem außerhalb des ESXi, also beim Netzwerk, Frontend-Controller oder Speichergerät.

In einem Szenario, in dem KVAG hoch ist (15-20 ms ist für die meisten VMware-Administratoren eine Warnschwelle), liegt das Problem auf dem ESXi-Host. Im Allgemeinen bedeutet dies ein Problem mit den HBAs oder deren Treibern, die diesem ESXi zugeordnet sind. Möglicherweise ist die Tiefe der Queue auf den HBAs zu erhöhen; hier empfiehlt es sich, die Richtlinien der jeweiligen Hersteller zu befolgen.

Die KVAG kann auch ein Problem mit der Tiefe der Queue in den Software-ISCSI-Adaptern auf der virtuellen Maschine selbst bedeuten. VMware KB 2053145 beschreibt, wie verschiedene Queue-Größen anzupassen sind, diese Konfiguration ist jedoch ein Balanceakt. Wer die Tiefe der Warteschlange überall erhöht, vermeidet Wiederholungsversuche, die die Datenbank-Leistung beeinträchtigen. Wenn man jedoch in einer Warteschlange festsitzt, steigt die Latenzzeit für den Zugriff auf die Virtual-Machine-Disk-Dateien.

Wenn die DVAG hoch ist, ist es an der Zeit, ein Gespräch mit den Storage-Administratoren zu führen. Die meisten Storage-Anbieter bieten Best Practices für eine optimale Anbindung der Speicherpfade an die ESXi-Hosts. VMware bietet verschiedene Speicherpfad-Richtlinien, darunter MRU (Standard), Fixed und Round-Robin. Zusätzlich bieten die Storage-Anbieter auch Lösungen für Drittanbieter an, die auf die Bedürfnisse von Multipathing-Anbietern zugeschnitten sind.

VMware überlässt die optimale Richtlinie zur Auswahl der Ein-/Ausgabepfade den jeweiligen Storage-Anbietern. Dell EMC Powerpath bietet beispielsweise HBA-Monitore mit Warteschlangen-Tiefen und Informationen über die CPU-Auslastung von Speichermanagern, die zur Optimierung einer effektiven Auswahl von E/A-Pfaden verwendet werden. Dies kann zu einem höheren Durchsatz und einer besseren Leistung gegenüber Native-Multipathing-Lösungen führen.

Weitere nützliche Tools zur Überprüfung der Leistung

Eine noch bessere Möglichkeit, um ESXi-Leistungsstatistiken zu betrachten, ist die Verwendung von VMware vRealize Operations, was allerdings eine separate Lizenz erfordert. Es ermöglicht Administratoren, mit wenigen Klicks ihre gesamte Infrastruktur zu überblicken. Man kann Warnmeldungen einstellen, wenn die Festplatten-Partitionen fast voll sind oder wenn auf dem ESXi-Host Ballooning/Swapping anfällt. Es wird natürlich auch jede Latenzzeit beim Lesen oder Schreiben auf vielen verschiedenen Ebenen angezeigt *(siehe Abbildung 5)*.

Viele VMware-Kunden verwenden das PowerMax-Array von Dell EMC. In diesem Fall wird Unisphere mitgeliefert. Esxtop hat keine Möglichkeit, die Ereignisse außerhalb des Hypervisor darzustellen. In Unisphere kann man die Suche nach Zahlen jedoch fortsetzen, indem man den Abschnitt „Database Storage Analyzer“ der Unisphere-for-PowerMax-Anwendung von Dell EMC betrachtet sowie DB- und Storage-KPIs wie Read, Writes, IOPS für die Oracle-Datenbank und PowerMax/VMAX-Arrays vergleicht. Das Beispiel in *Abbildung 6* zeigt, dass die Leistung der Oracle-Anwendung unzureichend ist.

Im VMware-Bereich von Unisphere for PowerMax können weitere Recherchen durchgeführt werden, indem der ESXi-Host und die Speichergruppe für die Oracle-Datenbank ausgewählt werden *(siehe Abbildung 7)*. Aus den Informationen im obigen Dashboard geht hervor, dass die Datenbank wegen von Ressourcen-Konflikten auf dem Frontend-Port aufgrund von Workloads anderer Speichergruppen mit hohem Host-EA ihr Service-Level nicht erreicht. Dies kann durch die Festlegung eines anderen Service-Levels als Diamant für die konkurrierenden Speichergruppen behoben werden, um die Auswirkungen auf die Reaktionszeit der Oracle-Datenbank zu reduzieren; alternativ könnten die konkurrierenden Anwendungen auf einen anderen Satz von Frontend-Direktoren verschoben werden.

Das Tool verbindet die VMware- mit der Oracle-Performance im Zusammenspiel mit dem Storage zugleich und ermöglicht es Speicher-, Datenbank- und VMware-Administratoren, mehr Transparenz über alle Ebenen des E/A-Stacks zu erhalten, was die Zeit zur Lösung von Performance-Problemen verkürzt.

Der „schneller“-Knopf

Es ist möglich, eine virtuelle Maschine als „Latenz optimiert“ einzustellen. Dies kann über den Web-Client mit „VM → Edit Settings → VM Options → Advanced For Oracle“ erfolgen, es wird allerdings empfohlen, diese Einstellung nicht ohne sehr guten Grund zu verwenden. Zunächst sollte man seine VMware-Umgebung gemäß dem Best Practice Guide „Oracle on VMware“ einrichten. Wer diese Richtlinien befolgt und ein Performance-Problem hat, sollte die Referenz „Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere VMs“ lesen. Man sollte die Einstellungen in diesem Dokument vor der Implementierung überprüfen und deren Auswirkungen verstehen, Änderungen sollten jeweils nur einzeln gemacht und anschließend getestet werden. Beide Dokumente sind auf „vmware.com“ verfügbar.

Wer Probleme beim Ausführen seiner Oracle-Datenbanken oder -Anwendungen auf VMware hat, wendet sich an den VMware-Support. Dort den Kundendienst bitten, dass der Fall zur Autorin durchgestellt wird, und sie kümmert sich darum.

Hinweis: Übersetzt aus dem Englischen von einer unabhängigen Agentur.

Yvonne Murphy
ymurphy@vmware.com

Realistischere Kosten für den Tabellen-Zugriff über einen Index

Clemens Bleile, dbi services sa

Die am meisten verwendete Datenablage in Oracle sind sogenannte „Heap-Tabellen"; die am meisten eingesetzte Zugriffstruktur, um auf die Daten effizient zugreifen zu können, sind B-Tree-Indizes (zumindest im OLTP-Umfeld). Zur bestmöglichen Performance beim Zugriff auf die Daten gehört dementsprechend die richtige Indexierung der Tabellen. Auch bei korrekter Indexierung gibt es Fälle, bei denen der Oracle Cost Based Optimizer einen Plan berechnet, der den gewünschten Index nicht nutzt. Der Artikel zeigt Möglichkeiten auf, den Cost Based Optimizer zu unterstützen, um realistischere Kosten beim Index-Zugriff zu berechnen.

Es gibt heutzutage viele Möglichkeiten, um ohne B-Tree-Indizes schnell auf Daten zuzugreifen, etwa Exadata mit seinen Storage-Indizes, dem In-Memory-Column-Store oder mit Funktionalitäten wie Partitionierung, Parallelität, Direct Path Reads oder Komprimierung. Trotzdem setzen die meisten Applikationen den effizienten Datenzugriff über traditionelle B-Tree-Indizes um.

Die Indizierung erlaubt den Zugriff auf eine (üblicherweise Teil-) Menge der Tabellendaten in kürzester Zeit oder unter Nutzung geringster Ressourcen. Man erkauft sich den schnellen Zugriff durch höheren Platzbedarf und ein erhöhtes Ressourcen-Aufkommen bei der Änderung der indizierten Spalten (durch das Nachführen des Index und die Generierung erhöhter Redolog-Datenmenge).

Ein B-Tree-Index ist aus einem Wurzel-Knoten („root node"), 0-n Ebenen mit Verzweigungsknoten („branch blocks") und einer Ebene der Blatt-Knoten („leaf blocks") aufgebaut *(siehe Abbildung 1)*. Falls alle Daten in den Wurzelknoten passen, ist die Wurzel quasi gleichzeitig das Index-Blatt. Das Navigieren durch den Index-Baum geschieht sehr schnell, weil Oracle auf jedem Verzweigungsknoten eine Ebene tiefer springt, bis man am Blatt ankommt; man macht also bis dahin maximal n logische IOs. Diese Anzahl n der Ebenen bis zum Blatt (also der Wurzel-Knoten plus die Ebenen der Verzweigungs-Knoten) wird bei Oracle als „BLEVEL" („branch level") bezeichnet. Die Höhe („height") des Baums inklusive der Ebene der Blätter ist also „BLEVEL + 1". Auch bei sehr großen Indizes wird ein BLEVEL von 3 kaum überschritten *[1]*. Am Blatt angekommen kann Oracle den/die indizierten Spaltenwert/e und den Verweis in die Tabelle („RowId") lesen. Sollten mehrere Werte gelesen werden, wird dies als „Index-Range-Scan" bezeichnet.

Wie aus *Abbildung 1* ersichtlich, sind die benachbarten Blatt-Knoten über Zei-

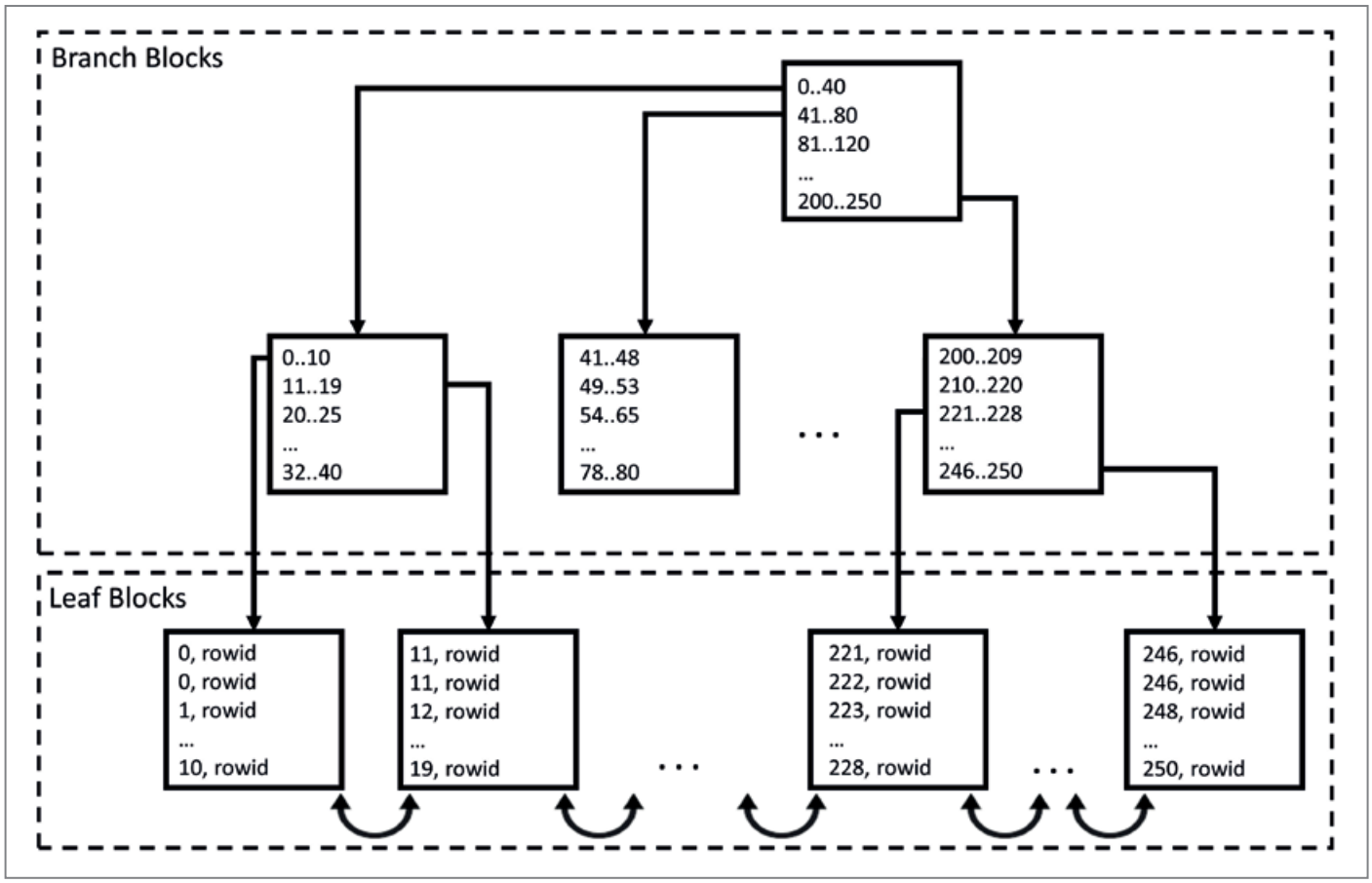


Abbildung 1: Der Aufbau eines B-Tree

ger verbunden, sodass die Blätter in aufsteigender und absteigender Reihenfolge gelesen werden können. Falls Oracle die benötigten Daten für die Abfrage nicht aus dem Index befriedigen kann, muss über die RowId(s) auf die Tabelle zugegriffen werden. Wenn man sich vor Augen führt, dass ein Blatt-Knoten viele Verweise auf die Tabellen-Zeilen enthalten kann, wird klar, dass der eigentliche Index-Zugriff (n logische IOs für den BLEVEL plus den Range-Scan über ein oder mehrere Blätter) in den meisten Fällen nicht sehr Ressourcen-intensiv ist. Mit einem Unique-Index auf einer „NUMBER"-Spalte können beispielsweise trotz eines „PCTFREE" von „30" initial durchschnittlich schon 370 Werte pro Blatt-Knoten

```
SQL> create table ti1 as select * from all_objects;
SQL> insert into ti1 select * from ti1;
…
SQL> insert into ti1 select * from ti1;
SQL> commit;
SQL> create sequence ti1s;
SQL> begin
   2 for i in (select rowid from ti1 order by object_id) loop
   3     update ti1 set object_id=ti1s.nextval where rowid=i.rowid;
   4 end loop;
   5 end;
   6 /
SQL> commit;
SQL> create unique index ti1_i1 on ti1 (object_id) pctfree 30;
SQL> exec dbms_stats.gather_table_stats(user,'TI1');
SQL> select blevel, leaf_blocks, num_rows, round(num_rows/leaf_blocks) avg_rows_per_leaf  from ind where
index_name='TI1_I1';

    BLEVEL LEAF_BLOCKS   NUM_ROWS AVG_ROWS_PER_LEAF
---------- ----------- ---------- -----------------
         2         827     306176               370
```

Listing 1

Platz finden *(siehe Listing 1)*.

Anmerkung: Der Wert „PCTFREE = 30“ wird nur beim Anlegen des Index berücksichtigt, um frühe „Block-Splits“ zu vermeiden. Zukünftige Insert-Statements füllen die Index-Blöcke zur Gänze. Bei einem BLEVEL von „2“ sind also nur drei logische IOs erforderlich, um beispielsweise auf 370 Werte im Index zuzugreifen *(siehe Listing 2)*. Teuer wird es, wenn man auch auf Daten in der Tabelle zugreifen muss *(siehe Listing 3)*.

Man muss im obigen Beispiel bei 370 RowId-Zeigern auf 370 unterschiedliche Tabellen-Blöcke zugreifen. Wenn also die RowIds beim Index-Range-Scan auf viele unterschiedliche Tabellen-Blöcke zeigen, wird der Tabellenzugriff teuer. Ist die Tabelle gemäß der indizierten Spalte(n) sortiert, so reduziert sich die Anzahl der Tabellen-Zugriffe *(siehe Listing 4)*.

