Nr. 1 | Februar 2026 | ISSN 2366-7915 | www.doag.org

Red Stack Magazin

DOAG SOUG swiss oracle user group AOUG AUSTRIAN ORACLE USER GROUP

DATENBANKEN

Aus der Praxis

GoldenGate Microservices – Von 0 auf Replikation in 45 Minuten

Im Interview

Maria Colgan, VP Product Management für KI und geschäftskritische Daten, Oracle

Datenbanken

Neue Transaktionsansätze in Oracle AI Database 26ai

WISSEN GEHT.

BETRIEB BLEIBT.

Wer betreibt Ihre Datenbank, wenn das Wissen geht?

Managed Services stellen sicher, dass Ihr Datenbankbetrieb auch dann zuverlässig funktioniert, wenn Wissen das Unternehmen verlässt. Der Abgang erfahrener Spezialisten ist Realität, qualifizierter Nachwuchs bleibt knapp und Übergaben ersetzen keine jahrelange Erfahrung. Ob Oracle, Microsoft SQL Server, PostgreSQL, MongoDB oder andere Plattformen: Systeme kennen keinen Übergang und keinen Aufschub. Der Betrieb muss weiterlaufen. OPITZ CONSULTING sorgt mit Managed Services für klare Verantwortlichkeiten, feste SLAs und einen stabilen Betrieb Ihrer Datenbanklandschaften, unabhängig von Personalwechseln oder Engpässen.

JETZT STABILEN DATENBANK-BETRIEB SICHERN

Für Oracle, Microsoft, Postgres, MongoDB & mehr

Nur Kundenzufriedenheit zählt

VIESMANN · BARTELS – LANGNESS · mcs · RENOLIT · ORACLE | Partner · Barmenia Versicherungen · ERGO · SCHERDEL

OPITZ CONSULTING SYSTEMS

Christian Trieb
Vorstand Datenbanken, Leiter Datenbank Community

Liebe Mitglieder, liebe Leserinnen und Leser,

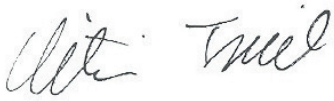
das Warten hat ein Ende: Die Oracle AI Datenbank 26 ai ist auf Linux On-Prem verfügbar. Lange hat es gedauert. Erst war die Oracle-Datenbank 23 ai nur in der Cloud und auf Engineered Systems verfügbar, aber nun ist sie als Oracle AI Datenbank 26 ai Enterprise Edition auch für On-Prem-Linux-Systeme installierbar. Das sie nun endlich verfügbar ist, dazu hat die DOAG entscheidend beigetragen. In zahlreichen Gesprächen mit Oracle-Repräsentanten, zuletzt im Rahmen der DOAG 2025 Konferenz + Ausstellung mit Oracle Executive Vice President Database Engineering Hasan Rizvi, hat die DOAG dies im Sinne der DOAG-Mitglieder thematisiert. Mit dieser Version ist nun die neue Multitenant-Datenbankarchitektur obligatorisch.

In dieser Ausgabe erscheinen viele Artikel, die sich mit unterschiedlichen Aspekten der Oracle-Datenbank befassen. Auch das Interview mit der Oracle-Produkt-Managerin Maria Colgan bietet zahlreiche interessante Einblicke. Einige weitere informative Artikel zur Oracle-Datenbank, auch von Mitgliedern der DOAG Datenbank Community, greifen diese Thematik anschaulich und gut auf. Aber auch andere Datenbanken spielen in dieser Ausgabe eine Rolle.

Für Entwicklerinnen und Entwickler sind auch einige interessante Artikel über datenbanknahe Programmierung in diesem Heft enthalten.

In diesem Sinn wünsche ich euch einen regen Erkenntnisgewinn beim Lesen der Artikel dieser Ausgabe.

Bleibt gesund und ein gutes erfolgreiches 2026.



Christian Trieb

Ausgabe Nr. 1/2026
auf Abruf!

Manuelle Datenmigration von Oracle 11 zu Oracle 19c

SQL-Tuning für PostgreSQL

Einleitung

Development

Cloud

APEX

PL/SQL

Development

KI

News

Intern

TIMELINE

2. Oktober 2025

Im DOAG DevTalk mit Matthias Schulz und Christian Schwitalla lautet das Thema „Modernes SQL: Neuerungen in Oracle 23 AI“

8. Oktober 2025

Das Expertenseminar „Praxisworkshop Oracle Database Appliance” mit Florian Barth findet in Berlin statt.

16. Oktober 2025

Beim DOAG DevTalk laden Martin Bach und Sabine Heimsath zum Erfahrungsaustausch zum Thema „Oracle Datenbank und AI“.

22. Oktober 2025

Im DOAG DBTalk mit Bruno Cirone und Christian Pfundtner wird über das Thema „Datenbank Security Check Tools in der Praxis“ gesprochen.

23. Oktober 2025

Das Regionaltreffen Dresden findet in Dresden statt. Auf dem Programm steht ein Vortrag von Steffen Reißig zum Thema „DBA ade? Der Weg vom Datenbankspezialisten zum Cloud-Architekten“.

7. November 2025

In der DB WebSession „Gegenüberstellung SE2 und EE“ beleuchtet Marco Pachaly-Mischke beide Editionen aus funktionaler Sicht und gibt Entscheidungshilfen, ob und wann sich der Einsatz der Enterprise Edition lohnt.

13. November 2025

Im DOAG DevTalk mit Moritz Klein und Niels de Bruijn geht es um „Prozessmodellierung und KI mit Oracle APEX“.

17. bis 18. November 2025

Die European NetSuite User Days finden zum siebten Mal im Rahmen der DOAG 2025 Konferenz + Ausstellung in Nürnberg statt. Das Event bietet ein vielseitiges Programm rund um NetSuite – mit starkem praxisorientiertem Fokus auf KI-, ERP- Finanz- und Produktionsthemen.

18. bis 20. November 2025

Ein vielfältiges Programm zu den sechs Streams "Datenbank", "Development", "Infrastruktur", "Data Analytics & KI", "Strategie & Softskills" sowie "Data Analytics & KI" erwartet die Besucherinnen und Besucher der DOAG 2025 Konferenz + Ausstellung in Nürnberg. Die führende Anwender-Konferenz mit Fokus auf Oracle-Technologien im deutschsprachigen Raum lockt rund 1.500 Teilnehmerinnen und Teilnehmer ins NCC Ost. Neben jeder Menge Wissen gibt es auch viele Gelegenheiten zum persönlichen Austausch und Networking sowie eine Reihe von Community-Aktivitäten. Auf den Ausstellungsflächen bieten knapp 40 Partner die Gelegenheit mit Firmen ins Gespräch zu kommen. Den Abschluss der Anwenderkonferenz bildet schließlich der Schulungstag am Freitag.

18. bis 19. November 2025

Die Konferenz Low-Code Creator 2025 für Low-Code-Entwicklerinnen und -Entwickler in Zusammenarbeit mit Low-Code Association e.V. findet ebenfalls im Nürnberg Convention Center (NCC Ost) statt.

19. bis 20. November 2025

Rund 400 Teilnehmerinnen und Teilnehmer freuen sich auf das Wiedersehen der KI Community zur Konferenz KI Navigator im Nürnberg Convention Center und teilen ihr Wissen rund um die professionelle Anwendung von KI in Unternehmen und Organisationen sowie deren Auswirkungen auf Politik, Wirtschaft und Gesellschaft.

26. November 2025

Im DOAG DBTalk mit Marco Pachaly-Mischke und Christian Trieb dreht sich alles um die Oracle Datenbank SE2.

2. bis 16. Dezember 2025

Die Mitglieder des DOAG e.V. nominieren die Kandidaten zur Wahl der Delegiertenversammlung für die Amtszeit 2026 – 2029.

3. Dezember 2025

Das Regionaltreffen Osnabrück/Bielefeld/Münster findet in Gütersloh statt. Auf der Agenda stehen zwei Vorträge zu den Themen „Golden Gate in der OCI“ und „Oracle Database Security – Alles unter Kontrolle?“.

11. Dezember 2025

Beim DOAG DevTalk „Moderne Oracle Softwarenentwicklung“ gibt es zwei Referenten: die KI-Mika (XAI/Grok4/expert) und Frank Hoffmann.

12. Dezember 2025

„Effizienter Datenbankbetrieb für MongoDB“ steht im Mittelpunkt der DB WebSession mit Marcin Trębacz und Jeremy Smeets.

9. Januar 2026

Mit „101 Data Governance” beschäftigt sich Matthias Jung in seiner DB WebSession.

28. Januar 2026

Im DOAG DBTalk mit Christian Pfundtner und Johannes Ahrends steht das Thema „Oracle Datenbank Lizenzierung – Q&A“ auf der Agenda.

2. bis 5. Februar 2026

Die Mitglieder des DOAG e.V. wählen die Delegiertenversammlung für die Amtszeit 2026 bis 2029. Die Wahl findet elektronisch statt.

„Obwohl KI es uns ermöglicht, viele Routineaufgaben zu automatisieren, die DBAs und Datenbankentwickler einmal verwaltet haben, glaube ich nicht, dass KI diese Rollen in absehbarer Zeit ersetzen wird."

Martin Meyer, Reaktionsleiter des Red Stack Magazin, unterhielt sich mit Maria Colgan, Vizepräsidentin Product Management für KI und geschäftskritische Daten im Datenbankteam von Oracle, über Datenbanken, die aktuellen und zukünftigen Aufgaben von Datenbank-Administratoren, den Einfluss von Künstlicher Intelligenz, den Oracle Optimizer, die Wahl zwischen On-Premise, Public Cloud und Sovereign Cloud und Cloud-Kostenmanagement.

Bitte stellen Sie sich unseren Lesern vor. Wer sind Sie und was ist Ihre Rolle bei Oracle?

Hallo, ich bin Maria Colgan, Vizepräsidentin Product Management für KI und geschäftskritische Daten im Datenbankteam von Oracle. Ich arbeite seit fast drei Jahrzehnten im Datenbank-Entwicklungsteam und bin das, was manche als Datenbank-Nerd bezeichnen würden. Es macht mir große Freude, Kunden dabei zu helfen, ihren Datenbanken die beste Leistung und den größten Wert zu entlocken. Ich liebe es auch, mein Wissen zu teilen, indem ich auf Konferenzen präsentiere und meinen Blog *sqlmaria.com* pflege. Dafür habe ich heutzutage jedoch weniger Zeit, da ich ein Team von wirklich großartigen Produktmanagern leite, was sehr zeitaufwendig ist.

Welche Rollen und Aufgaben werden Ihrer Meinung nach Datenbanken – insbesondere Oracle-Datenbanken – in Zukunft haben, sowohl kurzfristig als auch langfristig?

In naher Zukunft wird Oracle Database weiterhin das Rückgrat wichtiger Systeme in den Bereichen Finanzen, Personalwesen, Lieferkette, Kundenmanagement und vielen anderen Bereichen sein. Und da sich Unternehmen neuen Herausforderungen stellen und neue Technologien wie KI einsetzen, werden sich Datenbanken mit ihnen weiterentwickeln. Ich gehe davon aus, dass wir im Zuge dieser Transformation noch mehr Automatisierung und eine tiefere Integration mit KI, Cloud-Services und Advanced Analytics sehen werden.

Die Oracle-Datenbank hat sich in der Vergangenheit stark weiterentwickelt und das wird sich auch in den nächsten Jahren fortsetzen. Wie wird sich die Arbeit von DBAs und Entwicklern verändern oder weiterentwickeln? Welche Fähigkeiten werden wichtiger, welche vielleicht weniger?

Während sich Cloud-Technologien – und jetzt auch KI – weiterentwickeln, nimmt Oracle Database einem DBA mehr von der mühsamen Arbeit ab. Aufgaben wie Provisionierung, Backups und Patching, die früher ganze Tage aufgezehrt haben, sind jetzt weitgehend automatisiert. Dadurch wird die DBA-Rolle weitaus strategischer. Anstatt nur „alles am Laufen zu halten", arbeiten DBAs enger mit dem Unternehmen zusammen, konzentrieren sich auf langfristige Planung, proaktive Optimierung und übernehmen Verantwortung auf Architekten- oder Beraterebene. Dennoch bleibt die Fähigkeit zur Überwachung, Diagnose und Feinabstimmung von Anwendungs-Workloads ein wesentlicher Bestandteil der Arbeit und jeder DBA sollte diese Fähigkeiten weiter verbessern.

Wie sehen Sie die Zukunft von Oracle DBAs und Datenbankentwicklern? Könnte die Entwicklung von KI dazu führen, dass diese Berufe überflüssig werden?