Oracle nimmt also die RowIds aus dem Index-Range-Scan, sortiert sie nach Datei- und Block-Nummer und greift nur einmal auf jeden benötigten Tabellen-Block zu. Die Statistik, die der Cost Based Optimizer nutzt, um zu berücksichtigen, wie verteilt die Daten-Blöcke der Tabelle bei einem Zugriff nach Index-Range-Scan sind, ist der sogenannte „Clustering Factor“. Im schlechtesten Fall hat der Wert nahe-

```
SQL> set autotrace on
SQL> select count(*) from ti1 where object_id <= 370;

  COUNT(*)
----------
       370

------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost  |
------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |     1 |     5 |     3 |
|   1 |  SORT AGGREGATE   |        |     1 |     5 |       |
|*  2 |   INDEX RANGE SCAN| TI1_I1 |   370 |  1850 |     3 |
------------------------------------------------------------

Statistics
----------------------------------------------------------
          3  consistent gets
```

Listing 2

```
SQL> select /*+ index(ti1 ti1_i1) */ max(object_name) from ti1 where object_id <= 370;

-------------------------------------------------------------------------------
| Id  | Operation                            | Name   | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |        |     1 |    25 |   373 |
|   1 |  SORT AGGREGATE                      |        |     1 |    25 |       |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| TI1    |   370 |  9250 |   373 |
|*  3 |    INDEX RANGE SCAN                  | TI1_I1 |   370 |       |     3 |
-------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
        373  consistent gets
```

Listing 3

```
SQL> create table ti2 as select * from ti1 order by object_id;
SQL> create unique index ti2_i1 on ti2 (object_id) pctfree 30;
SQL> exec dbms_stats.gather_table_stats(user,'TI2');
SQL> set autotrace on
SQL> select max(object_name) from ti2 where object_id <= 370;

-------------------------------------------------------------------------------
| Id  | Operation                            | Name   | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |        |     1 |    25 |    10 |
|   1 |  SORT AGGREGATE                      |        |     1 |    25 |       |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| TI2    |   370 |  9250 |    10 |
|*  3 |    INDEX RANGE SCAN                  | TI2_I1 |   370 |       |     3 |
-------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
         11  consistent gets
```

Listing 4

```
SQL> select index_name, clustering_factor, leaf_blocks, blocks table_blocks, ind.num_rows
  2  from ind, tabs where ind.table_name like 'TI%'
  3  and ind.table_name=tabs.table_name ;

INDEX_NAME                     CLUSTERING_FACTOR LEAF_BLOCKS TABLE_BLOCKS   NUM_ROWS
------------------------------ ----------------- ----------- ------------ ----------
TI1_I1                                    306184         827         5172     306240
TI2_I1                                      5126         827         5216     306240
```

Listing 5

```
BLEVEL + Anzahl Blatt-Blöcke + Anzahl Tabellen-Blöcke =
BLEVEL + (effektive Index-Selektivität * Blatt-Knoten) + (effektive
Tabellen-Selektivität * Clustering Factor) =
2 + ceil(0.0012082 * 827) + ceil(0.0012082 * 306184) =
2 + 1 + 370 = 373
```

Listing 6

zu die Anzahl der Zeilen in der Tabelle, im besten Fall nahezu die Anzahl der Blöcke in der Tabelle (Abweichungen dieser Regel können sich etwa bei vielen leeren Tabellen-Blöcken ergeben. Der Clustering Factor gibt an, wie viele IO-Tabellenzugriffe nötig sind, um die Tabelle vollständig über einen Index-Full-Scan zu lesen. Multipliziert mit der Selektivität der Abfrage ergeben sich bei einem Index-Range-Scan die voraussichtlichen IO-Operationen auf der Tabelle *(siehe Listing 5)*. Die Berechnungsmethode (vereinfacht im I/O-Kosten-Modell) der Kosten für obigen Plan mit schlechtem Clustering Factor ist in *Listing 6* dargestellt.

Anmerkung: Es gibt bei Oracle oft Ausnahmen zur Regel. Die obige Berechnungsmethode wird hier nur als Grundregel angesehen und deckt nicht alle Fälle ab. Eine Ausnahme ist beispielweise „BLEVEL = 1“. In dem Fall wird BLEVEL (der Wurzelknoten-Zugriff) nicht als Kosten berechnet.

Wie man im obigen Beispiel sieht, beeinflusst der Clustering Factor (bei Aus-

führungsplan mit Index-Range-Scan und anschließendem Tabellen-Zugriff) die Kosten maßgeblich. Je kleiner also der Clustering Factor, desto wahrscheinlicher wird ein Plan mit entsprechendem Index die geringsten Kosten haben. Der Clustering Factor bleibt natürlich auch beim Neuerstellen des Index gleich; nur die Änderung des Orts der Daten in der Tabel-

```
SQL> select index_name, clustering_factor, leaf_blocks, blocks table_blocks, ind.num_rows
  2  from ind, tabs where ind.table_name like 'TI%'
  3  and ind.table_name=tabs.table_name ;

INDEX_NAME                     CLUSTERING_FACTOR LEAF_BLOCKS TABLE_BLOCKS   NUM_ROWS
------------------------------ ----------------- ----------- ------------ ----------
TI1_I1                                    306184         827         5172     306240
TI2_I1                                      5126         827         5216     306240
TI3_I1                                    210436         827         5158     306240
```

Listing 8

```
SQL> select max(object_name) from ti3 where object_id <= 370;

-----------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |     1 |    25 |   258   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE                     |         |     1 |    25 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| TI3    |   370 |  9250 |   258   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                 | TI3_I1  |   370 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          9  consistent gets
```

Listing 9

```
SQL> select dbms_stats.get_prefs('TABLE_CACHED_BLOCKS',user,'TI3') from dual;

DBMS_STATS.GET_PREFS('TABLE_CACHED_BLOCKS',USER,'TI3')
------------------------------------------------------
1

SQL> exec dbms_stats.set_table_prefs(user,'TI3','TABLE_CACHED_BLOCKS',16);
SQL> select dbms_stats.get_prefs('TABLE_CACHED_BLOCKS',user,'TI3') from dual;

DBMS_STATS.GET_PREFS('TABLE_CACHED_BLOCKS',USER,'TI3')
------------------------------------------------------
16
```

Listing 10

```
SQL> exec dbms_stats.gather_table_stats(user,'TI3',cascade=>TRUE);
SQL> select index_name, clustering_factor, leaf_blocks, blocks table_blocks, ind.num_rows
  2  from ind, tabs where ind.table_name like 'TI%'
  3  and ind.table_name=tabs.table_name ;

INDEX_NAME                     CLUSTERING_FACTOR LEAF_BLOCKS TABLE_BLOCKS   NUM_ROWS
------------------------------ ----------------- ----------- ------------ ----------
TI1_I1                                    306184         827         5172     306240
TI2_I1                                      5126         827         5216     306240
TI3_I1                                      5116         827         5158     306240
```

Listing 11

le beeinflusst den Clustering Factor (etwa durch ein „ALTER TABLE SHRINK SPACE“). Um den Clustering Factor zu reduzieren, lässt sich die Tabelle sortiert nach den Index-Spalten neu erstellen. Das hat allerdings zwei Nachteile:

- Der Clustering Factor anderer Indizes kann sich verschlechtern
- Durch Hinzufügen oder Löschen von Daten kann sich der Clustering Factor dieses Index wieder verschlechtern

In der Standard-Einstellung wird der Clustering Factor wie folgt berechnet: Oracle liest die RowId der Tabelle sortiert nach den Index-Spalten. Sind zwei benachbarte Zeilen in der sortierten Ausgabe nicht im gleichen Tabellen-Block, wird der Clustering Factor um „1“ erhöht. Sind sie im gleichen Block, bleibt der Clustering Factor gleich. Diese Berechnungsmethode ist sehr pessimistisch, weil sie davon ausgeht, dass bei einem Index-Range-Scan nur der referenzierte Block des vorherigen Index-Eintrags im Cache bleibt. Tatsächlich ist jedoch die Wahrscheinlichkeit groß, dass mehrere der vorher gelesenen Tabellen-Blöcke im Datenbank-Cache verbleiben. Im Extremfall hat man eine Tabelle, in die ein aufsteigender Wert einer Sequenz durch mehrere Sessions gleichzeitig eingefügt wird.

Aufgrund des Algorithmus bei Inserts in einen Automatic Segment Space Management Tablespace (ASSM, Standard in Oracle) werden die gleichzeitigen Inserts auf mehrere Blöcke verteilt. Bei n Sessions, die Daten einfügen, kann es passieren, dass Zeilen mit dem aufsteigenden Sequenz-Wert alternierend in n Blöcke eingefügt werden. Wird der aufsteigende

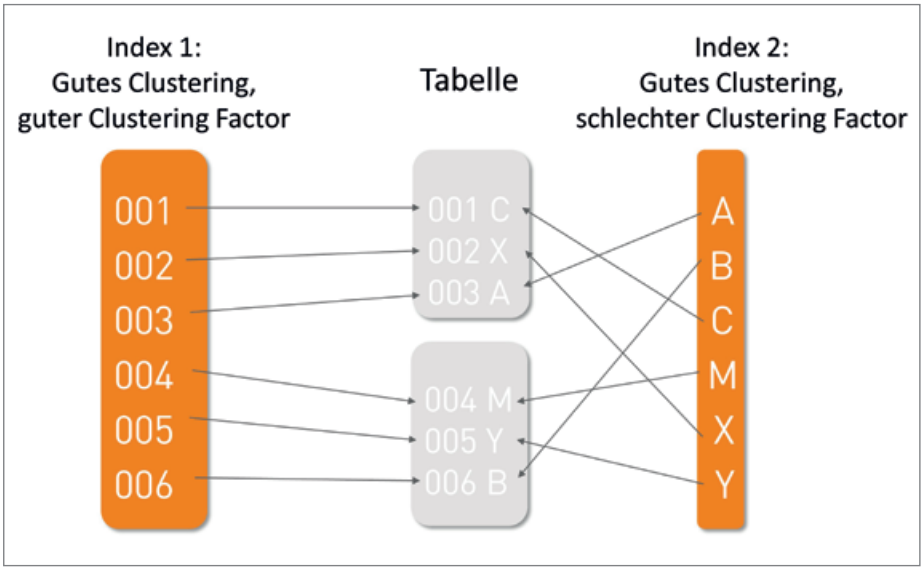


Abbildung 2: Gutes Clustering, schlechter Clustering Factor

Wert indexiert, so wird der Clustering Factor des Index sehr groß, obwohl ein Index-Range-Scan nur auf wenige (meist nur n) Tabellen-Blöcke zugreifen muss. In *Listing 7* werden in drei Sessions gleichzeitig in die Tabelle „TI3“ eingefügt *(siehe „http://www.doag.org/go/redstack201806listings“)*. Der Clustering Factor des neuen Index „TI3_I1“ ist hoch *(siehe Listing 8)*, die tatsächlichen Logical Reads jedoch gering *(siehe Listing 9 und Abbildung 2)*.

Um das Problem zu adressieren, hat Oracle in 11.2.0.4 (beziehungsweise mit dem Enhancement Request des Bug 13262857) die Statistik-Präferenz „TABLE_CACHED_BLOCKS“ eingeführt. Ein Wert zwischen 1 und 255 oder AUTO kann auf globaler, DB-, Session- oder Tabellen-Ebene angegeben werden *(siehe Listing 10)*.

Wenn also der momentan referenzierte Tabellen-Block beim Index-Statistik-Sammeln innerhalb der letzten sechzehn referenzierten Blöcke schon einmal vorkam, dann wird der Clustering Factor nicht erhöht. Man geht also davon aus, dass die zuletzt referenzierten sechzehn Tabellen-Blöcke beim Index-Range-Scan im Cache bleiben. Der Wert „AUTO“ bedeutet der kleinste Wert aus 255, 1 Prozent der Tabellen-Blöcke und 0,1% Prozent der Buffer-Cache-Größe (in Blöcken). Nach dem Sammeln der Statistiken verbessert sich der Clustering Factor *(siehe Listing 11)*; dementsprechend reduzieren sich die Kosten von 258 auf 10 *(siehe Listing 12)*.

Da der Clustering Factor maßgeblich für die Kosten der Index-Nutzung ist, stellt diese Erweiterung eine massive Verbesserung der Qualität von Index-Kosten dar. Leider wird die Präferenz meist nicht genutzt und es gibt noch zahlreiche Datenbanken, die mit angepasstem Parameter „OPTIMIZER_INDEX_COST_ADJ“

```
SQL> select max(object_name) from ti3 where object_id <= 370;

-----------------------------------------------------------------------------------------------
| Id  | Operation                            | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |        |     1 |    25 |    10   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE                      |        |     1 |    25 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| TI3    |   370 |  9250 |    10   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                  | TI3_I1 |   370 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Statistics
----------------------------------------------------------
          9  consistent gets
```

Listing 12

```
SQL> column value format a12
SQL> SELECT sname,nvl(to_char(sval1),spare4) value
  2  FROM sys.optstat_hist_control$
  3  WHERE sname='TABLE_CACHED_BLOCKS';

SNAME                          VALUE
------------------------------ ------------
TABLE_CACHED_BLOCKS            42
```

Listing 13

```
SQL> alter session set "_optimizer_compute_index_stats"=false;
SQL> create index …;
SQL> alter session set "_optimizer_compute_index_stats"=true;
SQL> exec dbms_stats.gather_index_stats(…);
```

Listing 14

betrieben werden, um dadurch die Index-Nutzung zu forcieren. Dies sollte schon seit Einführung des CPU-Kosten-Modells mit System-Statistiken nicht mehr gemacht werden; seit der Verfügbarkeit der Präferenz „TABLE_CACHED_BLOCKS“ gibt es einen weiteren Grund, dies nicht mehr zu tun.

Was ist nun ein guter Wert für die Präferenz „TABLE_CACHED_BLOCKS“? Tests haben gezeigt, dass bei großen Tabellen auch ein Maximalwert von „255“ keine Nachteile bringt. Anders verhält es sich, wenn die Präferenz mit beispielsweise „255“ nahe an der Anzahl von Blöcken in der Tabelle liegt. Das kann zu Index-Plänen führen, wenn ein Full-Table-Scan die bessere Variante wäre. Aus dem Grund hat Oracle beim Wert „AUTO“ 1 Prozent der Tabellen-Blöcke als möglichen Wert angegeben. Da bei „AUTO“ der kleinste der drei Werte (255, 1 Prozent der Tabelle, 0,1 Prozent des DB-Cache) genommen wird, führt dies bei kleinen Tabellen quasi immer zu 1 Prozent der Tabellengröße. Manch ein DBA vertritt die Meinung, dass 16 ein guter Wert sei, da dies die maximale Verteilung in Blöcke bei konkurrierenden Inserts in ASSM-Tablespaces ist. Richard Foote (Mr. Index) gibt (mit einem Augenzwinkern) den Wert 42 als mögliche Alternative an *[2, 9]*: 42 ist deshalb ein guter Wert, weil es gemäß „Per Anhalter durch die Galaxis“ von Douglas Adams die Antwort auf alle Fragen ist: „The Answer to the Ultimate Question of Life, The Universe, and Everything from the supercomputer, Deep Thought, specially built for this purpose. It takes Deep Thought 7½ million years to compute and check the answer, which turns out to be 42“.

Im Normalfall sollte geprüft werden, ob Tabellen mit indizierten Spalten, die eine aufsteigende Sequenz enthalten, gute Kandidaten für die „TABLE_CACHED_BLOCKS“-Präferenz sind. Falls es nur mit reduziertem „OPTIMIZER_INDEX_COST_ADJ“ gut funktioniert, sollten die System-Statistiken geprüft werden und mit der globalen „TABLE_CACHED_BLOCKS“-Präferenz auf „AUTO“, „16“ oder auch „42“ getestet werden. Der Wert „AUTO“ (intern wird „0“ verwendet) wird mit „DBMS_STATS.AUTO_TABLE_CACHED_BLOCKS“ gesetzt: „exec dbms_stats.set_table_prefs(user,'TI1','TABLE_CACHED_BLOCKS',DBMS_STATS.AUTO_TABLE_CACHED_BLOCKS);“ oder „exec dbms_stats.set_global_prefs('TABLE_CACHED_BLOCKS',DBMS_STATS.AUTO_TABLE_CACHED_BLOCKS);“. Oracle führt die globale Präferenz in der Tabelle „SYS.OPSTAT_HIST_CONTROL$“ mit *(siehe Listing 13)*.