Obwohl KI es uns ermöglicht, viele Routineaufgaben zu automatisieren, die DBAs und Datenbankentwickler einmal verwaltet haben, glaube ich nicht, dass KI diese Rollen in absehbarer Zeit ersetzen wird. Stattdessen sehe ich KI als ein Werkzeug, das DBAs und Entwickler bei komplexeren Aufgaben wie dem Query-Tuning, der Schema-Optimierung und der Einhaltung immer strengerer Datensicherheitsvorschriften unterstützen kann. KI kann ihnen helfen, indem sie unter anderem einen ersten Entwurf eines Codes erstellen oder einen ersten Durchlauf bei der Definition eines Schemas für eine neue App machen kann. KI wird jedoch noch lange Zeit eine menschliche Aufsicht oder Überprüfung erfordern.

Welche Indikatoren sollten die Entscheidung eines Unternehmens leiten, KI-Logik in die Datenbank zu integrieren, anstatt sie extern über Microservices oder cloud-native Laufzeiten zu orchestrieren?

Datenbankinterne KI/ML ist definitiv der beste Ansatz für geschäftskritische, latenzempfindliche und hochsichere Operationen. Dazu zählen Echtzeit-Kredit-Scoring bei Transaktions-Commits oder datenbankinterne Anomalie-Erkennung für sofortige Warnungen. Wenn sich die benötigten Daten bereits in der Datenbank befinden, ist es viel sinnvoller, die KI an die Daten zu bringen, anstatt die Daten an die KI zu senden. Schließlich vereinfacht die Minimierung der Datenverschiebung die Audit-, Verschlüsselungs- und Zugriffskontrollen.

Die Orchestrierung von KI über Microservices oder cloud-native Laufzeiten kann sinnvoll sein, wenn die KI-Logik Daten aus mehreren Quellen anstelle nur einer einzigen Datenbank integrieren muss oder wenn Sie KI/ML-Frameworks, Bibliotheken oder Sprachen benötigen, die in der Datenbank nicht nativ unterstützt werden. Ein weiterer Grund könnte sein, dass Sie GPU-basierte Hardware nutzen möchten, die üblicherweise nicht auf der Datenbankebene zu finden ist.

Welche KI-Funktionen sollten im Zuge der Entwicklung (Vektorsuche, RAG, Orchestrierung wie MCP) strategisch innerhalb der Datenbank vorhanden sein und welche sollten als separate Dienste oder Funktionen außerhalb bleiben (d. h. absichtlich entkoppelt für Geschwindigkeit, Kostenkontrolle oder unabhängige Skalierung)?

Die Vektor- oder Ähnlichkeitssuche funktioniert definitiv am besten innerhalb der Datenbank, insbesondere wenn die Vektoren dort gespeichert werden. Oder wenn Sie Vektor-Suchergebnisse mit aktuellen Transaktions- oder Geschäftsdatensuchen verknüpfen müssen. Ich würde auch empfehlen, schlanke, vorab genehmigte KI/ML-Modelle direkt in SQL oder PL/SQL für latenzempfindliche Workloads oder kritische Daten zu verwenden, die nicht außerhalb der Datenbank versendet werden können.

Die meisten RAG-Pipelines (Retrieval-Augmented Generation) nutzen sowohl die Datenbank als auch separate KI-Dienste (LLMs). Sie nutzen die Datenbank für den Kontextabruf über eine Kombination aus Vektorsuche und traditioneller Geschäftsdatensuche, generieren aber tatsächlich Antworten extern über einen LLM-Dienst, der entweder in der Public Cloud oder auf einem Mid-Tier-Dienst läuft. Dies ist sinnvoll, da LLMs oft von spezialisierter Hardware profitieren, die normalerweise nicht auf Datenbankservern verfügbar ist.

Auch wenn KI immer ausgefeilter wird – insbesondere bei Orchestrierungs-Frameworks wie MCP und autonomen Agenten –, gelten die gleichen Regeln. Alles, was mit Datenintegrität, Sicherheit oder Leistung zusammenhängt, sollte in der Datenbank gespeichert sein. Während das übergeordnete KI-Verhalten – wie Orchestrierung, Argumentation, mehrstufige Aufgabenausführung und systemübergreifende Automatisierung – in der Regel besser außerhalb der Datenbank gehalten wird.

Welche Annahmen über den Oracle Optimizer hören Sie noch in der Praxis – und welche davon sind heute einfach falsch oder waren nie wahr?

Im Laufe der Jahre habe ich viele Dinge über den Oracle Optimizer gehört und vieles davon war völlig übertrieben oder strikt falsch.

Zum Beispiel: „Der Optimizer wählt immer den falschen Plan." Das war nie wahr. Wenn der Optimizer einen „schlechten" Plan auswählt, liegt dies fast immer daran, dass er unvollständige oder irreführende Informationen erhalten hat – typischerweise fehlende oder veraltete Statistiken, verzerrte Daten oder falsche Kardinalitätsannahmen. Mit der heutigen automatischen Erfassung von Statistiken, dynamischem Sampling und adaptiven Plänen sowie Funktionen wie Auto SPM bringt der Optimizer die Dinge viel öfter in Ordnung, als ihm zugestanden wird.

Ein weiteres häufiges Missverständnis ist: „Hints sind die einzige Möglichkeit, den Optimizer zu steuern." Wenn Sie tatsächlich viele Hints hinzufügen, geben Sie Ihnen oft weniger Vorhersehbarkeit, nicht mehr. Hints können vergänglich sein, wenn sich das Schema, die Statistiken oder die Hardware ändern. Der kostenbasierte Optimizer ist so konzipiert, dass er flexibel ist und sich an Änderungen des Schemas oder des Datenvolumens anpasst. Das Erzwingen eines Plans durch Hints verhindert, dass der Optimizer Pläne entwickelt, wenn sich die Dinge ändern. Die Nutzung von SQL-Plan-Baselines ist ein weitaus nachhaltigerer Ansatz als das Streuen von Hints überall. Baselines ermöglichen es dem Optimizer, kontrolliert Pläne zu entwickeln, wenn sich die Dinge ändern. Er wird den neuen Plan nur verwenden, wenn er sich als besser als der bestehende Baseline-Plan erweist.

Wird Oracle Optimizer mit der Entwicklung von KI und der Unterstützung von KI-Assistenten auch in Zukunft noch benötigt?

Ich glaube nicht, dass der Oracle Optimizer in absehbarer Zeit irgendwohin gehen wird, was einige Leute enttäuschen könnte. Obwohl KI-Assistenten helfen können, Abfragen zu schreiben oder die Umgebung zu optimieren, benötigt die Datenbank immer noch eine vertrauenswürdige, deterministische Engine, um SQL in effiziente Datenoperationen zu übersetzen. Das bedeutet nicht, dass der Optimizer nicht von KI oder ML profitieren kann, aber es ist wahrscheinlich, dass er in Form verbesserter Optimizer-Statistiken kommt, wie zum Beispiel automatische Anomalie-Erkennung in Datendistributionen oder Planstabilität, die durch Funktionen wie automatisches SQL-Plan-Management und automatische Indexierung ermöglicht wird.

Welche konkreten Entscheidungskriterien empfiehlt Oracle jenseits von „Es kommt darauf an" für die Wahl zwischen On-Premise, Public Cloud und Sovereign Cloud?

Oracle hat seine Kunden immer ermutigt, die Bereitstellungsoption zu wählen, die für sie am besten geeignet ist, und zwar basierend auf zwei Kriterien: wo sie ihre Datenbanken ausführen möchten und wer sie verwalten möchte. Zum Beispiel eignet sich unsere Sovereign Cloud hervorragend für Unternehmen, die strenge gesetzliche oder regulatorische Vorschriften haben, die verlangen, dass ihre Daten innerhalb bestimmter geografischer oder politischer Grenzen bleiben. Wenn Sie eine umfassende Kontrolle über Dinge wie Firewalls, Load Balancer, Hardware oder private Netzwerke benötigen, ist ein lokales Setup oder Oracle Exadata Cloud@Customer möglicherweise sinnvoller. Und wenn Sie sich lieber nicht mit der Verwaltung der Datenbank oder der Infrastruktur befassen möchten, können Sie

einfach Oracle Autonomous Database verwenden – sei es in der Public Cloud (OCI, Azure, Google oder AWS) oder lokal mit Autonomous Database auf Exadata Cloud@Customer. Und natürlich haben Sie immer die Freiheit, nahtlos zwischen diesen verschiedenen Bereitstellungsoptionen zu wechseln, da Oracle die gleiche Oracle-Datenbanksoftware für alle diese Bereitstellungsoptionen verwendet.

Welche konkreten Mechanismen empfehlen Sie, um Überraschungsrechnungen in Autonomous oder generell in der Cloud zu verhindern?

Oracle bietet mit der Oracle Autonomous Database mehrere Mechanismen, mit denen Kunden Überraschungen bei ihren Rechnungen vermeiden können. **Auto-Skalierung** ist die naheliegendste Option. Wenn Sie die automatische Skalierung aktivieren, skaliert das System automatisch auf das Dreifache der Basisanzahl von ECPUs, wenn Ihre Arbeitslast zusätzliche Ressourcen benötigt. Sie zahlen nur für die zusätzliche Rechenleistung, während sie verwendet wird. Sobald sich Ihr Workload verringert, skaliert das System automatisch auf die Basisanzahl von ECPUs zurück. **Automatisches Stoppen und Starten** für Oracle Autonomous Database ist eine weitere großartige Funktion zur Kontrolle Ihrer Kosten, da Sie Ausfallzeiten planen können, insbesondere für Entwicklungs- oder Nicht-Produktionsdatenbanken, die nicht rund um die Uhr benötigt werden. Wenn die Datenbank gestoppt wird, müssen Sie nicht für die Rechenleistung bezahlen. Sie zahlen nur für die Speicherung.

Elastic Pools ist eine dritte wichtige Funktion, die zur Kostensenkung beiträgt. Sie können sich einen Elastic Pool als „Familienplan" für Ihre autonomen Datenbanken vorstellen. Anstatt für jede Datenbank einzeln zu bezahlen, können Sie sie in einen logischen Pool gruppieren, in dem Ihnen die Nutzung der Rechenleistung des gesamten Pools in Rechnung gestellt wird. Ein Elastic Pool ermöglicht es Ihnen, das bis zu Vierfache der ECPU-Zahl Ihrer Poolgröße bereitzustellen. Wenn Sie beispielsweise einen Pool mit einer Poolgröße von 128 ECPUs haben, können Sie bis zu 512 ECPUs in diesem Pool bereitstellen. Eine weniger bekannte, aber ebenso vorteilhafte Funktion ist die **sekundengenaue Abrechnung** der Oracle Autonomous Database (für die ECPU-Nutzung), die dazu beiträgt, die Kosten eng an der tatsächlichen Nutzung auszurichten. Und wenn Sie Ihre autonome Datenbank auf OCI betreiben, können Sie ein Budget für Ihre Datenbank festlegen und sich von Oracle automatisch benachrichtigen lassen, wenn Sie dieses Budget überschreiten.

Was steht für die nächsten 12 Monate auf der Roadmap?