Anmerkung: Die Präferenz „TABLE_CACHED_BLOCKS“ wird bei einem „CREATE INDEX“- oder „INDEX REBUILD“-Kommando nicht berücksichtigt *[7, 8]*. Diese Problematik wird momentan im Oracle-Bug 28292026 adressiert. Sollte man also die Präferenz „TABLE_CACHED_BLOCKS“ einsetzen und Indizes mit „CREATE“ oder „REBUILD“ erstellen und ist der Patch für Bug 28292026 noch nicht verfügbar, so empfiehlt es sich, im Anschluss die Statistiken mit „DBMS_STATS.GATHER_INDEX_STATS“ zu sammeln, etwa durch das Ausschalten des automatischen Statistik-Sammelns beim Index-Anlegen und anschließenden „DBMS_STATS“-Aufruf *(siehe Listing 14)*.

Anmerkung: Interne Oracle-Parameter (Parameter, die mit einem Unterstrich beginnen) sollten nur nach Absprache mit dem Oracle-Support eingesetzt werden. Obiger Parameter „_optimizer_compute_index_stats“ ist jedoch unkritisch, da der Support ihn in einigen Artikeln selbst als Lösung anbietet (etwa MOS-Note 793585.1). Auf der anderen Seite beträgt der Overhead des Statistik-Sammelns beim Index-Erstellen nur wenige Prozentpunkte (in Tests um zwei bis vier Prozent längere Erstellungszeit).

Referenzen

[1] *https://mwidlake.wordpress.com/2009/11/18/depth-of-indexes-on-vldbs/*
[2] *https://richardfoote.wordpress.com/2013/05/08/important-clustering-factor-calculation-improvement-fix-you/*
[3] *https://richardfoote.wordpress.com/2013/05/14/clustering-factor-calculation-improvement-part-ii-blocks-on-blocks/*
[4] *https://richardfoote.wordpress.com/2013/06/04/clustering-factor-calculation-improvement-part-iii-too-much-rope/*
[5] *https://jonathanlewis.wordpress.com/2011/05/19/clustering_factor/*
[6] *https://jonathanlewis.wordpress.com/2013/05/09/clustering_factor-2/*
[7] *https://jonathanlewis.wordpress.com/2018/07/02/clustering_factor-5/*
[8] *https://richardfoote.wordpress.com/2018/07/17/rebuilding-indexes-danger-with-clustering-factor-calculation-chilly-down/*
[9] *https://en.wikipedia.org/wiki/Phrases_from_The_Hitchhiker's_Guide_to_the_Galaxy*

Clemens Bleile
clemens.bleile@dbi-services.com

Alternative Fakten – das Unbeeinflussbare beeinflussen

Stefan Winkler, merlin.zwo InfoDesign GmbH & Co. KG

Du hast keine Chance – also nutze sie! Was tun, wenn plötzliche Performance-Einbrüche oder fehlerhaftes SQL den Anwendungsbetrieb lahmlegen oder eine Migration gefährden und zudem vorgegebene Restriktionen scheinbar alle Lösungsmöglichkeiten verhindern? Der Artikel schildert verschiedene solcher Szenarien aus der Praxis und zeigt mögliche Lösungen.

Eine typisches, allen sicherlich bekanntes Szenario aus dem Oracle-Alltag: An einem Montagmorgen läuft die zentrale Anwendung nur noch in Zeitlupe und alle Beteiligten versichern glaubhaft, dass sie rein gar nichts geändert haben. Die eigene IT-Abteilung kann jedoch aus dem AWR schnell das verantwortliche Statement ermitteln, prüft auf etwaige Änderungen des Execution Plan, der Indizierung, Parametrisierung etc. Gemeinsam mit dem gerade verfügbaren externen Oracle-Experten wird ein getuntes Statement erarbeitet und der Einbau mit dem Software-Hersteller für das Produktiv-Deployment am Abend abgestimmt. Unrealistisch? Natürlich!

Rein technisch lassen sich die üblichen potenziellen Problemfelder wie in *Abbildung 1* einteilen. Manche Bereiche lassen sich bei Problemen nur mit viel Aufwand und dem Einverständnis aller Beteiligten verändern (etwa Oracle-Versionen oder Patches); bei anderen sind der Aufwand und/oder die Restriktionen dagegen wesentlich kleiner (beispielsweise Statistiken). Dieser Artikel schildert in der Praxis angetroffene Situationen und dafür gefundene Lösungen bei Performance- und Deployment-Problemen, sowohl für den kooperativen Fall (man kommt an den Code heran und darf ihn ändern) als auch für die verschärfte Variante, dass man an die Ursache (wie den Source-Code) nicht herankommt und/oder ihn nicht ändern darf. Die Frage lautet also: Welche Restriktionen bestehen und wie kann man sie so brechen oder zumindest umgehen, dass eine akzeptable Lösung sowohl für den Endkunden als auch den Software-Lieferanten und die Betriebsverantwortlichen (typischerweise die IT-Abteilung) gefunden werden kann?

Erforderliche Indizes – wenn die Applikation zusätzliche Objekte eliminiert

Index-Probleme bei Migrationen sowie im Betrieb gehören sicherlich zu den häufigsten Themen. Für die gängigsten Aufgaben gibt es in der Oracle-Dokumentation und im Web viele Vorschläge für folgende Fälle:

- *Größe und I/O*
  Basic/Advanced Compression
- *Selektivität*
  Reverse Columns, Column Compression
- *Performance*
  Partitionierung, „optimizer_%"-Parameter etc.

Was aber tun, wenn ein Software-Hersteller sein eigenes Data Dictionary mit den von ihm erwarteten und gepflegten Objekten führt? Noch verheerender, wenn er bei der Migration sein Data Dictionary überprüft, alles „Ungewollte = Nicht-Eigene" gnadenlos eliminiert und dieser Code nicht deaktiviert werden kann oder darf? Im konkreten Fall wurde bei der Probe-Migration festgestellt, dass der Software-Hersteller seine Befehle lediglich mit geringen Datenmengen und nicht mit den beim Kunden vorhandenen Mengen getestet hatte. Einige Updates und Deletes liefen ohne passende Indizes mehr als vierzig Stunden lang und sprengten damit die geplante Wochenend-Downtime. Dank AWR könnten zwar die erforderlichen Indizes bestimmt und angelegt werden, aber die Prüfung der vorhandenen Objekte durch den Hersteller am Anfang der Migration würde sie leider direkt wie-

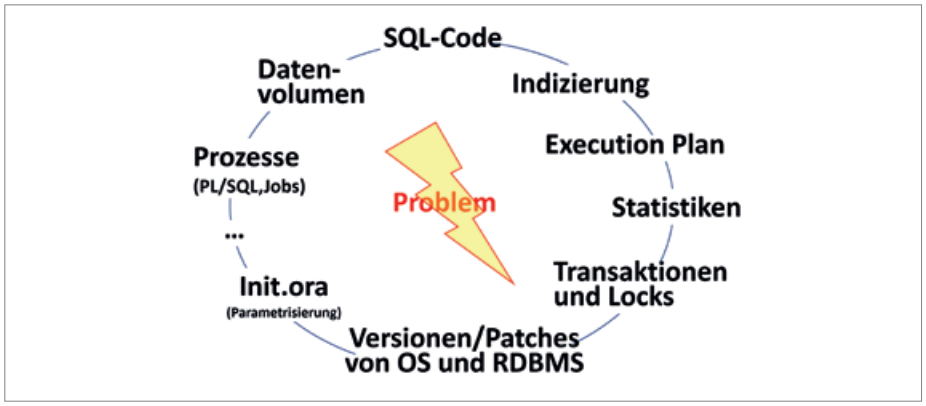


Abbildung 1: Typische Bereiche bei Oracle-Problemen

der entfernen. Im konkreten Fall wurde mit Tolerierung des Herstellers folgende Gegenmaßnahme ergriffen:

- Die erforderlichen Indizes wurden vor der Migration in einem anderen Schema angelegt und dann als „INVISIBLE" gekennzeichnet *(siehe Listing 1)*
- Nach dem Check vom DBA wieder auf „VISIBLE" geändert

Der Check des Software-Herstellers verlief nun erfolgreich, da der Vergleich der erwarteten Objekte mit den tatsächlich im Schema vorhandenen Objekten die „fremden" Indizes nicht sehen konnte. Eine einfache Lösung mit voller Wirkung. Hinweis: Für diese Lösung ist eine Oracle Standard Edition ab Version 11g R1 ausreichend.

Überlisten von SELECT und DML

Sofern eine Tabelle eine sehr asymmetrische Datenverteilung hat (also aktive und passive Daten) und sich Dialogmasken mit Zugriffen auf aktive Daten mit Reporting-Abfragen und Batches auf passive Daten mischen, können diese konkurrierenden Anforderungen in der Regel durch Partitionierung gelöst werden. Sollte die Partitioning-Option jedoch nicht zur Verfügung stehen und der Software-Hersteller kooperieren, lässt sich auch eine manuelle Lösung implementieren, die für die Anwendung völlig transparent ist *(siehe Abbildung 2)*.

Neben dem PL/SQL-Code des Triggers zur statusabhängigen Pflege bei Insert/Updates/Deletes in der passenden Tabelle sollten zur Absicherung der Konsistenz zusätzlich die Constraints wie in *Listing 2* angelegt werden. Der Optimizer kann beim „SELECT" mit dem Prädikat „status" performant umgehen: Er filtert den Zugriff auf die nicht passende Tabelle mit „NULL IS NOT NULL" weg *(siehe Abbildung 3)*. Hinweis: Diese Lösung funktioniert mit einer Oracle Standard Edition (bis zur Version 8.1.6 waren die „INSTEAD OF"-Trigger ein Feature der Enterprise Edition); getestet mit 12.1.

```
SYSTEM> CREATE INDEX SYSTEM.migspezial_insupd_idx
ON applownerschema.tablename (columns…) INVISIBLE;
```

Listing 1

```
ALTER TABLE T_a ADD CONSTRAINT T_a_status_chk CHECK (status='aktiv');
ALTER TABLE T_p ADD CONSTRAINT T_p_status_chk CHECK (status='passiv');
```

Listing 2

Austausch der Execution-Pläne

Im nächsten Beispiel traf der Autor bei einem Kundeneinsatz auf eine besonders restriktive Situation im Finance-Umfeld. Beim Update auf eine neue Version der Anwendung wurden einige zusätzliche Indizes durch den Hersteller der Standard-Software angelegt. Diese verbesserten zwar einige SQL-Befehle, allerdings gab es einige wenige, die nun mit Faktor 100 langsamer liefen („Nested Loops" auf sehr großen Datenmengen und massivem Einzelsatz „Table by Index ROWID"-Zugriff). Alle Änderungen in der Datenbank wa-

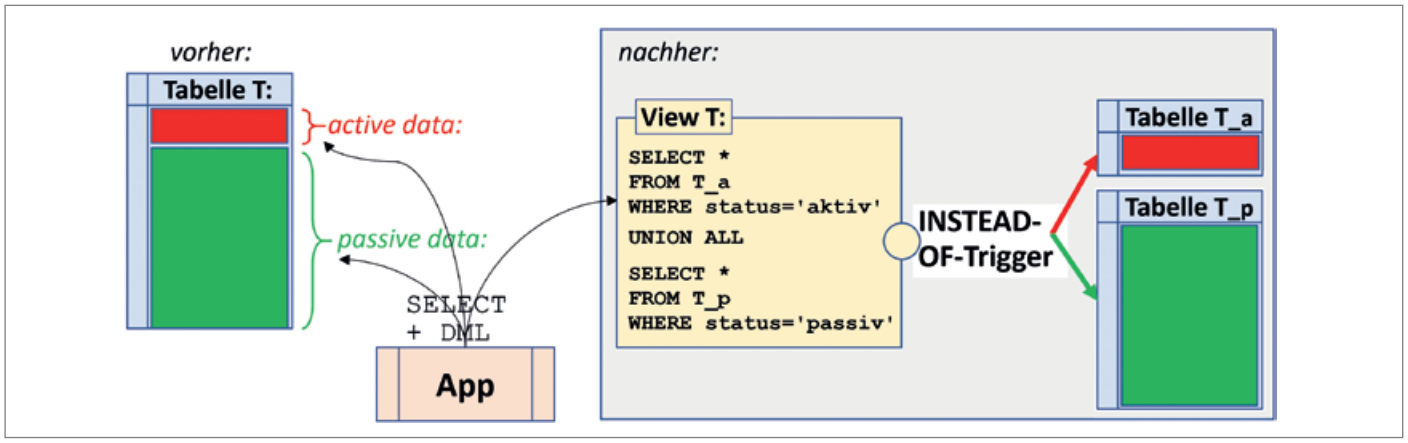


Abbildung 2: Konzept einer „UNION ALL"-View mit „INSTEAD OF"-Trigger

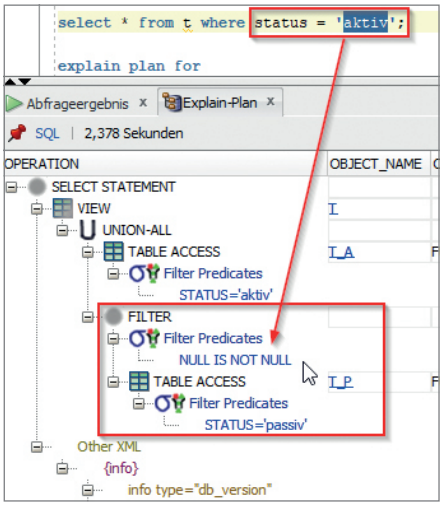


Abbildung 3: Der Optimizer erkennt über das Prädikat die zu filternde Tabelle

ren verboten; eine optimierte Version der Software würde frühestens im folgenden Quartal verfügbar sein. Was tun?

Oracle stellt drei Mechanismen zur Verfügung, um Änderungen an Ausführungsplänen vorzunehmen: SQL-Profiles, SQL-Baselines und SQL-Patches. Der Tuning Advisor schlägt SQL-Profiles vor, basierend meist auf der Anpassung der Kardinalitäten an die Realität. Baselines ermöglichen es, eine Liste der akzeptierten Ausführungspläne für eine Anweisung zur Verfügung zu stellen. SQL-Patches sind Teil des SQL Repair Advisor und fügen „HINTS“ zu einer bestimmten Anweisung hinzu.

Da der Kunde über die passende Oracle-Lizenzierung verfügte, konnte frei entschieden werden, wie der Fake mit alternativen Fakten direkt auf der Ebene des Execution Plan durchgeführt werden soll. Die Wahl fiel auf die sehr mächtigen „SQL Profiles“, die überaus komplex werden können; im Web gibt es zahlreiche Artikel dazu. Nachfolgend die kleine, schlanke Lösung für die Aufgabe „Ersetze für einen Befehl X dessen Execution Plan mit einer getunten Variante“ – für den Software-Anbieter und seine Anwendung absolut transparent, da unsichtbar. *Listing 3* zeigt den vereinfachten Code *(siehe Abbildungen 4 bis 7)*.