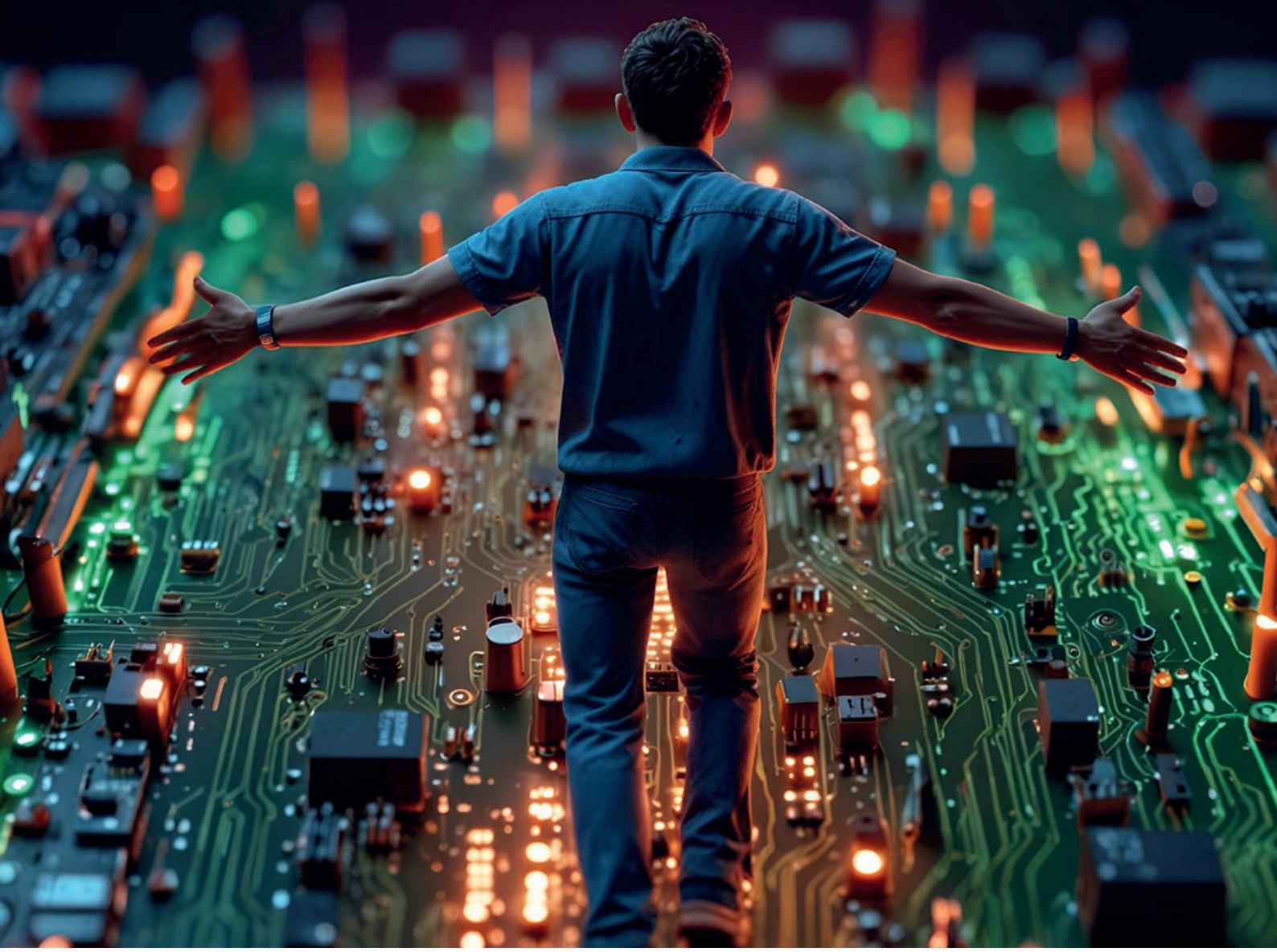
Das Oracle Database-Team hat einige aufregende neue Funktionen und Tools auf unserer kurzfristigen Roadmap. Auf der letzten AI World in Las Vegas haben wir **Oracle Private AI Services Container** angekündigt, eine vorgefertigte, getestete Umgebung für die Ausführung privater Instanzen von KI-Modellen, einschließlich Einbettungsmodellen, offenen LLMs und Named Entity Recognizern. Der Container wird dazu beitragen, die Sicherheit der KI-Workloads zu verbessern, indem Kunden die Weitergabe von Daten an KI-Drittanbieter vermeiden können. Der Container kann an jedem beliebigen Ort bereitgestellt werden, innerhalb Ihres eigenen Mandantenbereichs in der Public Cloud, in privaten Clouds oder lokal.

Ein weiteres aufregendes neues Tool, das auf der AI World vorgestellt wurde, war **Oracle Private AI Agent Factory**. Die Factory ist eine No-Code-Benutzeroberfläche mit Drag-and-Drop (ähnlich wie Langflow), mit der Entwickler intelligente Agenten mit benutzerdefinierten Workflows, Tools und Datenökosystemen schnell bereitstellen können.

Trusted Answer Search ist ebenfalls in Kürze verfügbar. Dieses kostenlose Tool hilft Kunden, eine Schnittstelle in natürlicher Sprache für eine Anwendung zu erstellen, die die Vektorsuche anstelle eines LLM verwendet. Endbenutzerfragen werden in Vektoren umgewandelt, und die Vektorsuche ordnet den Fragenvektor dem nächstgelegenen Berichtsbeschreibungsvektor zu, wodurch der auszuführende Bericht ausgelöst wird, und die Ergebnisse werden an den Benutzer gesendet.

MARIA COLGAN

Maria Colgan ist Vice President of Product Management für KI und unternehmenskritische Daten im Datenbankteam von Oracle – eine Position, die sie seit fast dreißig Jahren im Unternehmen innehat. Sie ist, was manche als Datenbank-Nerd bezeichnen würden, und unterstützt Unternehmen mit Leidenschaft dabei, die nötige Flexibilität und Ausfallsicherheit in einer sich ständig wandelnden Welt zu erreichen. Was ihr an ihrem Job am meisten gefällt? Kundenwünsche zu verstehen und ihre Ideen in zukünftige Versionen einfließen zu lassen. Außerdem schreibt sie den SQLMaria-Blog (genau, unter *https://sqlmaria.com*), auf dem sie Tipps, Einblicke und jede Menge Datenbankwissen teilt.



Neue Transaktionsansätze in Oracle AI Database 26ai: Reservierungen, Prioritäten und mehr

Rainer Willems, Oracle Deutschland

MVCC – die Multiversion Concurrency Control – hat sich in der Oracle-Datenbank seit vielen Jahren als zuverlässiger Mechanismus zur Sicherstellung von Data Concurrency und Data Consistency bewährt. Allerdings kann die strikte Isolation von Transaktionen in bestimmten Szenarien auch eine Herausforderung darstellen – insbesondere dann, wenn Anwendungen nicht auf dedizierten Datenbank-Sessions basieren. Mit dem Release Oracle AI Database 26ai stehen nun neue Funktionen zur Verfügung, die genau an diesem Punkt ansetzen und den Umgang mit solchen Situationen deutlich vereinfachen.

Herausforderung 1: Kommt es vor, dass viele Nutzer gleichzeitig die gleiche Spalte in derselben Tabellenzeile ändern möchten, führen die dabei gesetzten Sperren zwangsläufig zu Wartezeiten.

Stellen wir uns ein Szenario mit Konzerttickets für Stehplätze oder mit Lagerbeständen von Produkten vor. Bei jedem Ticketverkauf oder Produktkauf wird dieselbe Information in der Datenbank geändert. In solchen Anwendungen greifen zahlreiche Transaktionen gleichzeitig auf diese Daten zu – sie werden dadurch schnell zu sogenannten *heißen Ressourcen*, da sie fortlaufend von vielen Prozessen gelesen und aktualisiert werden. Die Folge sind häufige Sperren und spürbare Wartezeiten.

Mit der in Oracle AI Database 26ai eingeführten **Lock-Free-Reservierung** lässt sich dieses Problem elegant umgehen. Sie ermöglicht Transaktionen, Änderungen an derselben Spalte vorzunehmen, ohne sich gegenseitig zu blockieren. Dazu wird eine Spalte als ***RESERVABLE*** definiert und mit einem *Check-Constraint* versehen, der sicherstellt, dass ausreichend Bestand vorhanden ist, bevor eine Reservierung vorgenommen werden darf.

Die Reservierungen selbst werden dabei nicht direkt in der Datenspalte gespeichert (und verursachen daher auch keine Sperren), sondern in einer **Journal-Tabelle** verwaltet. Diese Tabelle ermöglicht es der Datenbank zu prüfen, ob eine neue Reservierung noch zulässig ist. Wenn alle bestehenden Reservierungen tatsächlich ausgeführt würden (also nach einem *COMMIT*) und eine weitere Reservierung das definierte Constraint verletzen würde, wird diese automatisch abgelehnt.

Das Verfahren funktioniert nur bei relativen Änderungen einer numerischen Spalte – also bei einem „+“ oder „-“ im Vergleich zum Ausgangswert. Zusätzlich gilt: In einer Reservierung dürfen ausschließlich *RESERVABLE*-Spalten geändert werden (keine Mischung mit normalen Spalten), und der Primärschlüssel muss in der *WHERE*-Bedingung angegeben sein.

Listing 1 zeigt ein Beispiel dafür, der Ablauf wird in *Listing 2* demonstriert.

Auch mehrere Spalten einer Tabelle können als RESERVABLE definiert werden. In diesem Fall erweitert sich die Journal-Tabelle automatisch um zusätzliche Spalten, in denen sowohl die jeweilige Operation (+ oder –) als auch der Änderungswert protokolliert werden.

```
CREATE TABLE cinema
   ( id          NUMBER  PRIMARY KEY,
     film       VARCHAR2(30),
     capacity    NUMBER RESERVABLE CHECK (capacity >= 0),
     room        NUMBER);

INSERT INTO cinema VALUES (1,'A Fish Called Wanda',10,1);
```

Listing 1: Tabelle Cinema mit RESERVABLE-Spalte capacity

Herausforderung 2: In jeder Datenbank können Sperren (Locks) auftreten. Problematisch wird es, wenn kritische Transaktionen blockiert werden, weil sie auf weniger wichtige Vorgänge warten müssen – teilweise zu lange.

In solchen Fällen könnte ein DBA eingreifen, die blockierende Session identifizieren und manuell beenden. Doch das ist mühsam und fehleranfällig. Genau hier setzt Oracle AI Database 26ai mit dem neuen Konzept der **Priority Transactions** an.

Jede Session – und damit auch ihre Transaktionen – kann einer bestimmten Priorität zugeordnet werden. Kommt es zu einem Konflikt, werden nach Ablauf einer festgelegten Wartezeit Transaktionen mit niedrigerer Priorität automatisch zurückgesetzt – ganz ohne Eingreifen des DBAs.

Es stehen drei Prioritätsstufen zur Verfügung: *HIGH, MEDIUM* und *LOW*. Auf Systemebene wird festgelegt, wie lange Transaktionen mit höherer Priorität auf solche mit niedriger Priorität warten, bevor der Mechanismus aktiv wird:

ALTER SYSTEM SET priority_txns_high_wait_target = <secs>;
ALTER SYSTEM SET priority_txns_medium_wait_target = <secs>;

Das Verhalten kann zunächst im Tracking-Modus getestet werden. Dabei werden potenzielle Eingriffe lediglich protokolliert, ohne dass tatsächlich ein Rollback erfolgt:

ALTER SYSTEM SET priority_txns_mode = TRACK;

Soll die Funktion aktiv genutzt werden, genügt ein Umschalten auf den Rollback-Modus:

ALTER SYSTEM SET priority_txns_mode = ROLLBACK;

Ein einfaches Beispiel dazu zeigt *Listing 3*.

In der View *V$TRANSACTIONS* lassen sich die Spalten *TXN_PRIORITY* und *PRIORITY_TXNS_WAIT_TARGET* einsehen. In *V$SYSSTAT* wiederum wird erfasst, wie oft Transaktionen mit hoher *(HIGH)* oder mittlerer *(MEDIUM)* Priorität anderen Transaktionen ihre Locks entzogen haben. Im Alert Log kann dieses Verhalten anschließend im Detail nachvollzogen werden.

Herausforderung 3: Moderne Anwendungen sind heute meist *stateless*. Solange eine Anwendung über eine durchgehende Datenbank-Session verfügt, lassen sich Transaktionen relativ einfach handhaben. Doch was passiert beispielsweise bei einer REST-basierten Anwendung, die bei jedem Aufruf eine andere Verbindung zur Datenbank nutzt?

Zwar existieren Transaction Manager, die Transaktionen koordinieren können – mit Oracle AI Database 26ai lassen sich ähnliche Anforderungen nun direkt in der Datenbank lösen, ganz ohne zusätzliche Software.

```
<<Session 1>>
UPDATE cinema SET capacity = capacity - 4 WHERE id = 1;
1 row updated.
/* Dies ist aber nur eine Reservierung.
   In der Tabelle ist diese Reservierung noch nicht sichtbar. */

SELECT * FROM cinema;
ID FILM                CAPACITY ROOM
-- ------------------- -------- ----
 1 A Fish Called Wanda       10    1

/* Die Reservierung ist im Journal vermerkt, wobei 101257 die Object-ID der Tabelle CINEMA ist. */

SELECT ORA_STATUS$ STATUS,
       ORA_STMT_TYPE$ TYPE,
       ID,
       CAPACITY_OP OP,
       CAPACITY_RESERVED RES
FROM SYS_RESERVJRNL_101257;

STATUS TYPE   ID OP RES
------ ------ -- -- ---
ACTIVE UPDATE  1 -    4

<<Session 2>>
UPDATE cinema SET capacity = capacity - 4 WHERE id = 1;
1 row updated.

/* Das Update wurde nicht blockiert. Beide Sessions sehen in der Tabelle weiterhin den Wert 10 für die ca-
pacity, aber es gibt jetzt zwei Reservierungen. */

<<Session 3>>
UPDATE cinema SET capacity = capacity - 4 WHERE id = 1;
ORA-02290: check constraint (SCOTT.SYS_C0013442) violated

/* Keine weitere Reservierung möglich, da durch das Erfüllen aller Reservierungen das Constraint verletzt
würde */
```

Listing 2: Lock-free Reservations

```
<<System>>
ALTER SYSTEM SET priority_txns_high_wait_target = 5;
/* HIGH wartet 5 Sekunden auf MEDIUM und LOW */
ALTER SYSTEM SET priority_txns_medium_wait_target =10;
/* MEDIUM wartet 10 Sekunden auf LOW */
ALTER SYSTEM SET priority_txns_mode = ROLLBACK;

<<Session 1>>
ALTER SESSION SET txn_priority = low;
UPDATE emp SET sal = 0;
/* Damit werden alle Zeilen in emp gelockt */

<<Session 2>>
ALTER SESSION SET txn_priority = high;
UPDATE emp SET sal = 400 WHERE ename = 'KING';
/* 5 Sekunden vergehen */
1 row updated.

<<Session 1>>
SELECT * FROM emp;
ORA-63302: Transaction must roll back
ORA-63300: Transaction is automatically rolled back since it is blocking a higher priority transaction
from another session.
```