Manipulation des SQL mit Translation Framework

Das Beispiel zeigt einen Weg, um den Execution Plan eines gegebenen SQL-Statements zu „fälschen“. Wie sieht es aber aus, wenn nicht nur der Plan, sondern der gesamte SQL-Befehl, also sein Code an sich, verändert werden soll, und zwar so, dass es für die Applikation völlig transparent bleibt? Hier bietet Oracle erneut mehrere Möglichkeiten, um SQL-Code durch einen anderen SQL-Code auszutauschen:

- DBMS_ADVANCED_REWRITE: ab Version 10g; hat jedoch einige unangenehme Restriktionen
- SQL Translation Framework: ab Version 12c; ändert Code, noch bevor er überhaupt analysiert wird. Dies kann die Ausführung von „falschem“ SQL bewirken oder eben „Falsches zu Richtigem“ korrigieren
- SQL-Patch („dbms_sqldiag internal.i_create_patch“): ähnlich wie „DBMS_ADVANCED_REWRITE“, vielleicht etwas leistungsfähiger, aber selbst in Version 18 nicht dokumentiert *(siehe „https://docs.oracle.com/en/database/oracle/oracle-database/18/arpls/DBMS_SQLDIAG.html#GUID-37E72B14-17BB-47E1-9EA4-1EA1DE823867“)*

Wie bei den genannten Beispielen ergab sich bei einem Kunden die Anforderung, während einer Migration mit „alternativen Fakten“ einzugreifen: In einem kompilierten, unveränderlichen Migrationsprogramm („.exe“) wurde eine „WHERE“-Bedingung nicht ausreichend qualifiziert, sodass falsche Ergebnisse entstehen konnten. Leider war das Executable vom Hersteller nicht kurzfristig änderbar; in diesem Fall wurde nach Absprache mit allen Beteiligten das SQL durch das SQL-Translation-Framework manipuliert. Dazu waren einige Voraussetzungen zu schaffen (*Listing 4* zeigt ein vereinfachtes Beispiel).

Sobald der AppOwner in diesem Beispielsetup den Befehl „Update mytable set migstatus = 1;“ ausführt, erhält er zwei Zeilen aktualisiert – so weit, so normal. Jetzt wird das Translation Profile aktiviert *(siehe Listing 5)*. Anschließend werden mit der Variante des SQL-Befehls „Update mytable set migstatus = 1“ weitere zwei Zeilen aktualisiert. Die Variante „UPDATE mytable SET migstatus = 1;“ aktualisiert hingegen nur noch eine Zeile. Oracle tauscht also vor der Ausführung das SQL nur dann aus, wenn der Code absolut identisch mit der Übersetzungs-Definition ist.

Ein Blick ins dynamische Data Dictionary zeigt, dass die SQL Translation aktiv ist und auch der „HASH_VALUE“ des SQL-Befehls umgemappt wird. Der

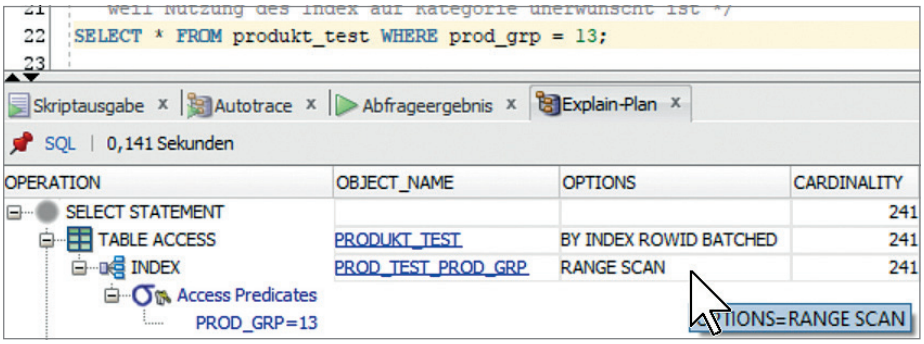


Abbildung 4: Das Original-SQL

```
CREATE table produkt_test AS
SELECT 1000000+rownum                          AS prod_nr,
       sysdate-trunc(dbms_random.value(0,90)) AS datum,
       trunc(dbms_random.value(0,25))          AS personal_nr,
       trunc(dbms_random.value(0,75))          AS prod_grp,
       'asjdhfösdfhasöfhasöfhasdögihgaöoghi'   AS kommentar
  FROM dual
CONNECT BY level <= 1000000
;
CREATE INDEX prod_test_prod_grp on produkt_test (prod_grp)
;
```

Listing 3

```
WHERE sql_fulltext LIKE '%FROM produkt_test WHERE prod_grp = 13';
-- Ergebnis: SQL_ID=4x9an5guyavpk

-- getuntes SQL:
SELECT /*+ full(x) */ * FROM produkt_test x WHERE prod_grp = 13;

SELECT *
  FROM v$sql
WHERE sql_fulltext LIKE '%FROM produkt_test x WHERE prod_grp = 13%';
-- Ergebnis: SQL_ID=85m0ryw0dfjyw

SELECT * FROM TABLE(dbms_xplan.display_cursor('85m0ryw0dfjyw', 0, 'outline'));
```

Skriptausgabe | Autotrace | Explain-Plan | Abfrageergebnis

SQL | Alle Zeilen abgerufen:32 in 0,156 Sekunden

```
PLAN_TABLE_OUTPUT
SQL_ID  85m0ryw0dfjyw, child number 0
-------------------------------------
SELECT /*+ full(x) */ * FROM produkt_test x WHERE prod_grp = 13

Plan hash value: 3973325831

-------------------------------------------------------------------------------
| Id  | Operation         | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |              |       |       |  2533 (100)|          |
|*  1 |  TABLE ACCESS FULL| PRODUKT_TEST |   241 | 14701 |  2533   (1)| 00:00:01 |
-------------------------------------------------------------------------------

Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('12.1.0.2')
      DB_VERSION('12.1.0.2')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$1")
      FULL(@"SEL$1" "X"@"SEL$1")
      END_OUTLINE_DATA
  */
```

Abbildung 5: Fälschung vorbereiten

```
-- Als DBA mit Rechten für die dbms_sqltune-Package nun dem DELETE das SQL-Profile unterschieben..
DECLARE
    v_sql_id    VARCHAR2(30) := '4x9an5guyavpk'; -- SQL-ID des Originals
    v_clsql_text CLOB;
BEGIN
    SELECT sql_fulltext
      INTO v_clsql_text
      FROM v$sqlarea
     WHERE sql_id = v_sql_id;

    dbms_sqltune.import_sql_profile(
        sql_text    => v_clsql_text,
        profile     => sqlprof_attr('FULL(@SEL$1 X@SEL$1)'),
        name        => 'PROFILE_' || v_sql_id,
        force_match => true
    );
END;
/

-- Nachschauen, ob es korrekt angelegt wurde
SELECT * FROM dba_sql_profiles;
```

Skriptausgabe | Abfrageergebnis

SQL | Alle Zeilen abgerufen:1 in 0,047 Sekunden

	NAME	CATEGORY	SIGNATURE	SQL_TEXT
1	PROFILE_4x9an5guyavpk	DEFAULT	2485911664915078956	SELECT * FROM produkt_test WHERE prod_grp = 13

Abbildung 6: Fälschung durchführen (alternative Fakten = der andere Plan)

```
/* Nun test, ob der Fake klappt... */
SELECT * FROM produkt_test WHERE prod_grp = 13;
```

Skriptausgabe | Autotrace | Abfrageergebnis | Explain

SQL | 0,063 Sekunden

ERATION	OBJECT_NAME	OPTIONS	CAR
SELECT STATEMENT			
TABLE ACCESS	PRODUKT_TEST	FULL	

- Filter Predicates
  - PROD_GRP=13
- Other XML
  - {info}
    - info type="db_version"
      - 12.1.0.2
    - info type="parse_schema"
      - "BLMGR"
    - info type="plan_hash_full"
      - 1034621722
    - info type="plan_hash"
      - 3973325831
    - info type="plan_hash_2"
      - 1034621722
    - info type="sql_profile" note="y"
      - "PROFILE_4x9an5guyavpk"
    - {hint}
      - FULL(@"SEL$1" "PRODUKT_TEST"@"SEL$1")
      - OUTLINE_LEAF(@"SEL$1")
      - ALL_ROWS

Abbildung 7: Fälschungserfolg überprüfe

Execution Plan wird also für den ausgetauschten SQL-Befehl erstellt und ausgeführt *(siehe Abbildung 8)*. Da in dem konkreten Szenario die Änderung des Executable nicht möglich war, war auch das interaktive Absetzen der zwei notwendigen „ALTER SESSION"-Befehle nicht möglich. Die Lösung dieses Problems gelingt jedoch mit einem einfachen Datenbank-Trigger *(siehe Listing 6)*. Einschränkungen bei der Ersetzung von SQL-Texten gibt es nur wenige:

- Die Bind-Variablen müssen hinsichtlich Reihenfolge, Alias und Typen identisch sein; wobei gilt, dass die Anzahl im Zielstatement größer oder gleich der Anzahl im ursprünglichen Statement sein muss.
- Die Ergebnis-Struktur (Anzahl, Reihenfolge und Typ der Ergebnis-Spalten) muss unverändert bleiben.

Das SQL-Translation-Framework ist von Oracle ursprünglich gedacht gewesen, um Anwendungen von anderen Datenbanken-Herstellern (MS SQL Server, Sybase, IBM DB2) auf Oracle-Syntax zu konvertieren. Während der Entwicklung hat sich das Team jedoch – erfreulicherweise – dazu entschlossen, die Möglichkeiten zu öffnen und die Übersetzung als Datenbank-Feature mit API zu implementieren. Hinweis: Für diese Lösung mit dem SQL-Translation-Framework ist die Oracle-Version 12gR1 erforderlich.

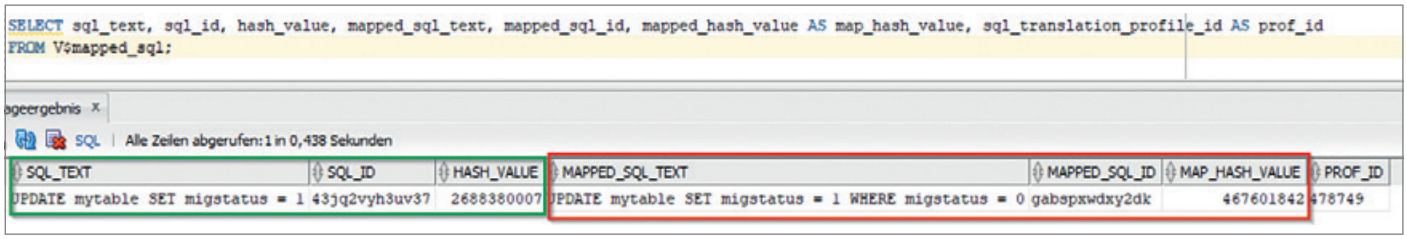


Abbildung 8: SQL-Translation im Data Dictionary

```
BEGIN -- als SYS das Profil und den „Fake-Code" anlegen
   dbms_sql_translator.create_profile(
       profile_name => 'MIGRATION_MYAPP')
   ;
   dbms_sql_translator.register_sql_translation(
       profile_name    => 'MIGRATION_MYAPP',
       sql_text        => 'UPDATE mytable SET migstatus = 1',
       translated_text => 'UPDATE mytable SET migstatus = 1 WHERE mig-
status = 0',
       enable          => TRUE)
   ;
END;
/

GRANT alter session to AppOwner;
GRANT all on sql translation profile MIGRATION_MYAPP to AppOwner; -- das
SQL Translation Profil dem Owner zuordnen

CREATE TABLE AppOwner.mytable
(irgendein_inhalt VARCHAR2(10), migstatus NUMBER(1,0), lzt_aeanderung
DATE);

CREATE TRIGGER AppOwner.mytable_lzt_aend_trg
   BEFORE INSERT OR UPDATE ON mytable FOR EACH ROW
BEGIN
   :new.lzt_aeanderung := sysdate;
END mytable_lzt_aend_trg;
/

INSERT into AppOwner.mytable VALUES ('unmigriert',0,NULL);
INSERT into AppOwner.mytable VALUES ('migriert',1,NULL);
```

Listing 4

```
ALTER SESSION set sql_translation_profile = sys.MIGRATION_MYAPP;
ALTER SESSION set events ='10601 trace name context forever, level 32';
```

Listing 5

```
CREATE OR REPLACE TRIGGER sql_translation_trg
   AFTER LOGON ON DATABASE
BEGIN
   IF USER IN ('APPOWNER') THEN
      execute immediate 'ALTER SESSION set sql_translation_profile =
sys.MIGRATION_MYAPP';
      execute immediate 'ALTER SESSION set events =''10601 trace name
context forever, level 32''';
    END IF;
END;
/
```

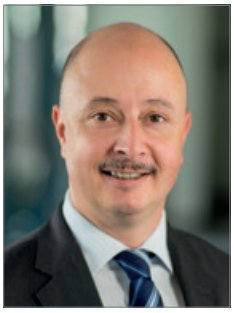
Listing 6

Fazit

Es gibt viele Möglichkeiten, mit mehr oder weniger aufwendigen und abstimmungsintensiven Mitteln an der Performance-Schraube zu drehen – auch wenn es scheinbar keine Chance gibt. Mit dem SQL-Translation-Framework besteht sogar die Möglichkeit, den SQL-Code selbst zu manipulieren.

Verweise

- SQL Profiles, Oracle White Paper: *https://docs.oracle.com/database/121/TGSQL/tgsql_profiles.htm*
- DBMS_ADVANCED_REWRITE (PL/SQL Packages and Types Reference): *https://docs.oracle.com/en/database/oracle/oracle-database/18/arpls/DBMS_ADVANCED_REWRITE.html#GUID-77CBA65E-FEC9-4E87-8846-CA1A7D065878*
- Oracle Database, SQL Translation and Migration Guide: *https://docs.oracle.com/en/database/oracle/oracle-database/18/drdaa/sql-translation-and-migration-guide.pdf*
- PL/SQL Packages and Types Reference, DBMS_SQL_TRANSLATOR: *https://docs.oracle.com/en/database/oracle/oracle-database/18/arpls/DBMS_SQL_TRANSLATOR.html#GUID-80ADAC71-714F-4B91-A7F2-0F0063AC3499*
- Kerry Osborne's Oracle Blog, SQL Translation Framework: *http://kerryosborne.oracle-guy.com/2013/07/sql-translation-framework/*



Stefan Winkler
stefan.winkler@merlin-zwo.de

Das Beste aus zwei Welten

Bruno Cirone, bc-consult

Manchmal muss man eine Tuning-Maßnahme noch vor der Arbeit mit Oracle beginnen. Bei dem hier beschriebenen Projekt werden die Stärken von Unix/Linux und von Oracle zusammengeführt und dies führt dann zu erstaunlichen Ergebnissen.

Bei einem Projekt eines großen europäischen Stahlkonzerns geht es darum, dass eine große Menge an Daten (ca. 4 Millionen Sätze) in regelmäßigen Abständen vom Hostsystem an die untergeordneten Systeme (ca. 400 Systeme) übertragen wird. Diese Datei ist unstrukturiert mit festen Satzlängen, im EBCDIC-Format und dazu noch komprimiert. Aus diesen 4 Millionen Sätzen werden lediglich ca. 8.000 bis 20.000 Sätze und nur einige Spalten pro System benötigt.

Die infrage kommenden Sätze müssen entweder in die Tabelle eingetragen werden oder, falls bereits vorhanden, muss der Datensatz geändert und mit einem neuen Kennzeichen versehen werden. Leider konnten die Host-Betreuer nicht dazu bewegt werden, nur die benötigten Sätze zu übertragen. Auf dem Hostsystem ist es etwas aufwendiger, solche Maßnahmen umzusetzen. Also kurz gesagt, wie ein Mitarbeiter des Kunden meinte, „flexibel wie eine Eisenbahnschranke".

Ein wenig Historie dieses Projekts: Im Jahr 1990 wurde es in Cobol mit Oracle 7.3.2 entwickelt. Das Programm lief regelmäßig alle acht Stunden; die Laufzeit betrug etwa zwei Stunden. Zu diesem Zeitpunkt gab es keine External Tables und auch kein Merge-Statement. Für das Laden von Daten hätte man den SQL-Loader („sqlldr") nutzen können, was jedoch nicht gewollt war und den Ansprüchen auch nicht gerecht worden wäre.

Neue Hardware und aktuelle Oracle-Versionen ergaben eine Verbesserung der Laufzeit auf rund 40 Minuten. Die Frequenz wurde dadurch auf vier Stunden reduziert. Während der Laufzeit des Programms ist die Anwendung gesperrt.

Da mittlerweile die Programmierer in den Ruhestand gingen, traute sich niemand mehr an dieses Programm heran. Die Anforderung, die Frequenz zu erhöhen, wurde andererseits immer dringender. Es sollte daher eine Optimierung in der Datenbank vorgenommen werden. Nach eingehender Erklärung des Transaktionskonzepts von Oracle bestand keine Notwendigkeit mehr für einen Stillstand der Anwendung. Diese „Tuning"-Maßnahme griff sofort und es gab keine Stillstände mehr. Damit war allerdings die lange Laufzeit des Programms nicht behoben, Änderungen des Host-Systems dauerten also immer noch sehr lange, bis sie in den Subsystemen komplett verarbeitet wurden.

Erste Tuning-Phase

Nach Analyse des Cobol-Programmes hat der Autor festgestellt, dass die Anforderung mit Unix/Linux und Datenbank-Bordmitteln komplett umgesetzt werden könnte. Es wurde eine Datei mit rund vier Millionen Sätzen mit jeweils 800 Bytes pro Satz und einer Blockgröße von 32000 Bytes bearbeitet. Insgesamt wurden etwa 8.000 Sätze in der Datenbank eingefügt beziehungsweise geändert. Ein einfaches Shell-Skript hat dazu ausgereicht *(siehe Listing 1)*.

Das Ergebnis ist erfreulich: Ein Cobol-Programm komplett durch ein Shell-Skript sowie SQL*Plus ersetzt und dazu die Laufzeit auf ca. 25 Prozent reduziert. Bei den Laufzeiten kann man sehr gut erkennen, dass die Hauptarbeit immer noch von Oracle vorgenommen werden muss. Immerhin muss Oracle ja jeden Datensatz lesen und die infrage kommenden Spalten selektieren und bearbeiten. Weitere Optimierungen sollten vorerst nicht erfolgen. Diese Variante wurde in der Produktion eingesetzt und die Frequenz weiter reduziert. Das Shell-Skript sollte nun alle zwei Stunden laufen.

Zweite Tuning-Phase

Etwa zwei Jahre später kam die Frage auf, ob weitere Tuning-Maßnahmen möglich wären. Auch dabei kommt man mit einem einfachen Shell-Skript zum Erfolg. Bei dieser Variante sind die Befehle „gunzip", „dd", „grep" und „cut" mit „pipe" verbunden *(siehe Listing 2)*.

Eine solche Laufzeitreduktion auf ca. 105 Sekunden war vorher nicht vorstellbar. Bei dem Unix/Linux-Kommando könnte auch zuerst „cut" und danach „grep" vertauscht werden, dies hat aber zur Folge, dass die Laufzeit wieder stark ansteigt *(siehe Listing 3)*.

Der Effekt ist damit zu erklären, dass beim ersten Befehl zunächst alle infrage kommenden Sätze gefiltert werden und danach der zu bearbeitende Satz mit „cut" zusammengestellt wird. Bei der zweiten Variante werden praktisch alle vier Millionen Sätze weitergegeben und danach gefiltert. Trotzdem könnte die zweite Variante sinnvoll sein, wenn viele Sätze an die Datenbank übergeben werden müssen. Die Zwischenergebnisse geben einen guten Anhaltspunkt dafür, welche Reihenfolge sinnvoll ist. Für einen Test reicht eine kleinere Menge (etwa ca. 10.000 Sät-

```
time gunzip transfer_file_ebcdic.gz
real    2m3.305s
time dd if=transfer_file_ebcdic of=transfer_file conv=ascii ibs=32000 obs=800
real    0m43.949s
time sqlplus bc/x1 @/home/oracle/external_table_tuning1.sql
Real    7m37.609s
```

Listing 1

```
time gunzip -c transfer_file_ebcdic.gz |  dd conv=ascii  ibs=32000 obs=800 | grep 2010 |  cut -c1-9,11-12,15-
18,20-29,50-59,70-74 > teil2
real    1m20.205s
time sqlplus bc/x1 @/home/oracle/external_table_tuning2.sql
Real    0m24.609s
```

Listing 2

```
time gunzip -c transfer_file_ebcdic.gz |  dd conv=ascii  ibs=32000 obs=800 | cut -c1-9,11-12,15-18,20-29,50-
59,70-74 | grep 2010   > teil2
real    2m10.100s
```

Listing 3

```
cut -c1-9,11-12,15-18,20-29,50-59,70-74  Testdatei |tee cut.file | grep 2010  > nach_grep.file
grep 2010 Testdatei |tee grep.file | cut -c1-9,11-12,15-18,20-29,50-59,70-74  > nach_cut.file
ls -l Testdatei grep.file cut.file

-rw-rw-rw-   1 user group    8088266 Aug  5 12:06 Testdatei
-rw-rw-rw-   1 user group     414510 Aug  5 12:06 cut.file
-rw-rw-rw-   1 user group      40190 Aug  5 12:06 grep.file
```

Listing 4

```
time sqlplus bc/x1 @/home/oracle/external_table_tuning3.sql
Real    0m48.201s
```

Listing 5

```
/usr/bin/zcat $1 | /usr/bin/dd conv=ascii 2>/dev/null \
                 | /usr/bin/grep 2010     2>/dev/null \
                 | /usr/bin/cut -c1-9,11-12,15-18,20-29,50-59,70-74
2>/dev/null
```

Listing 6

```
CREATE TABLE Fertigung
   (
    Satz_id     number(10),
    Satzart     varchar2(1),
    Werk        varchar2(4),
    Teile_id    varchar2(10),
    Kunde_id    varchar2(10),
    Menge       number(5)
   )
   ORGANIZATION EXTERNAL
    (
      TYPE oracle_loader
      DEFAULT DIRECTORY ext_tables
      access parameters
             (
             records delimited by newline
             preprocessor ext_tables:'bc.sh'
             fields (
                      Satz_id    position ( 1: 9) char(10),
                      Satzart    position (10:11) char(1),
                      Werk       position (12:15) char( 4),
                      Teile_id   position (16:25) char(10),
                      Kunde_id   position (26:35) char(10),
                      Menge      position (36:40) char(5)
                    )
             )
      LOCATION ('transfer_file_ebcdic.gz')
    )
   REJECT LIMIT UNLIMITED
;
```

Listing 7

ze) aus, um eine gesicherte Aussage treffen zu können. Zwischenergebnisse lassen sich wie in *Listing 4* einfach ermitteln. Die entsprechende Frequenz wurde wieder weiter reduziert, das Shell-Skript lief jetzt jede Stunde.

Dritte Tuning-Phase

Bis dahin war für den Autor klar, dass für External Tables, wie der Name schon sagt, eine wirkliche Datei als Input existieren muss. Er fand jedoch, dass dieses Problem mit einer Pipe oder Named-Pipe lösbar sein müsste. Während des Besuchs der Real-World Performance Tour 2015 in Wien konnte er das Problem mit Tom Kyte von Oracle besprechen. Er sagte, dass die External Tables auch eine Ausgabe aus der Pipe sein dürfe. Solange die Daten sequenziell angeliefert werden, sei es unerheblich, ob reale Datei oder Pipe. Mit diesem Wissen ließ sich der Prozess weiter optimieren. Zum Schluss blieb noch ein Befehl übrig *(siehe Listing 5)*.

Das Ergebnis ist überzeugend. Von ehemals 40 Minuten auf ca. 50 Sekunden, das sind nur noch etwa zwei Prozent der Laufzeit gegenüber früher. Um dies zu erreichen, sind ein paar Schritte notwendig. Zunächst ist ein Directory zu erstellen, in dem die Eingabedatei (etwa 'transfer_file_ebcdic.gz') hinterlegt wird: „Create Directory ext_tables as '/home/oracle/external_table';".

Dann ist ein Preprocessor-Skript zu erstellen, das die Unix-Kommandos enthält. Wichtig ist dabei, dass die Unix-Kommandos immer mit vollem Pfadnamen eingetragen sind. Die Fehlermeldungen

```
Merge  into locale_fertigung LF using fertigung F
       on (LF.Satz_id = F.Satz_id)
when matched then
       update set lf.satzart=F.Satzart, lf.Menge=F.Menge
when not matched then
       insert   (satz_id, satzart, werk, teile_id, kunde_id, menge)
       values   (f.satz_id, f.satzart, f.werk, f.teile_id, f.kunde_id,
f.menge)
where F.werk = '2010'
;
```

Listing 8

müssen umgeleitet werden, beispielsweise „2>/dev/null“. In diesem Fall lautet der Name des Preprocessor-Skriptes „bc.sh“ *(siehe Listing 6)*. *Listing 7* zeigt das Erzeugen des „Create Table“ mit den erstellten Informationen über Directory und Preprocessor-Skript. In *Listing 8* ist das Merge-Statement zu sehen.

Ab diesem Zeitpunkt wurde das SQL-Skript nach Bedarf ausgeführt. Nur wenn eine aktuelle Datei angeliefert worden ist, erfolgt eine Verarbeitung. Damit war praktisch jeder Zeitverzug eliminiert.

Fazit

Dieses Verfahren hat sich auch bei vielen anderen Projekten bewährt, so konnte die Laufzeit von Fremddaten-Übernahmen, Laden von DWH-Daten, Zusammenführung verschiedener Eingabedateien etc. wesentlich verbessert werden. Der hauptsächliche Performance-Gewinn kommt daher, dass fast alles im Memory stattfindet und keine Zwischenmengen erzeugt werden, die immer wieder gelesen und geschrieben werden müssen.

Es ist wichtig, die jeweiligen Stärken des Betriebssystems und der Datenbank zu kennen, damit wie in diesem Projekt ein optimales Ergebnis erzielt werden kann. Häufig können Vorarbeiten (etwa mit „awk“, „sed“, „egrep“ etc.) sehr effizient im Betriebssystem vorgenommen werden, was dann zu einer besseren Lade-Laufzeit bei der Datenbank führt.

Bruno Cirone
bruno@cirone.de

Das DOAG Legal Council expandiert

Mit Dr. Ivo Rungg von der Kanzlei Binder Grösswang aus Wien/Innsbruck ist jetzt auch ein Vertreter aus Österreich im DOAG Legal Council aktiv. Die Schwerpunkte von Ivo Rungg sind IP- und IT-Recht sowie Datenschutz mit Themen wie geistiges Eigentum, Lizenz- und Kaufvereinbarungen sowie Datenschutzrecht. Das DOAG Legal Council freut sich, dass Dr. Ivo Rungg hier die rechtlichen Belange der österreichischen Anwender aufnehmen und bewerten kann.

Das DOAG Legal Council ist ein Gremium aus spezialisierten, unabhängigen Rechtsanwälten, das der DOAG in rechtlichen Angelegenheiten zur Seite stehen soll und ihrem Netzwerk fundiertes Fachwissen zur Verfügung stellt. Ob Vertragsgestaltung, Lizenzierung oder Datenschutz – in vielen Bereichen der IT spielen juristische Aspekte eine wesentliche Rolle. Doch das Thema „IT-Recht“ ist für viele IT-Spezialisten schwer zugänglich. Das DOAG Legal Council verrichtet seine Arbeit ehrenamtlich. Es hat sich als Ziel gesetzt, fundiertes Wissen zu rechtlichen Aspekten der IT im DOAG-Netzwerk zur Verfügung stellen, zum Beispiel in Form von Fachartikeln oder Fachvorträge in den eigenen Medien bzw. Veranstaltungen.

Weitere Informationen unter *„https://www.doag.org/de/themen/competence-center/legal-council“*.

Vom Nutzen der Best Practices

Jürgen Sieben, ConDeS GmbH & Co. KG

Immer wieder wird auf die Bedeutung der Einhaltung von Best Practices verwiesen. Die Ermunterungen sind wohlfeil und – seien wir ehrlich – ermüdend. Natürlich verstehen wir alle, warum Best Practices eingeführt wurden, wir verstehen auch deren Bedeutung für die Teamarbeit und dass sie eingehalten werden müssen. Aber sie erscheinen manchmal unnötig aufwendig. Diese Folge der Kolumne beleuchtet den Vorteil einer konkreten Best Practice, allerdings kann der Autor, um die Spannung nicht gleich herauszunehmen, noch nicht verraten, um welche.

Noch steht nicht die Empfehlung im Blickpunkt, sondern ein Stück Code, das ein sehr merkwürdiges Verhalten zeigte. Der Code lief monatelang einwandfrei und auf einmal warf er einen eher suspekt wirkenden Fehler. *Listing 1* zeigt den (im Grundsatz trivialen) Code.

Eine Prozedur vereinbart einen Cursor gegen eine Remote-Datenbank und iteriert über die Ergebniszeilen, um mit diesen Zeilen irgendeine Arbeit durchzuführen. Sieht harmlos aus, oder? Der Code funktionierte, wie gesagt, monatelang auch problemlos, doch dann erschien eine Fehlermeldung, die sich anschließend immer wieder zeigte *(siehe Listing 2)*.

```
SQL> create or replace procedure do_something
   2 as
   3   cursor my_cur is
   4     select *
   5       from some_view@dblink v
   6       join some_table t
   7         on v.id = t.id;
   8 begin
   9   for rec in my_cur loop
  10     -- do_something;
  11     null;
  12   end loop;
  13 end;
  14 /
```

Listing 1: Test-Prozedur

Nanu? Zugegeben: Dieser Fehler gehört definitiv in die Kategorie „Wer weiß denn so was?“, er ist aber auch nicht aus der Welt und kann so oder sehr ähnlich immer wieder mal auftreten. Zunächst einmal ist der Code, bis auf den Datenbank-Link in der Deklaration des Cursors, ganz normal. Da andererseits das Öffnen des Cursors bereits den Fehler wirft, muss das Problem irgendwo in dieser Ecke liegen. Auch die Fehlermeldung „Bulk Bind“ weist hierauf hin, denn seit Version 11g der Datenbank wird das Öffnen eines Cursors in einer „cursor for“-Schleife durch eine Bulk-Operation optimiert (wofür der ungewollte Fehler also gleich einen schönen Beleg liefert).

```
ORA-06502: PL/SQL: numerischer oder Wertefehler: Bulk Bind: Truncated Bind
ORA-06512: in "DOAG.DO_SOMETHING", Zeile 9
```

Listing 2

Erste Anlaufstelle: die Oracle Dokumentation. Dort erfährt man, dass dieser Fehler auftritt, wenn ein „type mismatch“ vorliegt, also zum Beispiel innerhalb einer Bulk-Operation eine Variable mit einer Länge von 10 mit 11 Zeichen beladen wird. Das kann hier jedoch ganz offensichtlich nicht das Problem sein, denn der Record, der eine Zeile des Cursors aufnimmt, wird implizit als „my_cur%ROWTYPE“ definiert, entspricht also exakt der Definition des Cursors. Woher die Deklaration des Typs kommt, ist auch schnell geklärt: aus dem Data Dictionary nämlich, aus den Tabellen-Eigenschaften der zugrunde liegenden Tabellen.

Um den Fehler einzugrenzen, sollte man zunächst ermitteln, ob es irgendwelche Probleme mit den Daten des Cursors gibt. Allerdings funktioniert die „select“-Anweisung des Cursors, für sich ausgeführt, ohne Probleme. Nur im Zusammenhang mit der Prozedur wird konstant der oben gezeigte Fehler geworfen. Die nächste Aktion, um den Fehler einzugrenzen, besteht darin, die Spalte zu finden, die den Fehler wirft. Dies ist einfach, man muss nur die Cursor-Deklaration so ändern, dass der Fehler kontrolliert geworfen oder verhindert werden kann. Im Beispiel fand sich schließlich eine Zeile als Übeltäter, die einen Kommentar zu einem Dokument enthielt.