Listing 3: Priority Transactions

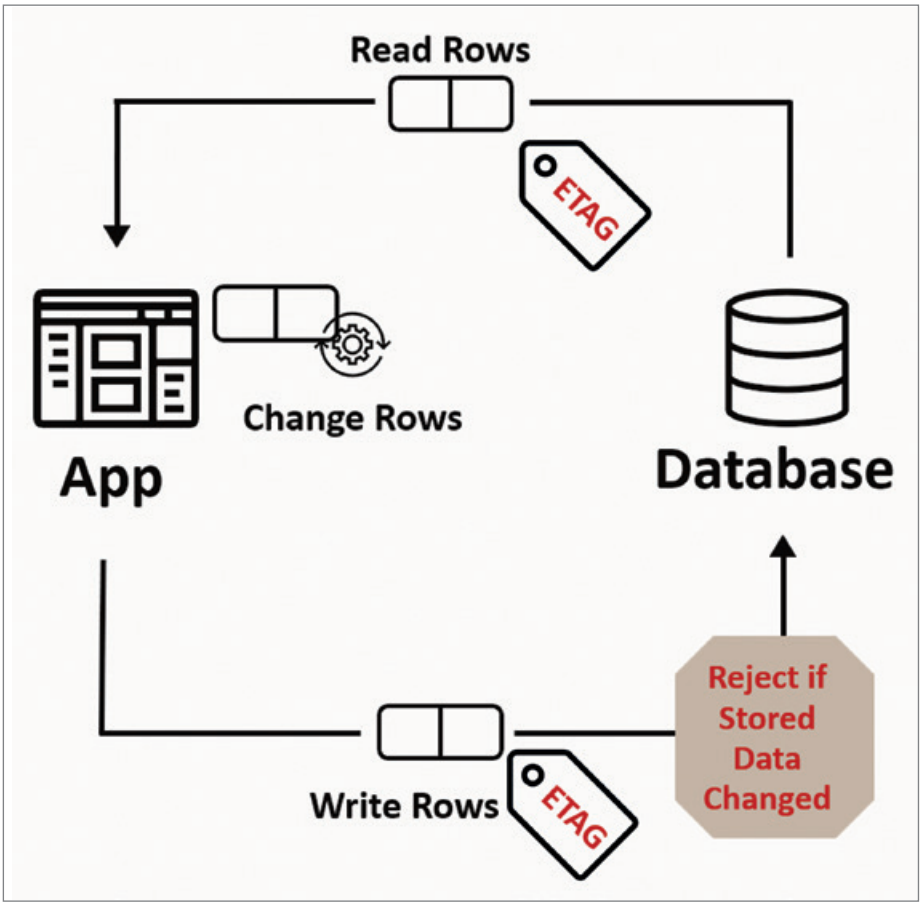


Abbildung 1: ETags für eine optimistische Concurrency Control (Quelle: Rainer Willems)

ändert, erkennt die Datenbank dies anhand des geänderten ETags und kann so eine optimistische Concurrency Control umsetzen – also ein Verfahren, das Konflikte vermeidet, ohne Sperren zu benötigen. Ein Beispiel dazu zeigt *Listing 4*.

Ein ETag kann nicht nur auf einzelne Zeilen, sondern auch über mehrere Zeilen und sogar tabellenübergreifend eingesetzt werden. So lässt sich beispielsweise sicherstellen, dass Änderungen nur dann durchgeführt werden, wenn sich abhängige Daten in der Zwischenzeit nicht verändert haben.

Listing 5 zeigt ein solches Szenario: Die Lokation einer Abteilung in der Tabelle DEPT darf nur dann geändert werden, wenn zwischen dem Lesen und Schreiben des Datensatzes weder das Gehalt noch die Position eines Mitarbeiters dieser Abteilung in der Tabelle *EMP* geändert wurde.

JSON Relational Duality Views und JSON Collection Tables nutzen dieses ETag-Konzept, um Änderungen an Daten effizient zu erkennen und abzusichern.

Ein weiterer innovativer Ansatz wurde mit den **Sessionless Transactions** eingeführt, die ab der Version 23.6 verfügbar sind. Sie ermöglichen es, eine laufende Transaktion vorübergehend zu unterbrechen *(suspend)* und später – auch in einer anderen Session – wieder aufzunehmen *(resume)*.

```
/* beim Lesen wird der ETag für die angegebenen Spalten ermittelt */
SELECT ename, job , sal, comm,
       SYS_ROW_ETAG(empno, job, sal)
  INTO :name, :beruf, :gehalt, :provision, :myetag
  FROM emp WHERE empno = 7844;

/* Beim Schreiben wird der beim Lesen ermittelte ETag mit dem aktuellen ETag verglichen */
UPDATE emp
   SET sal = <a new value>
 WHERE empno = 7844
   AND SYS_ROW_ETAG(empno, job, sal) = :myetag

0 rows updated.
/* Falls empno, job oder sal zwischen Lesen und Schreiben von einer anderen Transaktion geändert wurden.
*/

1 row updated.
/* Falls die Daten in der Zwischenzeit nicht geändert wurden. Die Spalte comm kann geändert worden sein,
da sie im ETag nicht berücksichtigt wurde */
```

Listing 4: ETags – Optimistische Kontrolle (Concurrency Control?) von Änderungen

Ein zentrales neues Konzept dabei sind die **ETags** (kurz für Entity Tags). Sie dienen dazu, einer Ressource eine Art eindeutigen Fingerabdruck zuzuweisen, anhand dessen sich feststellen lässt, ob sich diese Ressource verändert hat.

Im Datenbankkontext bedeutet das: Eine bestimmte Datensammlung erhält beim Lesen einen ETag *(siehe Abbildung 1)*, der beim anschließenden Schreiben überprüft wird. Hat sich der zugrunde liegende Datensatz in der Zwischenzeit ge-

```
/* Ermittlung eines aggregierten ETags für alle Zeilen der Abteilung 10 in der Tabelle emp */
SELECT SYS_AGG_ETAGS (SYS_ROW_ETAG(sal, job))
  INTO :myetag
   FROM emp
 WHERE deptno = 10;

…

/* Update der Tabelle dept – ETag-Vergleich über abhängige Zeilen in Tabelle emp */
UPDATE dept d
  SET loc = 'TRIER'
WHERE deptno = 10
  AND ( SELECT SYS_AGG_ETAGS (SYS_ROW_ETAG (sal, job))
          FROM emp
         WHERE deptno = 10) = :myetag;

1 row updated.
/* Falls bei keinem Mitarbeiter in Abteilung 10 in der Zwischenzeit Gehalt oder Job geändert wurde */

0 rows updated.
/* Falls sich das Gehalt oder der Job eines Mitarbeiters geändert hat */
```

Listing 5: ETags über mehrere Tabellen und Zeilen

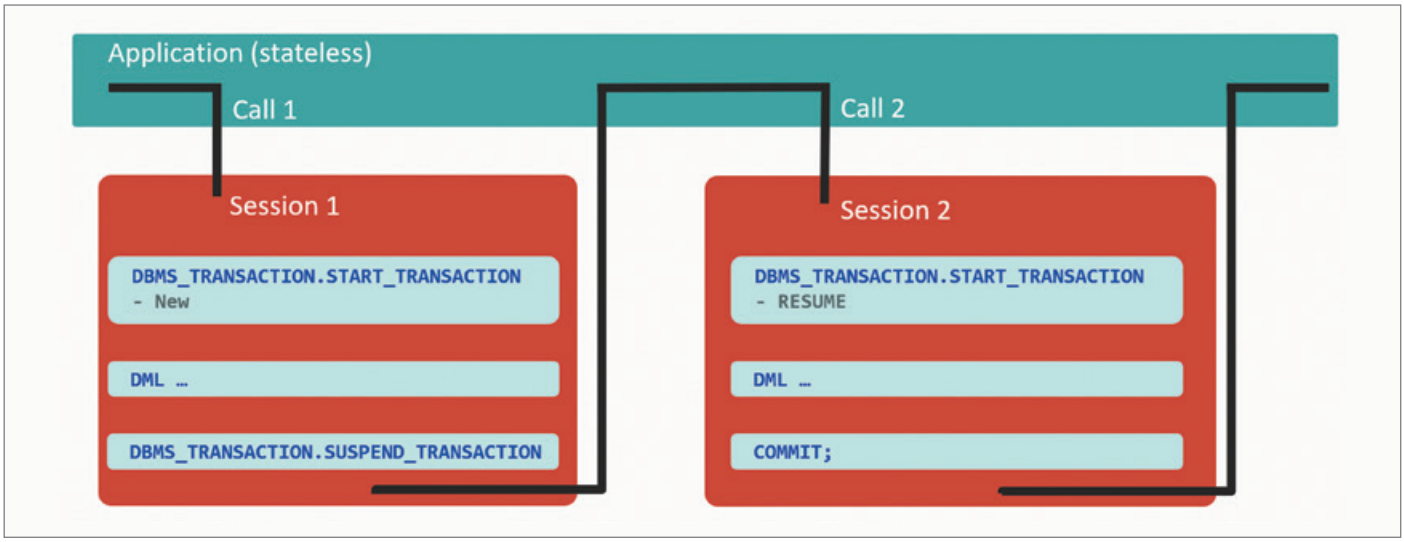


Abbildung 2: Transaktionen starten, unterbrechen und wieder aufnehmen (Quelle: Rainer Willems)

```
<<Session 1>>

/* Eine neue Transaktion wird gestartet. Sie erhält einen Identifier und ein Timeout (hier 30 Sekunden). */
DECLARE
  ta_id VARCHAR2(128);
BEGIN
  ta_id := DBMS_TRANSACTION.START_TRANSACTION
     ( UTL_RAW.CAST_TO_RAW('myidentifier')
     , DBMS_TRANSACTION.TRANSACTION_TYPE_SESSIONLESS
     , 30
     , DBMS_TRANSACTION.TRANSACTION_NEW
     );
END;
/

INSERT INTO mytable VALUES (1, 'One'), (2,'Two');
2 rows inserted.

/* Die Transakion wird ausgesetzt (suspend) */
```

```
exec DBMS_TRANSACTION.SUSPEND_TRANSACTION;

SELECT * FROM mytable;
/* Zeigt die beiden eingefügten Zeilen jetzt nicht mehr */

<<Session 2>>
SELECT * FROM mytable;
/* Zeigt die beiden eingefügten Zeilen ebenfalls nicht */

/* Die Transaktion wird wieder aufgenommen (resume) und is jetzt in dieser Session sichtbar */
DECLARE
    ta_id VARCHAR2(128);
BEGIN
    ta_id := DBMS_TRANSACTION.START_TRANSACTION
    ( UTL_RAW.CAST_TO_RAW('myidentifier')
    , DBMS_TRANSACTION.TRANSACTION_TYPE_SESSIONLESS
    , 30
    , DBMS_TRANSACTION.TRANSACTION_RESUME
    );
END;
/

SELECT * FROM mytable;
/* Zeigt die beiden eingefügten Zeilen */

SELECT
UTL_RAW.CAST_TO_VARCHAR2(DBMS_TRANSACTION.GET_TRANSACTION_ID());
myidentifier
/* Der Identifier der Session kann ausgelesen werden */

COMMIT;
/* Beendet die Transaktion, die in Session 1 gestartet wurde */
```

Listing 6: Sessionless Transactions

Damit können selbst stateless-Anwendungen, die bei jedem Aufruf eine neue Datenbankverbindung aufbauen, komplexe Transaktionen durchführen, ohne auf einen externen Transaction Manager angewiesen zu sein. *Listing 6* zeigt exemplarisch, wie eine Transaktion von einer Session auf eine andere übertragen werden kann und die *Abbildung 2* zeigt den Ablauf.

Dieses Konzept lässt sich problemlos in RESTful Services und ähnliche Architekturen integrieren. Dadurch können komplexe Transaktionen direkt in der Datenbank abgewickelt werden – ohne bei jedem einzelnen Aufruf ein separates COMMIT ausführen zu müssen. Derzeit werden Session-Einstellungen, wie beispielsweise die NLS-Parameter, noch nicht automatisch übertragen.

Fazit

Mit der Oracle AI Database 26ai präsentiert Oracle eine Reihe spannender Neuerungen im Bereich der Transaktionsverwaltung. Lock-Free-Reservierungen, Priority Transactions, ETags und Sessionless Transactions bieten Entwicklern und Architekten neue Werkzeuge, um die Performance, Skalierbarkeit und Benutzerfreundlichkeit anspruchsvoller Anwendungen zu verbessern.

Gerade in modernen, verteilten und häufig stateless aufgebauten Systemen eröffnen diese Technologien neue Möglichkeiten, Transaktionen effizienter zu steuern.

Die Datenbank ist längst mehr ist als nur ein Speicherort für Daten – sie ist ein aktiver und intelligenter Bestandteil der Anwendung.