Doch was war so speziell an dieser Spalte? Vielleicht liegt es an der unüblichen Verwendung von „LONG“-Datentypen oder ähnlichen, doch in diesem Datenmodell wird „LONG“ ausschließlich noch von Oracle selbst verwendet, nicht aber von den Datenbank-Modellierern. Die Spalte war schlicht vom Typ „varchar2(40 byte)“. Warum Byte? Es mag zunächst nicht so scheinen, aber diese Frage bringt uns auf die richtige Fährte. Alternativ hätte dort „varchar2(40 char)“ stehen können.

```
-- In entfernter Datenbank
SQL> create table encoding_test(
   2   id number,
   3   text varchar2(10 byte),
   4   constraint pk_encoding_test primary key(id)
   5 ) organization index;
SQL> table ENCODING_TEST erstellt.

SQL> insert into encoding_test(id, text)
   2 select 1, 'Zehn Ohne:' from dual union all
   3 select 2, 'Zehn mit Ö' from dual;
2 Zeilen eingefügt.

SQL> commit;
festgeschrieben.
```

Listing 3

```
SQL> -- Test fuer Zeile 1
SQL> call encoding_test(1);

encoding_test 1) erfolgreich.

SQL> -- Test fuer Zeile 2
SQL> call encoding_test(2);
Fehler beim Start in Zeile : 16 in Befehl - call encoding_test(2)

Fehlerbericht -
SQL-Fehler: ORA-06502: PL/SQL: numerischer oder Wertefehler: #
                       Bulk Bind: Truncated Bind
ORA-06512: in "DOAG.ENCODING_TEST", Zeile 9
06502. 00000 -  "PL/SQL: numeric or value error%s"
```

Listing 4

Unterschied zwischen Byte- und Char-Semantik

Bei Zeichensatz-Kodierungen mit variabler Länge pro Zeichen (hier ist eigentlich immer UTF-8 gemeint) besteht durchaus ein Unterschied zwischen der Angabe „Byte“ oder „Char“, bei Zeichensatz-Kodierungen aus dem ISO-8859-Umfeld jedoch nicht, weil dort jeder Buchstabe genau ein Byte lang ist. Der Unterschied besteht darin, dass im Fall einer UTF-8-basierten Datenbank die Datenbank für eine Spalte vom Typ „varchar2(40 char)“ 160 Byte Speicherplatz vereinbart, weil ein Unicode-Zeichen maximal vier Byte lang sein kann. Im Fall der Angabe „varchar2(40 byte)“ ist es in einer Unicode-Datenbank daher möglich, dass lediglich zehn Buchstaben in diese Spalte passen, zum Beispiel, wenn es sich um chinesische Zeichen handelt.

Wenn Sie bei der Anlage der Tabelle nicht definieren, ob man „40 Byte“ oder „40 Char“ speichern möchte, wird die Entscheidung von einem Initialisierungs-Parameter mit dem Namen „NLS_LENGTH_SEMANTICS“ abhängig gemacht. Der wiederum steht standardmäßig auf dem Wert „BYTE“, wenn er nicht durch den Administrator geändert wurde. Da man nicht gern von Standard-Werten abhängig ist, hat es sich daher als Best Practice etabliert, die Angabe, ob „Byte“- oder „Char“-Semantik zur Speicherung verwendet werden soll, bei der Tabellen-Deklaration explizit anzugeben. Dann ist beides auch gemischt möglich, in jedem Fall ist es jedoch eindeutig geregelt.

Woher der Fehler kommt

Nun keimt der Verdacht für den Fehler: Wir haben einen Datenbank-Link, der auf eine fremde Datenbank zeigt. Lokal ist unsere Datenbank in UTF-8 kodiert, aber ist das auch so für die fremde Datenbank? Nein, die fremde Datenbank ist in WIN-1252 kodiert, was kompatibel zu ISO-8859-1 ist und also eine Ein-Byte-Zeichensatzkodierung darstellt. Nun wird das Problem klar:

- In der entfernten Datenbank war als Spalten-Typ „varchar2(40 byte)“ eingetragen
- Nach langer Zeit war zum ersten Mal ein langer Kommentar mit 40 Zeichen eingetragen
- In diesem Kommentar war auch noch ein Umlaut enthalten (der zwei Byte Speicherplatz benötigt)

Diese Zeile der Tabelle benötigt in der entfernten Datenbank 40 Byte Speicherplatz, in der lokalen jedoch 41 Byte. Da andererseits der Cursor die Spaltenbreite aus dem Data Dictionary der entfernten Datenbank entnommen hat, hat das lokale Kommentar-Attribut des Records die Breite 40 Byte, und dort passt die übernommene Zeichenkette nicht hinein: „Exception -06052“.

Lösungsansätze

Wie gesagt: Wer weiß denn so etwas? Bevor wir nun sozusagen zur Moral der Geschichte kommen, hier zunächst einige Lösungsansätze. Auf der entfernten Datenbank in 1-Byte-Kodierung existiert eine einfache Tabelle, um das Problem zu demonstrieren *(siehe Listing 3)*. Die Prozedur in der Multibyte-kodierten, lokalen Datenbank aus *Listing 1* wirft nun für Parameter „1“ keinen, für Parameter „2“ jedoch den Fehler „-06052“ *(siehe Listing 4)*.

Um dieses Problem zu lösen, gibt es mehrere Möglichkeiten. Zum einen wäre es möglich gewesen, die Prozedur anzuweisen, die Daten in die neue Zeichensatz-Kodierung zu konvertieren. Erfolgt dies innerhalb der „select“-Anweisung des Cursors, wird damit automatisch

```
create or replace procedure encoding_test(
  p_id in number)
as
 cursor encoding_cur(p_id in number) is
   select id, convert(text, 'AL32UTF8')
     from encoding_test@kis
    where id = p_id;
begin
  for r in encoding_cur(p_id) loop
    null;
  end loop;
end;
/
```

Listing 5: Geänderte Prozedur

```
SQL> -- Erfolgreicher Test fuer Zeile 2
SQL> call encoding_test(2);
encoding_test 2) erfolgreich.
```

Listing 6

auch die Breite des Record-Attributs angepasst. Dies ermöglicht die Funktion „CONVERT“, aufgerufen wie im Beispiel in *Listing 5*.

Als Name der Zeichensatz-Kodierung, in die übersetzt werden soll, muss der interne Oracle-Bezeichner für die Zeichensatz-Kodierungen gewählt werden, hier „AL32UTF8“. Der Fehler wird nun nicht mehr ausgelöst, die Prozedur funktioniert *(siehe Listing 6)*. Natürlich wäre es auch möglich, einen ganz einfachen, anderen Weg zu gehen: Wenn man die Spalte der entfernten Tabelle als „varchar2(40 char)“ definiert, ist der Umweg über die „CONVERT“-Funktion nicht mehr erforderlich, der Code funktioniert einfach. Natürlich wurde wieder die Testprozedur aus *Listing 1* zum Testen verwendet *(siehe Listing 7)*.

Der Grund ist, dass nun die lokale Datenbank aus der Angabe, zehn Zeichen speichern zu müssen, ableiten kann, dass sie dafür bis zu 40 Byte benötigen wird. Durch die (fehlerhafte) explizite Festlegung auf zehn Byte im Quellsystem hatte es sozusagen die Zusicherung gegeben, dass nicht mehr als diese Datenmenge in Byte übermittelt werden. Eine Zusicherung, die die entfernte Datenbank aus ihrer Sicht ja auch einhält. Durch die Einhaltung der Best Practice kann die lokale Datenbank das Problem nun auflösen.

Fazit

Es gibt viele Beispiele für eine eher entmutigende Tatsache: Die möglichen Fehlerursachen sind vielfältig und zum Teil nur schwer zu testen. Natürlich hätten entsprechend sorgfältig erstellte Testdaten dieses Problem aufdecken können, aber eigentlich nur, wenn man den Fehler bereits kennt. Wer würde sonst daran denken, in Testdaten die maximale Breite einer Spalte nicht nur auszunutzen, sondern auch noch Umlaute zu integrieren? Natürlich ist es auch immer ein guter Rat, sich weiter und weiter mit der Datenbank auseinanderzusetzen, um besser darin zu werden, die Arbeitsweise sowie mögliche Fehlerursachen frühzeitig zu erkennen. Doch eigentlich sind dies alles wohlfeile Ermahnungen.

Wirklich hilfreich sind allerdings Best Practices. Denn wieder einmal zeigt sich, dass hier ein Problem auftaucht, das nur durch einen Verstoß gegen Best Practices möglich wurde: Man verheimlicht der Datenbank, welche Daten man zu speichern gedenkt. Wer zehn Zeichen speichern können möchte, muss dies bei der Anlage der Tabelle auch hinterlegen.

Natürlich ist dies nur ein Beispiel für den Nutzen von Best Practices, aber es zeigt, dass aus diesen Vorgaben Vorteile gezogen werden können, die nicht offensichtlich sind, den Code jedoch robuster machen: Das Beispiel verhindert Fehler, von deren Existenz man möglicherweise nicht einmal wusste. Man kann es auch so ausdrücken: Best Practices sind nicht zuletzt kristallisierte Erfahrung …

```
SQL> -- Zweite Moeglichkeit: Implementierung einer Best Practice
SQL> drop table encoding_test;

table ENCODING_TEST gelöscht.

SQL> create table encoding_test(
   2   id number,
   3   text varchar2(10 char),
   4   constraint pk_encoding_test primary key(id)
   5) organization index;

table ENCODING_TEST erstellt.

SQL> insert into encoding_test(id, text)
   2 select 1, 'Zehn Ohne:' from dual union all
   3 select 2, 'Zehn mit Ö' from dual;
2 Zeilen eingefügt.

SQL> commit;
festgeschrieben.

SQL> -- Erfolgreicher Test fuer Zeile 2
SQL> call encoding_test(2);
encoding_test 2) erfolgreich.
```

Listing 7

Jürgen Sieben
j.sieben@condes.de

Tipps und Tricks aus Gerds Fundgrube
UTF-8 in CSV-Dateien und das Problem mit Excel

Gerd Volberg, OPITZ CONSULTING Deutschland GmbH

Excel-Daten in einer Forms-Applikation zu erzeugen ist kein Hexenwerk. Dazu selektiert man in PL/SQL die benötigten Daten und speichert sie in einem CLOB als CSV ab. Dieses versendet man dann zum Beispiel als Dateianhang via E-Mail an den Anwender. Die Probleme fangen dann an, wenn Excel ins Spiel kommt.

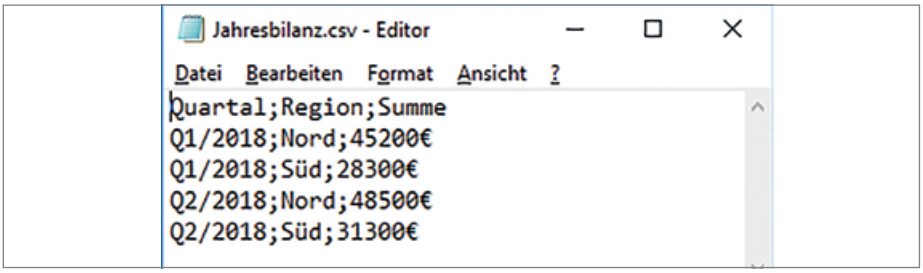


Abbildung 1: Ansicht der Rohdaten im Windows Editor

	A	B	C	D	E
1	Quartal	Region	Summe		
2	Q1/2018	Nord	45200â,¬		
3	Q1/2018	SÃ¼d	28300â,¬		
4	Q2/2018	Nord	48500â,¬		
5	Q2/2018	SÃ¼d	31300â,¬		
6					

Abbildung 2: Darstellung von UTF-8-Zeichen in Excel

Bytefolgen des BOM in verschiedenen Zeichenkodierungen

Kodierung	hexadezimale Darstellung	dezimale Darstellung
UTF-8	EF BB BF [4]	239 187 191

Abbildung 3: BOM-Artikel bei Wikipedia (wikipedia.org/wiki/Byte_Order_Mark)

	A	B	C	D	E
1	Quartal	Region	Summe		
2	Q1/2018	Nord	45.200 €		
3	Q1/2018	Süd	28.300 €		
4	Q2/2018	Nord	48.500 €		
5	Q2/2018	Süd	31.300 €		
6					

Abbildung 4: CSV-Datei mit BOM in Excel geöffnet

Jahresbilanz_mit_BOM.csv

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   EF BB BF 51 75 61 72 74 61 6C 3B 52 65 67 69 6F  ï»¿Quartal;Regio
00000010   6E 3B 53 75 6D 6D 65 0D 0A 51 31 2F 32 30 31 38  n;Summe..Q1/2018
```

Abbildung 5: CSV-Datei mit BOM im Hex-Editor

```
DECLARE
  V_UTF8_BOM   VARCHAR2 (10) := CHR (15711167);
  V_CLOB       CLOB;
BEGIN
...
  V_CLOB := V_UTF8_BOM || V_CLOB;
...
END;
```

Listing 1

Gehen wir einmal davon aus, dass die Datenbank einen UTF-8-Zeichensatz verwendet, die Forms-Applikation in diesem Zeichensatz kompiliert wurde und die CSV-Datei somit zu 100 Prozent aus UTF-8 besteht. Dies alles garantiert noch keine korrekte Darstellung der CSV-Daten in Excel.

Nehmen wir als Beispiel ein paar Bilanzdaten, die in unserer CSV-Datei gespeichert wurden. Der Windows-Editor hat sofort gemerkt, dass die Daten aus UTF-8 bestehen, und zeigt sie korrekt an *(siehe Abbildung 1)*. Wenn man das CSV per Doppelklick in Excel öffnet, sieht das hingegen weniger gut aus *(siehe Abbildung 2)*.

Deutsche Sonderzeichen werden nicht erkannt, das Euro-Symbol ist verschwunden, Eurowerte werden nicht als Zahlen erkannt und deswegen linksbündig dargestellt. So kann man nicht arbeiten.

Die Lösung

Byte Order Mark, auch „BOM" genannt, dient laut Wikipedia als Kennung zur Definition der Kodierungsfunktion in Unicode *(siehe Abbildung 3)*.

Lässt man eine Datei mit den Hex-Werten „EF BB BF" beginnen, werden die meisten Tools, wie auch Excel, erkennen, dass der weitere Text aus UTF-8 besteht. Das Ergebnis ist perfekt *(siehe Abbildung 4)*.

Die Änderungen am Source Code sind minimal. An der Stelle, an der man die fertige CSV-Datei versendet, muss vor dem Versand einfach nur der CLOB um das BOM erweitert werden *(siehe Listing 1)*. Die CSV-Datei sieht dann in einem Hex-Editor wie in *Abbildung 5* aus. Man erkennt sofort in den ersten drei Byte die Zeichenkette „EF BB BF":

Gerd Volberg
gerd.volberg@opitz-consulting.com
talk2gerd.blogspot.com

Optimale Vorbereitung auf Oracle-Zertifizierungen

Rainer Schaub, Acceleris AG

IT-Beratungsunternehmen legen bei der Einstellung von Informatikern schon seit längerer Zeit großen Wert auf eine Oracle-Cloud-Zertifizierung der Kandidaten. Neuerdings findet man jedoch auch bei Stellenausschreibungen von Firmen, die interne IT-Experten einstellen möchten, immer häufiger die Anforderung, dass Kandidaten Oracle-zertifiziert, also beispielsweise Oracle Infrastructure as a Service Cloud 2017 Certified Implementation Specialist, Oracle Database Cloud Administrator Certified Associate oder MySQL Cloud Service 2018 Certified Implementation Specialist sind. Daraus lässt sich folgern, dass eine – möglichst erst kürzlich erfolgte – Oracle-Zertifizierung einen Mehrwert sowohl für den Mitarbeiter/Bewerber als auch für den Arbeitgeber darstellt.

Um eine Prüfung bestehen zu können, benötigt man sowohl Kenntnisse über das Prüfungsthema (etwa Oracle Database Cloud Administration) als auch Kenntnisse über die Form der Prüfung (wie Multiple Choice oder Sprachform). Der vorliegende Artikel behandelt ausschließlich die Aspekte der Form einer Oracle-Zertifizierungsprüfung; die Hinweise sind lediglich für Multiple-Choice-Aufgaben anwendbar und damit nicht für eine Oracle-Certified-Master-Prüfung (OCM). Informationen über eine generelle Vorgehensweise zur Vorbereitung auf den Inhalt einer Oracle-Prüfung findet man beispielsweise im Video von Gwen Lazenby zu „How To Prepare For Your Oracle Certification Exam" *(siehe „https://www.youtube.com/watch?v=RcxXjN-QsbY")*.

Dieser Artikel gibt Hinweise zu den Unterschieden zwischen Oracle-Cloud- und Oracle-Nicht-Cloud-Zertifizierungen sowie zu Fragetypen und magischen Wörtern. Mit „Fragetyp" ist eine besondere Art der Fragestellung gemeint, die sich zum einen deutlich von einem anderen Typ unterscheidet und zudem einer eigenen Antwortstrategie bedarf. „Magische Wörter" sind Wörter, die spezielle Beachtung verdienen, da sie Hinweise darauf geben, ob diese Antwort eher mehr oder eher weniger infrage kommt.

Bis zum 21. Juli 2018 gibt es 75 Oracle-Cloud-Zertifizierungen (OCZ) und die Anzahl steigt kontinuierlich an. Die aktuelle Liste ist unter *„https://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=632"* einsehbar.

Zertifizierungs-Typen, notwendige Vorbereitungsdauer und Wert

Die Oracle-Zertifizierungen lassen sich nach den in ihnen vorkommenden Begriffen „Certified Implementation Specialist", „Essentials", „Oracle Certified Associate" (OCA), „Oracle Certified Professional" (OCP) und „Oracle Certified Expert" (OCE) unterscheiden und einteilen. Diese Bezeichnungen können sowohl Hinweise auf die notwendige Vorbereitungsdauer für das Bestehen der Prüfung als auch auf den Stellenwert der Zertifizierung geben. Je länger die Vorbereitungsdauer, desto größer auch der Stellenwert des Zertifikats.

Im Bereich „OCZ" gibt es derzeit überwiegend „Certified Implementati-

on Specialist"-Zertifizierungen (58 von 75). Dies liegt hauptsächlich daran, dass die Cloud-Domäne noch recht jung ist. Nur einige wenige „Oracle Certified Associate"-Prüfungen stehen zur Verfügung. Es ist allerdings zu vermuten, dass in naher Zukunft auch Oracle-Certified-Professional-Zertifizierungsprüfungen (OCP) hinzukommen werden. *Tabelle 1* gibt einen groben Überblick über die relative Vorbereitungsdauer, die für die unterschiedlichen Zertifizierungstypen notwendig ist.

Prüfungsdauer, Anzahl der Fragen und Gültigkeitsdauer der Zertifizierung

Die Dauer der OCZ-Prüfungen ist in der Regel kürzer als die der Nicht-OCZ-Prüfungen; eine Ausnahme bildet die Zertifizierung „Oracle Cloud Platform Data Management 2018 Associate", die 120 Minuten dauert. Sie beinhalten deshalb konsequenterweise auch weniger Fragen. Eine übliche Prüfungsdauer für OCZ liegt zwischen 60 und 105 Minuten, wohingegen die Prüfungsdauer von Nicht-OCZ zwischen 90 und 150 Minuten liegt. Die Anzahl der Fragen ist direkt abhängig von der Dauer der Prüfung. Im Durchschnitt stehen für die Beantwortung einer Frage etwa 90 Sekunden zur Verfügung. Deshalb muss der Kandidat in der Lage sein, die Fragen sehr schnell zu beantworten; infolgedessen kann es fatal sein, für einzelne Antworten zu viel Zeit zu brauchen.

Zertifizierungstyp	Relative Vorbereitungsdauer (Tage)
Certified Implementation Specialist	1
Essentials	1
Oracle Certified Associate (OCA)	1,5 - 2,5
Oracle Certified Professional (OCP)	2 - 3,5
Oracle Certified Expert (OCE)	3 - 5

Tabelle 1: Zertifizierungstyp und relative Vorbereitungsdauer

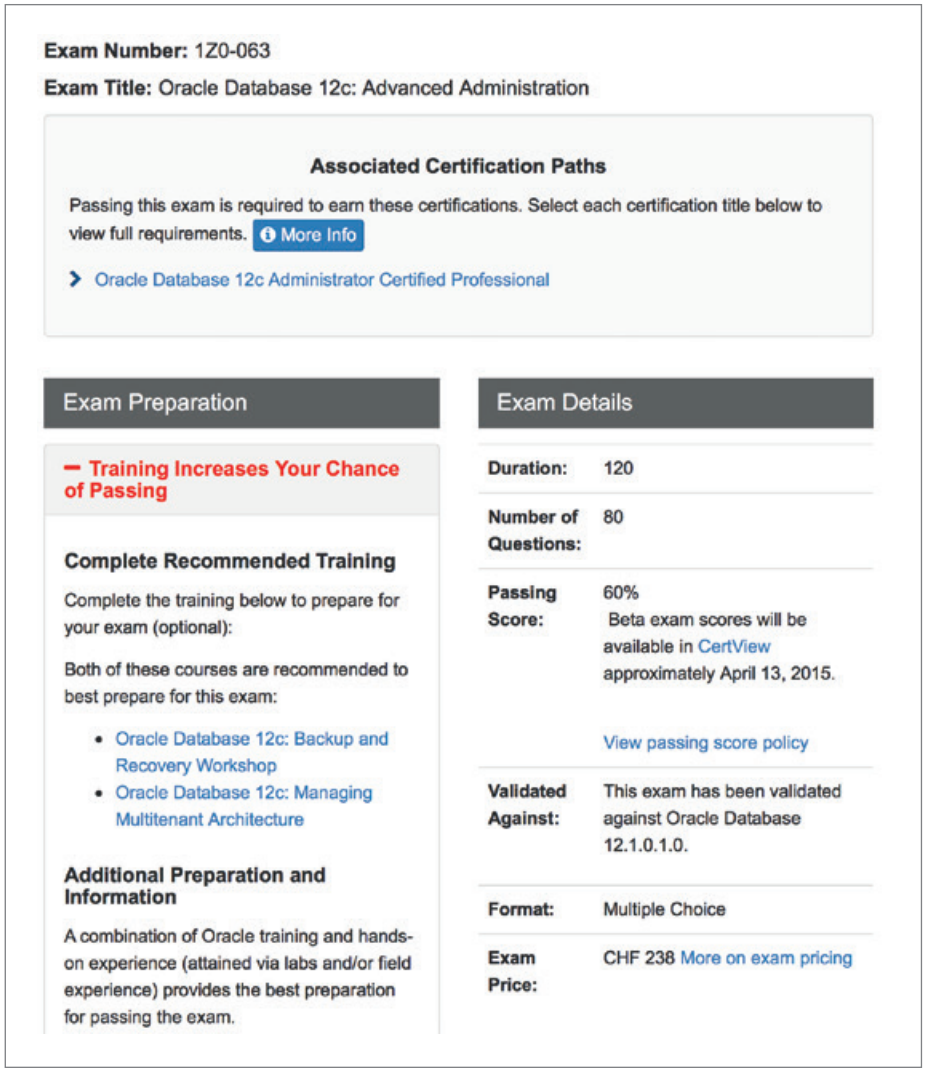


Abbildung 1: Prüfungsvorbereitung inklusive „Validated Against"

Eine Ausnahme im Hinblick auf die pro Frage zur Verfügung stehende Zeit stellt die Zertifizierung „Oracle Cloud Platform Content and Experience 2018 Associate" dar, die lediglich 40 Fragen beinhaltet. Es ist daher anzunehmen, dass die Fragen sehr anspruchsvoll sind, da die Prüfungsdauer dennoch 90 Minuten beträgt. Diese Prüfung ist die einzige dem Autor dieses Artikels bekannte Zertifizierungsprüfung, in der im Durchschnitt mehr als zwei Minuten pro Antwort zur Verfügung stehen.

Die Cloud ist schnelllebiger als die bisherigen Informatiksysteme. Dies macht sich auch in der Gültigkeitsdauer der Zertifizierungen bemerkbar. Oracle spricht beim Ablauf einer Zertifizierung von „retiring" und stellt unter dem Link *„http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=206"* die Liste der Zertifizierungen vor, deren Gültigkeit in naher Zukunft ablaufen wird. Während eine Gültigkeitsdauer von fünf Jahren für Nicht-OCZ üblich ist (Oracle Database 11g Adminstrator Certified Associate ist sogar schon zehn Jahre gültig), beträgt sie für Cloud-Zertifizierungen nur etwa zwei Jahre.

Dokumentationen und deren Dynamik sowie praktische Übungsmöglichkeiten

Auch die Dokumentationen der OCZ sind viel kurzlebiger als die der Nicht-OCZ. Dies erschwert die Prüfungsvorbereitung. *Abbildung 1* zeigt die relevanten Informationen für die Zertifizierung „Oracle Database 12c: Advanced Administration". Neben der Prüfungsdauer, der Anzahl der Fragen und dem „Passing Score" ist der Bereich „Validated Against" eine wichtige Information: Sie gibt die exakte Software-Version an, die der Kandidat zur Prüfungsvorbereitung benutzen sollte.

Bei OCZ-Prüfungen fehlt diese wichtige Information „Validated Against" bis-

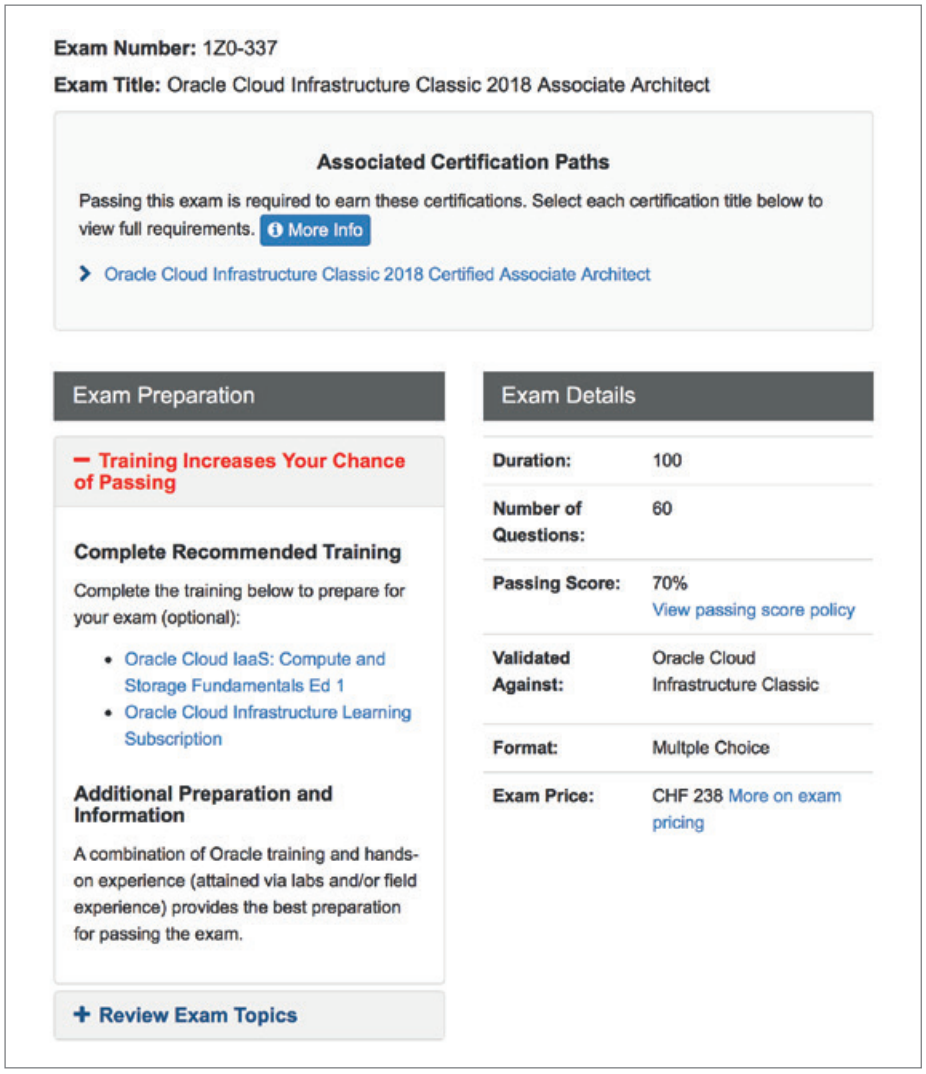


Abbildung 2: Prüfungsvorbereitung inklusive „Review Exam Topics“

weilen. Wird sie dennoch aufgeführt, ist es nicht ganz einfach, die relevante Dokumentation zu finden, da im Bereich „Oracle Cloud“ üblicherweise nur die Dokumentation zur aktuellsten Software-Version verfügbar ist.

Im Verlauf der letzten fünfzehn Jahre sind Oracle-Zertifizierungen immer anspruchsvoller geworden. Neben einer soliden Kenntnis der Theorie ist eine genügend lange Arbeit am Rechner (mindestens ein Jahr praktische Erfahrung) mit der entsprechenden Software unabdingbar geworden; der Anteil der praktischen Vorbereitung muss deutlich länger als noch vor zehn Jahren gewählt werden. Dies stellt bei der Vorbereitung auf eine OCZ für den Kandidaten eine zusätzliche Hürde dar, da der Zugriff auf die Oracle-Cloud mit Kosten verbunden ist. Mit der „Free Oracle Cloud Promotion“ lässt sich diese Schwierigkeit jedoch umgehen. Informationen hierzu unter *„https://cloud.oracle.com/trial-faq“*, *„https://cloud.oracle.com/en_US/tryit“* oder *„https://docs.oracle.com/en/cloud/paas/database-dbaas-cloud/index.html“*.

Inhaltliche Themen für ausgewählte Oracle-Cloud-Zertifizierungen

Es ist von Vorteil, immer die „Exam Topics“ (siehe *Abbildung 2* plus Review Exam Topics) der jeweiligen Zertifizierung zur Vorbereitung heranzuziehen. Daneben gibt es gewisse Themen, die übergreifend über mehrere OCZ Bedeutung haben, und andere, die zum Bestehen einer einzelnen OCZ essenziell sind. Folgende Themen sind übergreifend für mehrere OCZ-Prüfungen relevant:

- Netzwerk-Wissen und Netzwerk-Protokolle „ssh“, „ssh-key“, „http“, „https“, „proxy“, „gateway“, „authentication token“, „tunnel“ sowie „standard/default ports“ für Protokolle
- Connectivity: „REST“, „REST API“, „RESTful Web Service“, „stateful“, „stateless“, „Java library“
- Administration: Kenntnisse des spezifischen Command-Line-Interface (cli) der jeweiligen Applikation. Oftmals reichen hier die Kenntnis des Namens des „cli“ sowie die der groben Architektur und Funktionalität aus
- Kenntnis und Verständnis der unterschiedlichen Services innerhalb der jeweiligen Zertifizierungsthemen und Abläufe für diverse „cloud subscriptions“/ „services“ und deren Installation von der Bestellung bis zur Nutzung (My Service Dashboard, My Service Page, My Service Application, JavaScript, XML, WebLogic Server, Java Virtual Machine (JVM), JBoss)
- Kenntnis der unterschiedlichen Typen von Administratoren (wie Service Administrator, Identity Domain Administrator)