Über den Autor

Rainer Willems ist Account Cloud Engineer bei Oracle und beschäftigt sich schwerpunktmäßig mit Entwicklerthemen rund um die Datenbank.

Rainer Willems
rainer.willems@oracle.com

Manuelle Datenmigration von Oracle 11 zu Oracle 19c

Lena Rudenko, Prodata – a Dataciders Company

In diesem Artikel wird eine Schritt-für-Schritt-Anleitung vorgestellt, die es ermöglicht, die Oracle-11-Datenbank auf die Oracle 19c zu migrieren und das Problem mit der NLS_LENGTH_SEMANTICS zu umgehen.

Dieser Artikel (und zuvor ein Vortrag auf der DOAG-Konferenz) ist aus einer recht eigenartigen Situation entstanden, die vermutlich nicht sehr häufig vorkommt. Aber genau deswegen könnten unsere Erfahrung und die von uns ausgearbeitete Lösung – eigentlich eher ein Workaround – für diejenigen interessant sein, die vor derselben Herausforderung stehen.

Die Ausgangssituation sieht wie folgt aus: Wir haben mehrere mittelgroße Datenbanken der Version 11.2.0.3, die auf einem Windows-2008-Betriebssystem laufen. Es besteht keine Möglichkeit, die Datenbanken auf Version 11.2.0.4 zu patchen, was in manchen Fällen ein automatisches Upgrade auf Oracle 19c erlauben würde. Da wir uns aber auf Windows befinden, ist dieser Weg ausgeschlossen. Die Aufgabe besteht darin, die Datenbanken auf Oracle 19 und Linux zu migrieren.

Ein gut bekanntes Tool dafür ist Oracle Data Pump, ein leistungsstarkes Werkzeug, das den schnellen Export und Import von Daten und Metadaten zwischen Oracle-Datenbanken ermöglicht. Idealerweise werden Export- und Import-Dateien (.par) definiert und anschließend die Befehle `expdp system/<password> parfile=<file_name>.par` sowie `impdp system/<password> parfile=<file_name>.par` ausgeführt, um das gewünschte Ergebnis zu erhalten. Leider leben wir nicht in einer idealen Welt und Probleme können sogar zwischen Datenbanken der gleichen Version auftreten. In unserem Fall erhielten wir beim Versuch, den impdp-Befehl auszuführen, eine sehr große Anzahl an Fehlermeldungen *(siehe Abbildung 1)*.

Der Grund für **ORA-12899 (value too large)** liegt in `NLS_LENGTH_SEMANTICS` – einem Datenbankparameter, der festlegt, wie Oracle die Länge von `CHAR`- und `VARCHAR2`-Spalten misst – entweder in **bytes** oder in **characters**. Dieser Parameter ist

```
ORA-02374: conversion error loading table "DOAGTEST"."PBV_LIEFERSCHEINE_POS"
ORA-12899: value too large for column DBPBVP_ART_BEZ (actual: 31, maximum: 30)

ORA-02372: data for row: DBPBVP_ART_BEZ : 0X'475554204C414E47452048E4686E6368656E20496E6E656E66'


ORA-02374: conversion error loading table "DOAGTEST"."PBV_LIEFERSCHEINE_POS"
ORA-12899: value too large for column DBPBVP_ART_BEZ (actual: 31, maximum: 30)

ORA-02372: data for row: DBPBVP_ART_BEZ : 0X'475554204C414E47452048E4686E6368656E2D427275737466'


ORA-02374: conversion error loading table "DOAGTEST"."PBV_LIEFERSCHEINE_POS"
ORA-12899: value too large for column DBPBVP_ART_BEZ (actual: 31, maximum: 30)

ORA-02372: data for row: DBPBVP_ART_BEZ : 0X'475554204C414E4745204765666CFC67656C204E7567676574'


ORA-02374: conversion error loading table "DOAGTEST"."PBV_LIEFERSCHEINE_POS"
ORA-12899: value too large for column DBPBVP_ART_BEZ (actual: 31, maximum: 30)

ORA-02372: data for row: DBPBVP_ART_BEZ : 0X'475554204C414E47452048E4686E6368656E204F6265726B65'
```

Abbildung 1: Fehlermeldungen nach der Ausführung des impdp-Kommandos (Quelle: Lena Rudenko)

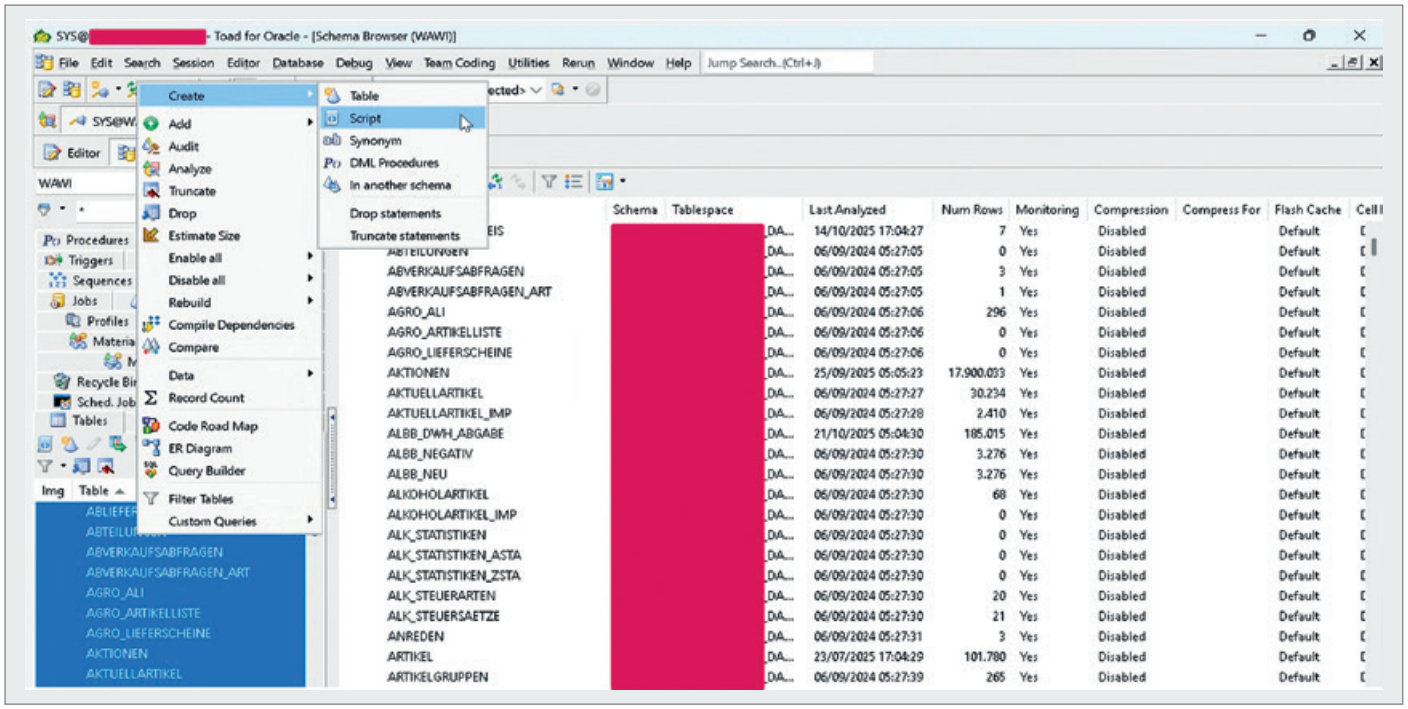


Abbildung 2: Exportieren eines DDL-Skriptes zum Erstellen der Tabellen (Quelle: Lena Rudenko)

Abbildung 3: Ersetzen von BYTE durch CHAR in den Tabellen-Definitionen (Quelle: Lena Rudenko)

in Umgebungen mit Multibyte-Zeichensätzen (z. B. UTF-8) relevant, da die Anzahl der zum Speichern eines Zeichens erforderlichen Bytes variieren kann.

In Oracle 11 wurden viele Schemata `VARCHAR2(n BYTE)` erstellt. Bei AL32UTF8 benötigen manche Zeichen 2–4 Bytes. Dieselbe Spalte (die zuvor immer für 10 Zeichen ausreichte) kann nun zwischen 2–10 Zeichen enthalten. Einfügungen/Aktualisierungen schlagen daher mit ORA-12899 fehl.

Dieses Problem lässt sich nur auf eine Weise lösen: In den Definitionen der Tabellen muss `VARCHAR2(n BYTE)` durch `VARCHAR2(n CHAR)` ersetzt werden. Das bedeutet, dass wir ein vollständiges DDL-Skript für alle Tabellen der Daten-

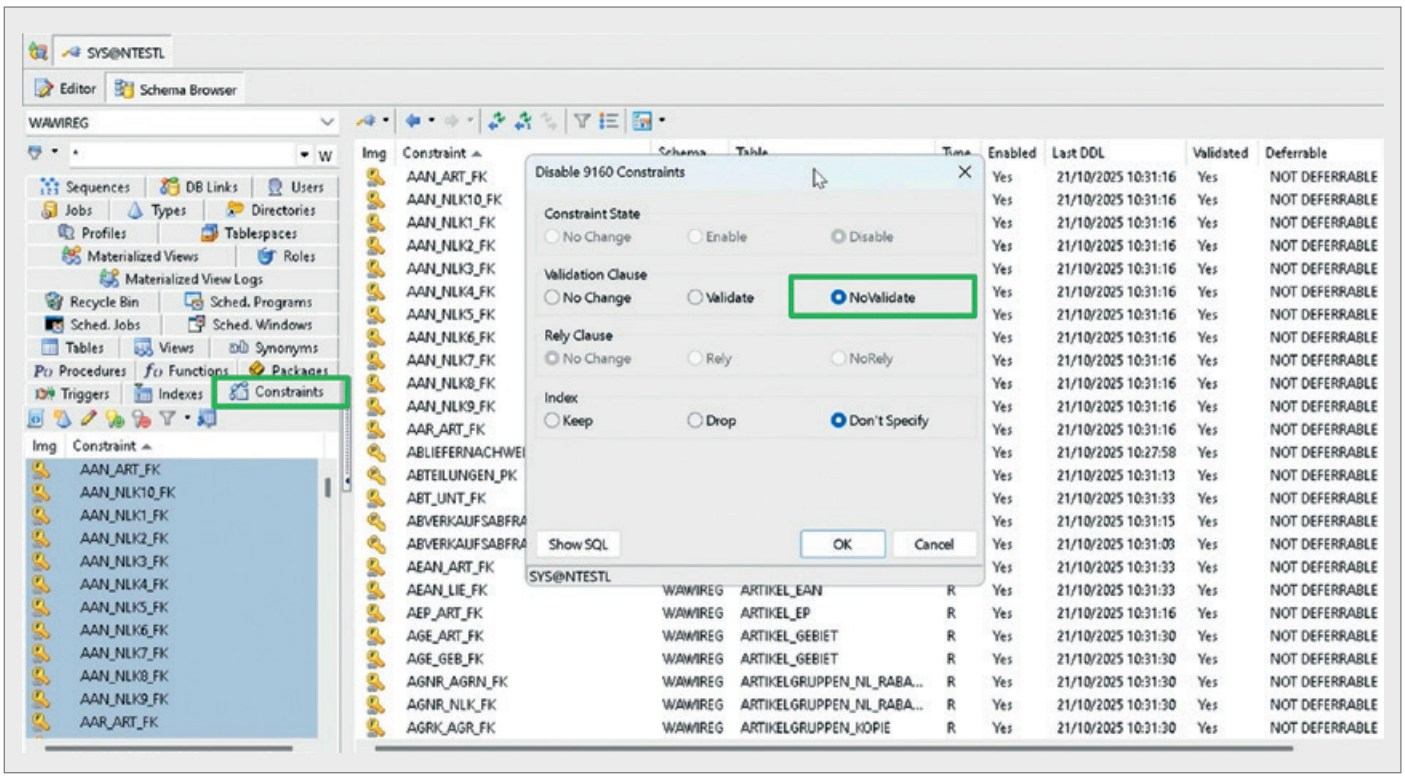


Abbildung 4: Disable Constraints mit NOVALIDATE-Option in Toad (Quelle: Lena Rudenko)

```
ORA-02374: conversion error loading table "WAWIREG"."ZE_KTR_ARTSPEZ"
ORA-12899: value too large for column KTRAS_KTR_SPEZIFIKATION (actual: 4000, maximum: 4000)

ORA-02372: data for row: KTRAS_KTR_SPEZIFIKATION : 0X'2320313535373322D3031204261736963204 16C6C726F756E64'


ORA-02374: conversion error loading table "WAWIREG"."ZE_KTR_ARTSPEZ"
ORA-12899: value too large for column KTRAS_KTR_SPEZIFIKATION (actual: 4000, maximum: 4000)

ORA-02372: data for row: KTRAS_KTR_SPEZIFIKATION : 0X'45696E737465696765722F54656C65205365743A0A416E6765'


ORA-02374: conversion error loading table "WAWIREG"."ZE_KTR_ARTSPEZ"
ORA-12899: value too large for column KTRAS_KTR_SPEZIFIKATION (actual: 4000, maximum: 4000)

ORA-02372: data for row: KTRAS_KTR_SPEZIFIKATION : 0X'2320313535373322D3031204261736963204 16C6C726F756E64'
```

Abbildung 5: ORA-12899 „value too large ... (actual: 4000, maximum: 4000)“ (Quelle: Lena Rudenko)

```
directory=EXPORT
dumpfile=WAWI_%U.DMP
logfile=wawi_imp.log
jobname=wawi_imp
content=DATA_ONLY
```

Listing 1: Inhalt der Import .par-Datei

bank benötigen. Es gibt verschiedene Möglichkeiten, dieses Skript zu erzeugen. Beispielsweise kann man es sich durch Oracle Data Pump generieren lassen, indem man den Import in eine .sql-Datei statt in die Datenbank durchführt. Eine andere Möglichkeit besteht darin, ein zusätzliches Tool (Toad, SQL Developer usw.) zu verwenden und sich mit dessen Hilfe das gewünschte Skript zu exportieren.

Da dieses angepasste Skript mit CHAR-Definitionen zum Erstellen der Tabellen in der neuen Datenbank ausgeführt wird, werden DDL-Kommandos für alle Datenbankobjekte benötigt, obwohl die Anpassung nur Tabellen betrifft. Das ist der Grund, warum die Verwendung eines Tools etwas umständlich ist: Man muss sich ein DDL-Skript für Tabellen, Funktionen, Prozeduren, Sequenzen, Typen, Materialized Views und vieles mehr generieren und später ausführen. Wir haben uns jedoch eine dritte Alternative überlegt. Aus unserer Sicht ermöglicht sie die maximale Verwendung von Oracle-Data-Pump-Instrumenten und somit den möglichst sauberen Import in unserer Situation.

Wir verwenden die exportierten Dump-Dateien und führen einen Metadaten-Import in die neue Datenbank

durch. Das lässt sich durch den Parameter `CONTENT=METADATA_ONLY` regeln. Dieser Schritt läuft bei der Migration von Oracle 11 auf Oracle 19c ebenfalls ohne Probleme, sodass alle Datenbankobjekte mit einem Schritt in der neuen Datenbank verfügbar sind. Die Tabellendefinitionen enthalten jedoch weiterhin BYTE.

Anschließend exportieren wir das DDL-Skript zum Erstellen der Tabellen aus der neuen Datenbank. Wir verwenden dafür Toad *(siehe Abbildung 2)*.

In diesem Skript suchen wir alle Vorkommnisse von `BYTE` bei der Spaltendefinition und ersetzen sie durch `CHAR` *(siehe Abbildung 3)*. Außerdem empfiehlt es sich, nach parametrisierten `DATE`-Spezifikationen wie `DATE(8)` zu suchen, die in Oracle 11 manchmal verwendet wurden, aber in Oracle 19c nicht mehr erlaubt sind.

Bevor das Skript ausgeführt wird, müssen die bereits vorhandenen Tabellen mit der falschen Semantik gelöscht werden. Danach kann das erzeugte DDL-Skript ausgeführt werden, um die Tabellen mit der neuen `CHAR`-Semantik in der neuen Datenbank zu erstellen.

Nach erfolgreicher Ausführung des Skripts kann mit der Datenmigration begonnen werden. Im ersten Schritt dieses Prozesses sollten die Constraints und Trigger deaktiviert werden. Besonders wichtig dabei ist die Option `NOVALIDATE` für Constraints. Das beschleunigt den Import der Daten deutlich. Dies kann entweder mithilfe eines Tools wie zum Beispiel Toad *(siehe Abbildung 4)* oder durch das Generieren eines Skripts und Hinzufügen von `DISABLE NOVALIDATE` für jeden Constraint erreicht werden. Ab der Data Pump Bundle Patch-Version 19.27 (in anderen Quellen 19.14+) gibt es die Möglichkeit, diese Option in der Import .par-Datei mit **transform=constraint_novalidate:y** zu spezifizieren.

Nun können die Daten aus den exportierten Dump-Dateien in die angepassten Strukturen der neuen Datenbank importiert werden. Der entscheidende Parameter in `der .par`-Datei lautet **content=DATA ONLY** *(siehe Listing 1)*.

Während des Imports können weitere ORA-12899-Fehler auftreten *(siehe Abbildung 5)*. Die Fehlermeldung *„value too large for column … (actual: 4000, maximum: 4000)“* wirkt irreführend, hat aber den gleichen Grund: Die neue `CHAR`-Semantik benötigt mehr Platz. Wenn ein Text in der alten Datenbank bereits nahe am `VARCHAR2`-Maximum (4000 Byte) lag, passt er nicht mehr in die neue Struktur.

Die Lösung besteht entweder darin, die betroffenen Texte zu kürzen – in vielen Fällen handelt es sich dabei um Kommentare oder Beschreibungen, die nach Absprache mit den Entwicklern manchmal problemlos gekürzt werden können – oder den `MAX_STRING_SIZE`-Parameter in der neuen Datenbank auf `EXTENDED` zu setzen, um größere `VARCHAR2`-Größen zu ermöglichen.

Nach erfolgreichem Abschluss des Imports müssen die Constraints und Triggers wieder aktiviert (enable) werden. Auch hierbei ist es wichtig, die Option `NOVALIDATE` zu verwenden, da es sonst zu einer sehr langen Wartezeit kommt. Nach diesem Schritt ist die Migration abgeschlossen.

Hier ist noch einmal die Migrationsanleitung in Kurzform:

1. Inhalt der alten Datenbank mit Oracle Data Pump exportieren.
2. Strukturen (`CONTENT=METADATA_ONLY`) mit Oracle Data Pump in die neue Datenbank importieren.
3. DDL-Skript zum Erstellen der Tabellen mit der alten `BYTE`-Semantik generieren.
4. In diesem Skript `VARCHAR2(n BYTE)` durch `VARCHAR2(n CHAR)` ersetzen.
5. Die leeren Tabellen mit der falschen Semantik in der neuen Datenbank löschen.
6. Das Skript mit der angepassten Semantik ausführen.
7. Constraints und Triggers deaktivieren, die `NOVALIDATE`-Option für Constraints verwenden.
8. Daten (`CONTENT=DATA_ONLY`) mit Oracle Data Pump in die neue Datenbank importieren.
9. Die möglichen weiteren ORA-12899 *„value too large … (actual: 4000, maximum: 4000)“*-Fehler durch Kürzen der betroffenen Inhalte oder Setzen des `MAX_STRING_SIZE`-Parameters auf `EXTENDED` beheben.
10. Constraints und Trigger aktivieren, die NOVALIDATE-Option für Constraints verwenden.

Wie am Anfang dieses Artikels erwähnt, haben wir eine eigenartige Situation, die uns nicht erlaubt, ein automatisches Upgrade auf Oracle 19c durchzuführen. Wir hatten keine andere Wahl, als nach Alternativen und Workarounds zu suchen. Daraus ist dieser Algorithmus entstanden. Wenn er jemandem auf dem gleichen Weg hilft, können wir zufrieden sein.

Über die Autorin

Lena Rudenko wurde in der Ukraine geboren. Im Jahr 2014 hat sie ihren Abschluss in Master Informatik in Augsburg gemacht. Im Jahr 2020 folgte Promotion am Lehrstuhl für Datenbanken und Informationssysteme der gleichen Universität. Seit 2021 ist sie bei Prodato tätig, wo sie sich seit drei Jahren leidenschaftlich mit der Administration der Oracle-Datenbanken beschäftigt.

Lena Rudenko
lena.rudenko@prodato.de

APEX connect

18. - 20. Mai 2026

+

DOAG 2026 Datenbank

mit Cloud Infrastructure

18. - 19. Mai 2026

Heide Park Soltau

Multitenant und Data Guard – was kann da schon schiefgehen?

Markus Flechtner, Ordix

Oracle Data Guard – die bewährte Technologie für Hochverfügbarkeit und Oracle Multitenant, die „neue“ Datenbank-Architektur seit Oracle 12c. Eine starke Kombination für Oracle-HA-Systeme. Allerdings gibt es da einige kleinere Fallstricke und wir werden sehen, welche das sind und wie man die daraus resultierenden Probleme in der Version 19c der Datenbank vermeiden kann.

Was ist Oracle Multitenant?

Die Oracle-Multitenant-Architektur *(siehe Abbildung 1)*, auch Container-Datenbank-Architektur genannt, ist die mit Oracle 12c eingeführte „neue" Datenbank-Architektur, die eine Container-Datenbank (CDB) als übergeordnetes Konstrukt bereitstellt. Die grundsätzliche Oracle-Architektur mit den bekannten Hintergrund-Prozessen und der SGA bleibt dabei unverändert. Innerhalb einer CDB können eine oder mehrere Pluggable Datenbanken (PDBs) betrieben werden, in denen die eigentlichen Anwendungsdaten liegen. Ohne die kostenpflichtige Multitenant-Option lassen sich bis zu drei PDBs pro CDB betreiben. Die Daten jeder PDB bleiben logisch voneinander getrennt; Control- und Redolog-Dateien werden hingegen gemeinsam genutzt. Die Architektur bietet zahlreiche Vorteile: Sie ermöglicht eine effiziente Konsolidierung von Datenbanken, eine einfache Bereitstellung neuer Datenbanken, eine bessere Nutzung von Ressourcen und vereinfacht die Administration (Stichwort: „Manage many as one").

Seit Oracle 21c ist die Multitenant-Architektur die einzige verfügbare Datenbank-Architektur – die klassische Non-CDB wird nicht mehr unterstützt.

Was ist Oracle Data Guard?

Oracle Data Guard als Hochverfügbarkeits- und Disaster-Recovery-Lösung für die Oracle Datenbank gibt es seit Oracle 8i und damit schon deutlich länger als Multitenant.

Data Guard ermöglicht die Synchronisation von Änderungen aus einer Primärdatenbank in eine oder mehrere Standby-Datenbanken, um Datenverlust zu vermeiden und die Ausfallsicherheit zu erhöhen. Dabei können je nach Anwendungsfall und Anforderungen verschiedene Typen von Standby-Datenbanken eingesetzt werden: physische, logische oder Snapshot-Standbys. Ein sogenannter Observer ermöglicht optional den automatischen Failover zur Standby-Datenbank im Fehlerfall; ein Switchover erlaubt einen Rollentausch der beiden Datenbanken. Oracle Data Guard unterstützt Modi wie Maximum Performance, Maximum Availability und Maximum Protection, je nach Anforderung an die Verfügbarkeit der Daten und an die Performance. Mit Active Data Guard kann eine Standby-Datenbank im Read-Only-Modus geöffnet werden, wodurch sich Lesezugriffe effizient auf mehrere Systeme verteilen lassen und die Last der Primärdatenbank reduziert wird *(siehe Abbildung 2)*.

```
SQL> create pluggable database PDBDEMO2 admin user pdbadmin
  2 identified by manager roles=(DBA)
  3 file_name_convert=('pdbseed','PDBDEMO2');
Pluggable database created.
```

Listing 1: Anlegen einer PDB

Wie spielen Multitenant und Data Guard zusammen?

Die Kombination aus Multitenant und Data Guard funktioniert *(siehe Abbildung 3)*. Da Data Guard auf dem Transfer der Redolog-Informationen zur Standyby-Seite beruht und da die Redolog-Dateien bei Container-Datenbanken eine gemeinsame Komponente sind, wird Data Guard in Oracle 19c immer für die komplette CDB eingerichtet. Das Feature "PDB-Level-Data Guard" wurde erst mit Oracle 21c eingeführt. Man kann zwar beim Anlegen einer PDB festlegen, dass sie nicht auf die Standby-Seite übertragen wird, in dem man den Parameter „STANDBYS=NONE" angibt, aber das ist im Normalfall nicht sinnvoll. PDBs beziehungsweise Anwendungen, die die Hochverfügbarkeit von Oracle Data Guard nicht benötigen, sollten nicht in einer Data Guard-CDB abgelegt werden. Allerdings werden wir später sehen, dass uns das Feature „STANDBYS=NONE" wertvolle Dienste leisten kann.

Was funktioniert problemlos?

Die Fälle, für die Data Guard gedacht ist, funktionieren auch mit Container-Datenbanken problemlos: Switchover (Rollentausch der beteiligten Datenbanken) und Failover (Aktivierung der Standby-Datenbank im Fehlerfall). Diese betreffen aber immer die komplette CDB, das heißt alle PDBs. Auch das Neu-Anlegen einer PDB als Kopie der Seed-Datenbank PDB$SEED oder das Kopieren einer Read-Only-PDB funktionieren. In diesen Fällen sind die benötigten Dateien statisch und auf beiden Seiten (Primär- und Standby-Datenbank) bereits vorhanden.