- Kenntnis der Oracle-Cloud-Benutzer (wie „oracle“, „opc“, „oracle“, „grid“, „root“) und der neuen Nutzergruppen

Markieren und mehrere Durchläufe

Im Gegensatz zu einer GMAT-Prüfung (Graduate Management Admission Test, *siehe „http://www.mba.com/us“*) kann bei einer Oracle-Zertifizierungsprüfung eine Frage markiert und später (nochmals) beantwortet werden. Dies ist sehr hilfreich, da es vorkommen kann, dass die Lösung oder eine Teillösung einer Frage aus dem Inhalt einer anderen Frage abgeleitet werden kann. Zudem ist es unter dem Gesichtspunkt der Prüfungsstrategie logisch, schwierige Fragen erst zu bearbeiten, wenn bereits ein gewisser Fundus von Antworten vorliegt, da so Zeit und Energie gespart werden können.

Last but not least ist es psychologisch geschickter, in einem ersten Durchgang die einfacheren Fragen zu bearbeiten. Die Ideen dieses Kapitels stammen zum Großteil aus John Watsons Buch „OCA Oracle Database 12c Installation and Administration Exam Guide“, 2014, McGraw-Hill Education, und wurden vom Autor erfolgreich angewandt.

Fragetypen

Die Kenntnis unterschiedlicher Fragetypen und der dazugehörigen optimalen Antwortstrategie hilft, kostbare Prüfungszeit zu sparen, die Ressourcen (Zeit, Energie, Konzentrationsfähigkeit) besser einzuteilen und so die Chancen zu erhöhen, eine höhere Antwortquote zu erreichen. Dem Autor sind derzeit vier unterschiedliche Fragetypen bekannt, von denen jedoch derzeit nur zwei in OCZ vorkommen. Es werden nur diese beiden Fragetypen erläutert und eine Strategie zur adäquaten Behandlung vorgeschlagen. Alle vier Fragetypen sowie ein ausführlicher Strategievorschlag ist im Artikel „Optimale Vorbereitung auf Oracle-Zertifizierungen" des Autors im Red Stack Magazin, Ausgabe 6 vom Dezember 2016, erschienen.

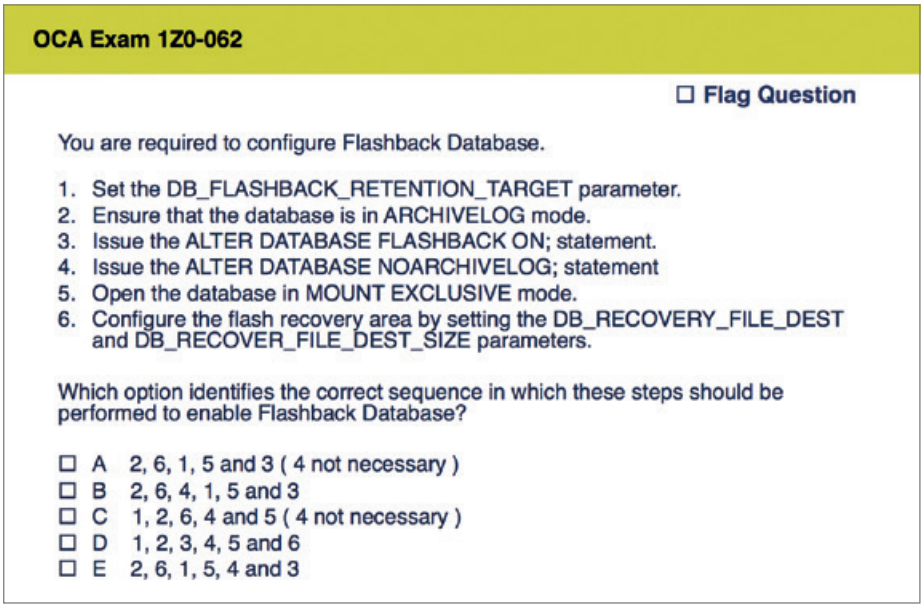


Abbildung 3: Beispiel für eine „Complicato"-Frage

Complicato

Dieser Fragetyp zeichnet sich durch eine komplizierte Struktur aus, bietet jedoch den Vorteil, dass nur eine Antwort richtig ist. Zuerst werden der Sachverhalt sowie eine Liste einzelner Schritte beschrieben, die zur Lösung des Sachverhalts notwendig sein könnten. Anschließend muss der Kandidat die richtige Antwort aus einer Reihe von Möglichkeiten auswählen.

Der Versuch, selbst die richtige Reihenfolge der einzelnen Schritte zu finden, erfordert kostbare Zeit und ist meistens zum Scheitern verurteilt. Eine gute Strategie besteht darin, die Beschreibung des Sachverhalts genau zu lesen und die Liste der einzelnen Schritte darauf zu prüfen, ob sie zur Lösung nicht notwendige Schritte enthält oder einzelne Schritte mehrfach zur Lösung notwendig sind. Mit diesem Wissen kann in der Regel schon die eine oder andere Antwort ausgeschlossen werden; gelegentlich findet man die richtige Antwort, weil ein Schritt mehrfach vorkommen muss und dies nur bei einer Antwort der Fall ist.

Weitere Möglichkeiten zur Lösungsfindung bestehen darin, den jeweils ersten oder letzten Schritt über alle Antwortmöglichkeiten zu vergleichen und so die Anzahl der möglichen Lösungen weiter einzugrenzen. Manchmal sind auch unnötige Schritte („not neccessary") aufgeführt. Dies stellt eine weitere gute Möglichkeit dar, die Lösung zu finden. Sind beispielsweise zur Lösung der Aufgabe notwendige Schritte als „not neccessary" ausgewiesen, so ist klar, dass diese Antwort falsch ist. Hier gilt es, diese Frage sofort zu markieren, zur nächsten überzugehen und die Frage erst in einem späteren Durchlauf zu bearbeiten, da in der Regel die Beantwortung dieses Typs von Fragen zeitaufwendig ist *(siehe Abbildung 3)*.

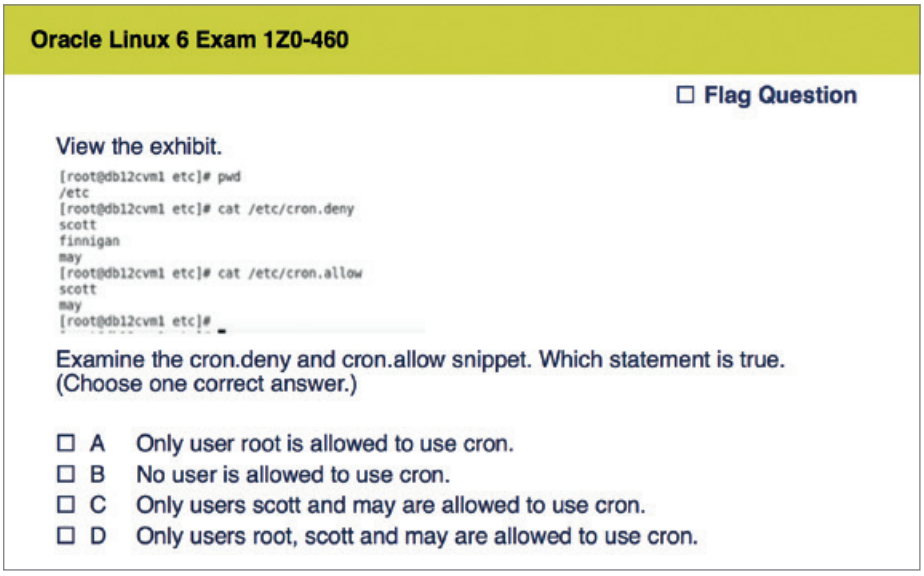


Abbildung 4: Beispiel für eine „Choose 1, 2 or 3"-Frage

Choose 1, 2 or 3

Auch dieser Fragetyp weist wie „Complicato" die richtige Anzahl der Antworten aus. Dies stellt den einfachsten Typ dar. Hier gilt es, mithilfe des „Process of Elimination"-Verfahrens (POE) falsche Antworten zu eliminieren und dann, falls die Antwort offensichtlich oder bekannt ist, diese auszuwählen *(siehe Abbildung 4)*.

Gesamtstrategie zu den Fragetypen

Es empfiehlt sich, für den Fragetyp „Complicato" (COMP) im ersten Durchgang keine Zeit aufzuwenden und nur zu markieren. Im zweiten Durchgang kann man dann „Complicato"-Fragen lösen. Fragen des Typs „Choose 1, 2 oder 3" löst man

im ersten Durchgang, falls sie einfach sind, andernfalls im zweiten Durchgang. Auch wenn jeder Kandidat letztlich seine eigene Strategie finden muss, stellt *Abbildung 5* eine für den Autor optimale Gesamtstrategie vor.

Abbildung 5: Strategie-Vorschlag zu Fragetypen

Magische Wörter

„Magische Wörter" sind Ausdrücke, die spezielle Beachtung verdienen, da sie Hinweise darauf geben, ob eine Antwort eher mehr oder eher weniger infrage kommt. Nachfolgend eine Liste magischer Wörter, die in Oracle-Zertifizierungsprüfungen vorkommen:

- ONLY
- MUST
- ALWAYS
- CAN
- CAN ONLY
- EXACTLY
- NEVER
- AUTOMATICALLY
- MIGHT

Wörter wie „must", „always", „exactly" und „never" sind „einengend", da sie auf einer Skala am einen (etwa „never") oder am anderen Ende (etwa „always") des möglichen Lösungs-Spektrums vorkommen und somit quasi Extremsituationen darstellen. Wörter wie „might" oder „can" sind im Gegensatz dazu „weitend", sie kommen auf einer Skala der möglichen Lösungen im mittleren Bereich vor und repräsentieren somit die große Masse. Das Wort „automatically" kann ebenfalls „einengend" sein, da auch hier kein Handlungsspielraum vorhanden ist.

Einengende Wörter wie „always" oder „never" geben einen Hinweis darauf, dass die Antwort, die dieses magische Wort enthält, tendenziell nicht richtig ist. Anwendungsbeispiel: Man hat bei einer „choose 2"-Frage bei fünf möglichen Antworten zwei Antworten als falsch und eine weitere als richtig identifiziert. Bei den übrigen zwei Antwortmöglichkeiten hat man jedoch keine Idee, welches die zweite richtige Antwort ist. Wenn nun in einer der beiden Antworten ein einengendes Wort wie „never" oder „always" vorkommt, so sollte man die andere Antwort als zweite richtige auswählen. Kommt in einer der beiden Antworten ein „might" vor, während die andere kein magisches Wort enthält, so wäre die „might"-Antwort zu bevorzugen.

Gegenseitiger Ausschluss

Manchmal kommt es vor, dass sich zwei Antworten völlig widersprechen. Dies stellt einen gegenseitigen Ausschluss dar. Ist die eine Antwort richtig, kann die andere es nicht sein. Beim gegenseitigen Ausschluss ist zudem wichtig zu wissen, dass meistens eine Antwort davon eine richtige Antwort ist. Anwendungsbeispiel: Man hat bei einer „choose 2"-Frage bei fünf möglichen Antworten eine als falsch und eine weitere als richtig identifiziert. Bei den letzten drei Antwortmöglichkeiten hat man jedoch keine Idee, welche die zweite richtige Antwort ist. Falls in keiner der drei zur Wahl stehenden Antworten ein magisches Wort vorkommt, jedoch zwei Antworten einen gegenseitigen Ausschluss darstellen, so sollte man eine der beiden sich ausschließenden Antworten wählen.

Doppelte Verneinung

Die doppelte Verneinung ist immer und unabhängig von der vorliegenden Prüfungssituation schwierig und verwirrend. Hier ist es ratsam, die Frage als Bejahung umzuformulieren und anschließend die Lösung der Frage in Angriff zu nehmen. Die doppelte Verneinung ist zwar nach Erfahrung des Autors selten in Oracle-Zertifizierungsprüfungen, aber kommt vor.

Fazit

Voraussetzung für das Bestehen einer Prüfung ist immer die Sachkenntnis. Bei einer Oracle-Zertifizierungsprüfung gibt es keine Noten, sondern lediglich „bestanden" oder „nicht bestanden". Es gilt also, die erforderliche Quote der richtigen Antworten zu erzielen. Die gute Kenntnis der Form einer Prüfung allein reicht zwar nicht aus, um diese zu bestehen, sie kann aber den Ausschlag zwischen „nicht bestanden" und „bestanden" geben. Deshalb ist die Kenntnis möglichst vieler Rahmen-Faktoren in vielen Fällen nützlich und hilfreich. Wenn diese Ausführungen dazu beitragen, so haben sie ihren Zweck erfüllt. Ein besonderer Dank des Autors gilt Herrn Dr. Hans-Peter Schlunke für das ausführliche Korrekturlesen dieses Artikels.

Rainer Schaub
rainer.schaub@acceleris.ch

Wir begrüßen unsere neuen Mitglieder

Persönliche Mitglieder

- Marcus Schuster
- Stefan Raabe
- Michaela Oswald
- Niklas Peters
- Norbert Kalbe
- Peter Schmidhofer
- Christoph Kraliczek
- Jan Richter
- Klaus Bumann-Schneider
- Athanasios Manolopoulos
- Uwe Richter
- Andreas Schlögl

Firmenmitglieder DOAG

- Norddeutscher Rundfunk für die ARD, Arne Sievers
- secion GmbH, René Sommer
- Landschaftsverband Westfalen-Lippe, Norbert Holtkamp
- Deutsche Telekom Clinical Solutions GmbH, Steven Noble
- Traveltainment GmbH, Björn Schild
- Telekom IT GmbH, Klaus-Dieter Vortmann
- Pixelpark GmbH, Klaus Kuntz

Termine

Januar

15.01.2019
Regionaltreffen Hannover
Andreas Ellerhoff

16.01.2019
DOAG Noon2Noon 2019
Frankfurt am Main

17.01.2019
Regionaltreffen NRW
Köln

22.01.2019
Regionaltreffen Bremen
Ralf Kölling

Februar

04.02.2019
Regionaltreffen München/Südbayern
Andreas Ströbel

21.02.2019
DOAG Forms Day 2019
München

März

11.03.2019
Regionaltreffen München/Südbayern
Andreas Ströbel

18.03.2019
NextGen-Programm Javaland 2019
Brühl

19.03.2019
Javaland 2019
Brühl

26.03.2019
Data Analytics 2019
Brühl

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:
Caroline Sengpiel,
DOAG Dienstleistungen GmbH
(Anschrift s.o.)

Fotonachweis:
Titel: © ar130405/Fotolia
S. 11: © MySQL
S. 17: © 1tjf/123RF
S. 22: © Pannawat Muangmoon/123RF
S. 26: © Artisticco LLC/123RF
S. 32: © Bakhtiar Zein/123RF
S. 36: © PostgreSQL
S. 40: © Shao-Chun Wang/123RF
S. 45: © Bakhtiar Zein/123RF
S. 49: © faithie/123RF
S. 53: © Oracle & © Vmware
S. 58: © mariok/123RF
S. 65: © enterline/123RF
S. 70: © natalimis/123RF
S. 73: © rangizzz/123RF
S. 74: © alphaspirit/123RF
S. 78: © Luca Bertolli/123RF
S. 80: © Bjoern Wylezich/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

JavaLand

Early Bird
bis 15. Jan. 2019

19. - 21. März 2019 in Brühl bei Köln

Ab sofort Ticket & Hotel buchen!

www.javaland.eu

Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu

Bei den mittlerweile unüberschaubaren Cloud-Angeboten sind wir Ihr verlässlicher und vorausschauender Navigator für Ihren Weg in die Cloud. Für Sie haben wir sämtliche Aspekte im Blick und entscheiden gemeinsam mit Ihnen die richtige Strategie. Wir sind an Ihrer Seite: von der Beratung über die Planung und Umsetzung bis hin zu Training und Betrieb. Sprechen Sie mit uns.
www.trivadis.com/cloud-solutions | info@trivadis.com

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

trivadis
makes IT easier.