Ein Beispiel für das Anlegen einer neuen PDB als Kopie der PDB$SEED ist in *Listing 1 und 2* zu sehen.

Was kann schiefgehen?

Es gibt einige Varianten der Bereitstellung von PDBs oder beim Kopieren von PDBs, die besondere Aufmerksamkeit erfordern. Diese sind:

- Kopieren einer R/W-PDB (lokal und remote), inkl. „Refreshable PDB" und „PDB Relocate"
- Plug-In einer PDB
- Remote-Cloning einer R/O-PDB
- Tempfiles auf der Standby-Seite (nach Switchover/Failover) Punkt 3

Was ist das Problem in diesen Fällen?

Beim Kopieren einer PDB, die Read-Write geöffnet ist, kommt es zu einem Recovery-Konflikt: nach dem Kopieren der Dateien werden auf der Primärseite die Änderungen, die seit Beginn des Kopierens angefallen sind, nachgefahren („PDB level redo apply"). Auf der Standby-Seite kollidiert dieses Recovery mit dem gleichzeitig laufenden Redo-Apply des Data Guard. Bis Oracle 12.2 kam es zu dem Fehler „ORA-1153 an incompatible media recovery is active" und der MRP-Prozess auf der Standby-Seite brach ab – sprich: Primär- und Standby-Datenbank laufen auseinander. Seit Oracle 18c erkennt Oracle dieses Problem und kopiert die Dateien nicht auf die Standby-Seite (alert.log: „Tablespace-SYSTEM during PDB create skipped since source is in r/w mode or this is a refresh clone"). Das Wichtige ist aber: Data Guard läuft

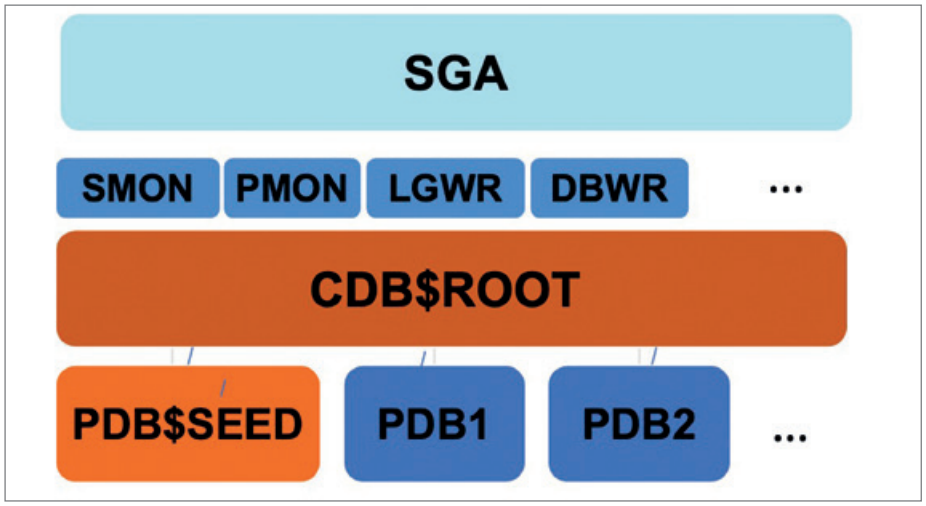


Abbildung 1: Oracle Multitenant (Quelle: Markus Flechtner)

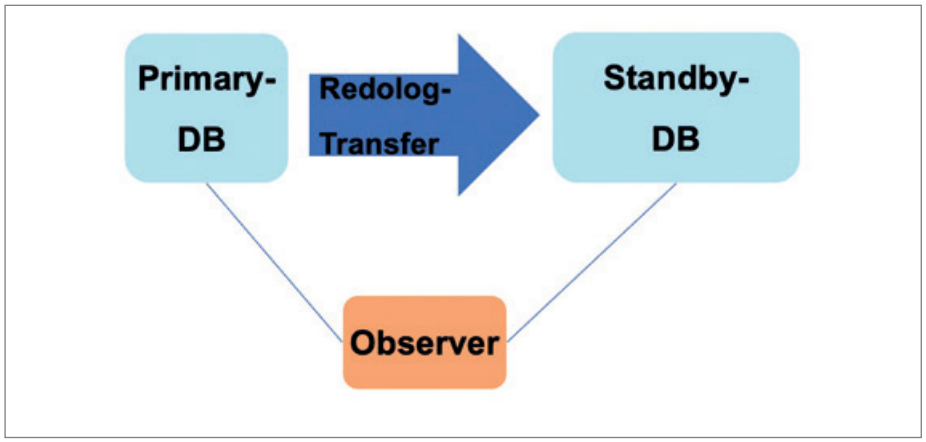


Abbildung 2: Oracle Data Guard (schematisch) (Quelle: Markus Flechtner)

weiter, die Hochverfügbarkeit bleibt gesichert.

Beim Einhängen einer PDB („Plug-In") „weiß" die Standby-Seite nicht, wo sie die Dateien finden soll – es kommt zum Fehler „Recovery was unable to create the file" und das Redo Apply bricht mit der Meldung „Background Media Recovery process shutdown" ab. Das heißt, dass auch in diesem Fall Primär- und Standby-Seite nicht mehr konsistent sind.

Wie können (neue) Parameter helfen?

Mit Oracle 18c wurden neue Parameter eingeführt, die zwei der oben geschilderten Probleme lösen, nämlich die Parameter STANDBY_PDB_SOURCE_FILE_DBLINK und STANDBY_PDB_SOURCE_FILE_DIRECTORY.

Der Parameter STANDBY_PDB_SOURCE_FILE_DBLINK hilft beim Kopieren einer Remote-PDB, die read-only geöffnet ist *(siehe Abbildung 4)*. Dieser Parameter gibt den Namen des Datenbank-Links an, über den die Primär-Datenbank die Dateien der Remote-PDB kopiert. Damit die Standby-Datenbank auf die – natürlich in der Datenbank gespeicherten – Informationen zu diesem Datenbank-Link zugreifen kann, muss der Root-Container CDB$ROOT der Standby-Datenbank „read-only" geöffnet sein. Dies ist aber erlaubt, erst wenn man eine PDB read-only öffnen möchte, und es ist eine Active-Data-Guard-Lizenz erforderlich.

Der Ablauf beim Kopieren einer Remote-R/O-PDB ist dann wie in *Listing 3* dargestellt.

Der Benutzer in der Quell-Datenbank muss die Rechte CREATE PLUGGABLE DATABASE sowie SYSOPER und CREATE SESSION (CONTAINER=ALL) haben.

Anschließend kann man den Parameter STANDBY_PDB_SOURCE_FILE_DBLINK auf der Standby-Seite zurücksetzen („ALTER SYSTEM RESET STANDBY_PDB_SOURCE_FILE_DBLINK;")

Die Kontrolle auf der Standby-Seite zeigt, dass die PDB dort erfolgreich angelegt wurde und dass das Redo-Apply weiterläuft *(siehe Listing 6)*.

Der Parameter STANDBY_PDB_SOURCE_FILE_DIRECTORY hilft beim Einhängen einer PDB, in dem er auf das physische Verzeichnis zeigt, in dem die Dateien der PDB liegen. In den meisten Fällen wird dies ein NFS-Mount sein, den die beteiligten Datenbank-Server gemeinsam nutzen *(siehe Abbildung 5)*.

Zuerst muss die PDB aus der betreffenden CDB ausgehängt werden und die Dateien müssen auf den NFS-Share kopiert werden *(siehe Listing 7)*.

Danach setzen wir den Parameter STANDBY_PDB_SOURCE_FILE_DIRECTORY auf der Standby-Seite. Da in diesem Fall nicht auf Informationen in der Datenbank zurückgegriffen werden muss, muss die Standby-CDB (CDB$ROOT) nicht read-only geöffnet sein *(siehe Listing 8)*.

Danach kann die PDB auf der Primär-Seite eingehängt werden *(siehe Listing 9)*.

Dieser Befehl ist vielleicht etwas erklärungsbedürftig:

- Die Metadaten der einzuhängenden PDB liegen in einer XML-Datei, dem sogenannten PDB-Manifest, das unter */nfs/ora2know-share/plugtest.xml* abgelegt ist.
- In dieser XML-Datei sind auch die Dateinamen der PDB aufgeführt. Diese sind allerdings nicht mehr in ihrem Original-Ort zu finden, sondern auch auf dem NFS-Share. Diese Änderung geben wir über den Parameter SOURCE_FILE_NAME_CONVERT mit.
- Die Dateien der PDB sollen natürlich nicht auf dem NFS-Share bleiben, sondern in ein lokales Verzeichnis auf dem Datenbank-Server kopiert werden. Dazu dienen die Optionen „COPY" und „FILE_NAME_CONVERT", die eine weitere Änderung der Dateinamen festlegt.

Nach dem Einhängen der PDB kann der Parameter STANDBY_PDB_SOURCE_FILE_DIRECTORY zurückgesetzt werden.

Ein Blick in die alert.log-Datei auf der Standby-Seite zeigt, dass die Dateien auch dort erfolgreich eingehängt wer-

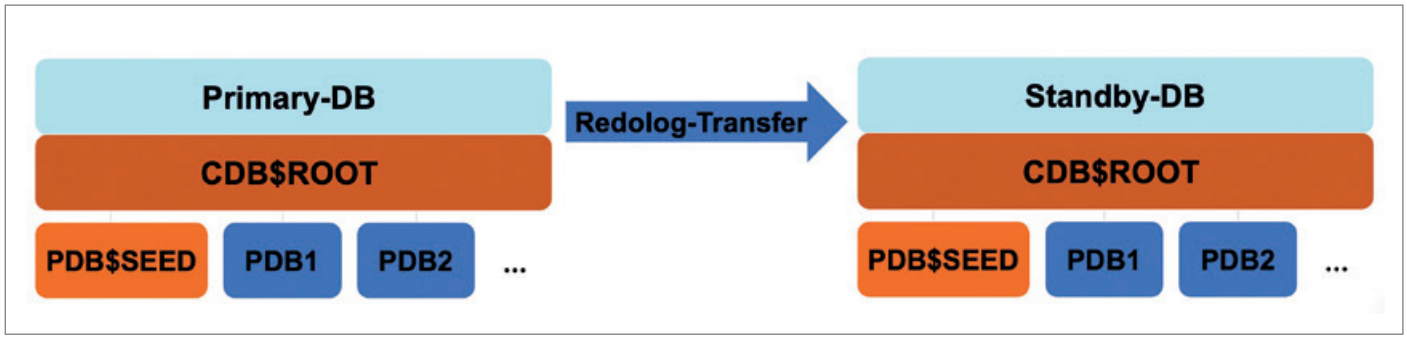


Abbildung 3: Multitenant-Datenbank und Oracle Data Guard (schematisch) (Quelle: Markus Flechtner)

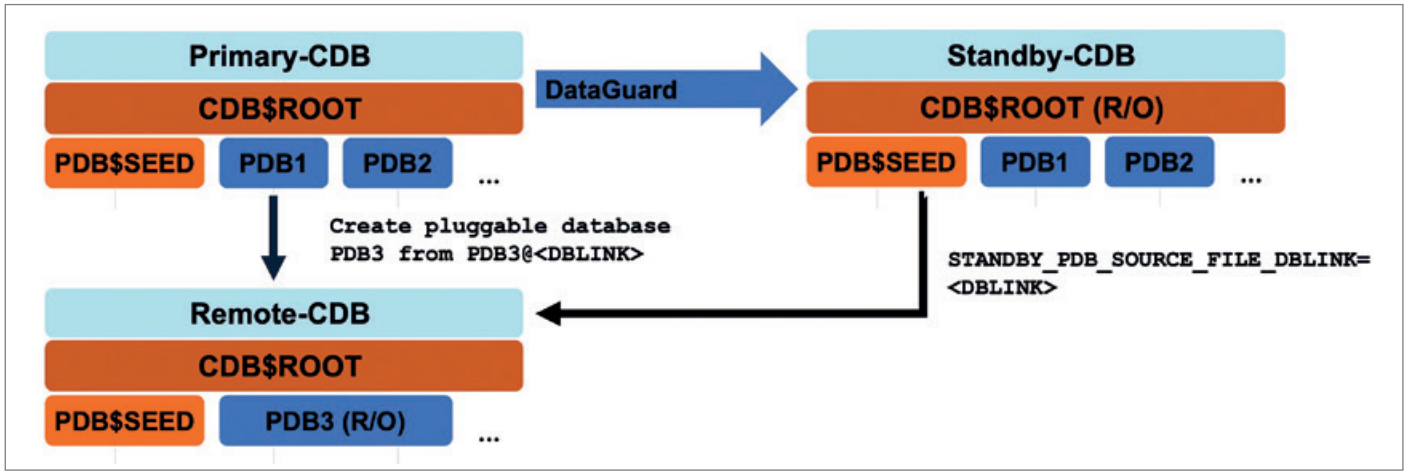


Abbildung 4: Ablauf des Remote-Copy mit STANDBY_PDB_SOURCE_FILE_DBLINK (Quelle: Markus Flechtner)

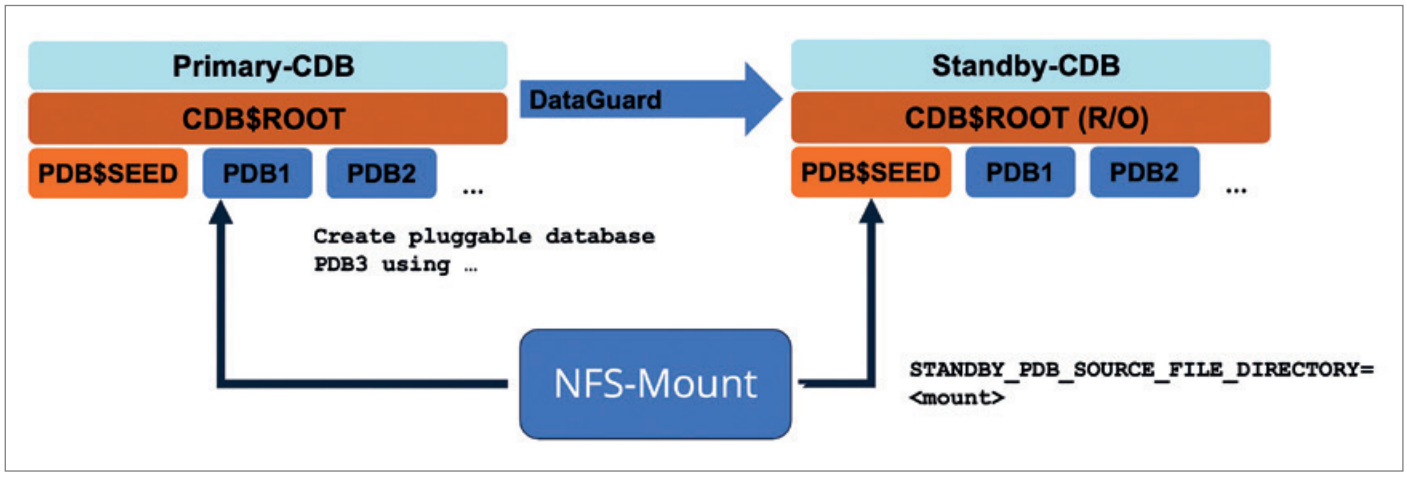


Abbildung 5: Einhängen einer PDB mit STANDBY_PDB_SOURCE_FILE_DIRECTORY (Quelle: Markus Flechtner)

den konnten und dass das Redo-Apply weiterläuft *(siehe Listing 10)*.

Verwendung von „STANDBYS=NONE“

Leider lösen die beiden Parameter nicht alle Problemfälle, so dass wir für die verbleibenden Fälle einen etwas aufwändigeren Weg nehmen müssen, nämlich die Verwendung der Option „STANDBYS=NONE“. Seit Oracle 18c ist diese Option allerdings nicht mehr erforderlich, denn Oracle macht das, wie oben ausgeführt, automatisch. Sie kann aber weiterhin zur Verdeutlichung, dass die PDB nicht auf der Standby-Seite angelegt werden soll, angegeben werden.

In diesen Fällen müssen wir mit diesem Verfahren arbeiten:

- Kopieren einer lokalen PDB, die zum Schreiben (read/write) geöffnet ist
- Kopieren einer Remote-PDB, die read-write geöffnet ist
- „Refreshable Clone“
- Relocate einer PDB

Ein Beispiel dafür findet sich in *Listing 11*. Zu beachten ist, dass „STANDBYS=NONE“ seit Oracle 18c implizit gesetzt wird und hier nur zur Verdeutlichung dient.

Auf der Standby-Seite sieht man, dass die Dateien nicht kopiert werden *(siehe Listing 12)*.

```
SQL> create database link remcdb connect to C##ORA2KNOW
 2   identified by <geheim>
 3   using 'server3.ora2know.internal:1521/REMCDB';
Database link created.
```

Listing 3: Anlegen des DB-Links auf der Primär-Seite

```
SQL> alter system set STANDBY_PDB_SOURCE_FILE_DBLINK=remcdb scope=memory;
System altered.
```

Listing 4: Setzen des Parameters STANDBY_PDB_SOURCE_FILE_DBLINK auf der Standby-Seite

```
SQL> create pluggable database REMTEST from REMTEST@REMCDB
  2  file_name_convert=('/u01/oradata/REMCDB/REMTEST','/u01/oradata/DGDEMO/REMTEST');
Pluggable database created.
```

Listing 5: Kopieren der Remote-PDB

```
2025-11-16T09:41:03.153161+01:00
ALTER SYSTEM SET standby_pdb_source_file_dblink='REMCDB' SCOPE=BOTH;
2025-11-16T09:44:59.374446+01:00
Recovery created pluggable database REMTEST
Recovery attempting to copy datafiles for pdb-REMTEST from source pdb-REMTEST at dblink-REMCDB
2025-11-16T09:45:02.076192+01:00
Recovery copied files for tablespace SYSTEM
Recovery successfully copied file /u01/oradata/DGDEMO/REMTEST/system01.dbf from /u01/oradata/REMCDB/
REMTEST/system01.dbf
[…]
REMTEST(6):Successfully added datafile 24 to media recovery
REMTEST(6):Datafile #24: '/u01/oradata/DGDEMO/REMTEST/users01.dbf'
```

Listing 6: alert.log der Standby-Seite nach dem Kopieren der Remote-PDB

```
SQL> alter pluggable database PLUGTEST unplug into '/nfs/ora2know-share/plugtest.xml';
Pluggable database altered.
SQL> drop pluggable database PLUGTEST keep datafiles;
Pluggable database dropped.
´SQL> !cp /u01/oradata/REMCDB/PLUGTEST/* /nfs/ora2know-share/.
```

Listing 7: Aushängen der PDB („Plug-Out“)

```
SQL> alter system set STANDBY_PDB_SOURCE_FILE_DIRECTORY='/nfs/ora2know-share/'
 2   scope=memory;
System altered.
```

Listing 8: Setzen des Parameters STANDBY_PDB_SOURCE_FILE_DIRECTORY

```
SQL> create pluggable database PLUGTEST
  2  using '/nfs/ora2know-share/plugtest.xml'
  3  source_file_name_convert=
  4  ('/u01/oradata/REMCDB/PLUGTEST/','/nfs/ora2know-share/')
  5  copy file_name_convert=
  6 ('/nfs/ora2know-share/','/u01/oradata/DGDEMO/PLUGTEST/');
Pluggable database created.
```

Listing 9: Einhängen der PDB

```
2025-11-16T10:15:35.327043+01:00
Recovery created pluggable database PLUGTEST
Recovery attempting to copy datafiles for pdb-PLUGTEST from           source dir-/nfs/ora2know-share/
Recovery copied files for tablespace SYSTEM
Recovery successfully copied file /u01/oradata/DGDEMO/PLUGTEST/system01.dbf from /nfs/ora2know-share/sys-
tem01.dbf
[…]
PLUGTEST(7):Successfully added datafile 28 to media recovery
PLUGTEST(7):Datafile #28: '/u01/oradata/DGDEMO/PLUGTEST/users01.dbf'
```

Listing 10: alert.log auf der Standby-Seite nach dem Einhängen der PDB

```
SQL> create pluggable database RWDEMO from PDBDEMO
 2   file_name_convert=('PDBDEMO','RWDEMO') standbys=none;
Pluggable database created.
```

Listing 11: Kopieren einer lokalen R/W-PDB

```
2025-11-16T10:24:15.155916+01:00
Recovery created pluggable database RWDEMO
RWDEMO(8):Tablespace-SYSTEM during PDB create skipped since source is in              r/w mode or this is
a refresh clone
RWDEMO(8):File #29 added to control file as 'UNNAMED00029'. Originally created as:
RWDEMO(8):'/u01/oradata/DGDEMO_S1/RWDEMO/system01.dbf'
RWDEMO(8):because the pluggable database was created with nostandby
RWDEMO(8):or the tablespace belonging to the pluggable database is
RWDEMO(8):offline.
```

Listing 12: alert.log-Datei der Standby-DB nach dem Kopieren einer lokalen R/W-PDB

```
RMAN> connect target /
RMAN run{
set newname for datafile 29 to '/u01/oradata/DGDEMO/RWDEMO/system01.dbf';
set newname for datafile 30 to '/u01/oradata/DGDEMO/RWDEMO/sysaux01.dbf';
set newname for datafile 31 to '/u01/oradata/DGDEMO/RWDEMO/undotbs01.dbf';
[…]
restore pluggable database RWDEMO from service 'DGDEMO_S1';
}
```

Listing 13: Kopieren der PDB-Datei mit RMAN

```
DGMGRL> edit database DGDEMO_s2 set state='apply-off';
```

Listing 14: Deaktivieren des Redo-Apply

```
RMAN> connect target /
RMAN> switch pluggable database RWDEMO to copy;
```

Listing 15: RMAN switch-to-copy

```
SQL> shutdown immediate;
SQL> startup nomount;
SQL> alter database mount standby database;

SQL> alter session set container=RWDEMO;
SQL> alter pluggable database enable recovery;

SQL> alter database datafile 29 online;
SQL> alter database datafile 30 online;
SQL> alter database datafile 31 online;
```

*Listing 16: Abschließende Schritte in SQL*Plus*

```
DGMGRL> edit database DGDEMO_s2 set state='apply-on';
```

Listing 17: Aktivierung des Redo-Apply

Danach nutzen wir RMAN um die Dateien der PDB zu übertragen *(siehe Listing 13)*.

Würden wir Oracle Managed Files (OMF) nutzen, dann wäre es einfacher, denn dann könnten wir den Befehl *„set newname for pluggable database <PDB> to new“* nutzen.

Danach müssen wir das Einspielen der Redolog-Informationen („Redo Apply“) vorübergehend deaktivieren *(siehe Listing 14)*. Ab jetzt sind Primär- und Standby-Datenbank also nicht mehr synchron!

Danach können wir die kopierten Dateien in der Standby-CDB „aktivieren“ *(siehe Listing 15)*.

Jetzt müssen wir die Instanz der Standby-DB durchstarten, für die neue PDB das Redo-Apply aktivieren und dann die Dateien der PDB „online“ setzen *(siehe Listing 16)*.

Für größere PDBs ist es empfehlenswert, sich für das Online-Setzen der Dateien vorab ein Skript zu generieren.

Zum Abschluss können wir dann das Redo-Apply wieder aktivieren *(siehe Listing 17)*.

Der Media-Recovery-Prozess (MRP) spielt jetzt die seit Schritt 3 angefallenen Änderungen in die Datenbank ein und nach kurzer Zeit sind Primär- und Standby-Datenbank wieder synchron.

Der letzte Problemfall, der die Arbeit des DBAs erfordert, sind die Tempfiles der Standby-Datenbank: sie werden nicht automatisch angelegt, sondern müssen nach einem Failover oder Switchover mit dem Befehl „ALTER DATABASE ADD TEMPFILE <nummer>;“ angelegt werden.

Insgesamt gilt, dass Oracle Data Guard auch mit Container-Datenbanken gut funktioniert. Beim Einhängen von PDBs und beim Kopieren von Read-Write-PDBs muss man zwar einige Besonderheiten beachten, aber Oracle wird hier von Version zu Version besser. Und im Vergleich zu der hier behandelten Version 19c gibt es in 21c, 23ai und 26ai schon weitere hilfreiche Verbesserungen.

Über den Autor

Markus Flechtner ist Principal Consultant bei der Ordix AG. Seine Schwerpunkte liegen in den Bereichen Multitenant, Hochverfügbarkeit sowie Upgrades und Migrationen. Weiterhin gibt er Seminare zu Oracle und PostgreSQL. Er ist Oracle ACE Pro und Mitgründer der Oracle Community ora2know.

Markus Flechtner
mfl@ordix.de

SQL-Tuning für PostgreSQL

Wolfgang Wilhelm, Opitz Consulting Deutschland

Die Datenbank ist so langsam… Welcher DBA kennt das nicht? Neben Standardlösungen wie einem neuen Index oder schnellerer Hardware kann das SQL getunt werden. Dazu reicht oftmals ein Blick in den Output von Explain, um herauszufinden, woran es hakt.

Tuning ist einfach: Parameter „make_postgres_faster=true“ setzen </halluzination>

So einfach ist es nicht, aber es ist auch keine Raketenwissenschaft. Vieles ist bei PostgreSQL ähnlich dem, was Oracle kennt. Es ist wie dort auch keine einmalige Aktion, sondern eine Teamaufgabe von User, Dev, DBA und anderen Stakeholdern.

Getunt wird nur, wenn ein paar Voraussetzungen gegeben sind:

- DB-Statistiken sind aktuell
- Kenntnisse über Daten sind vorhanden
- Kenntnisse über Metadaten, z. B. Indices, sind vorhanden
- Das zu tunende SQL ist bekannt

Es wird gar nicht versucht zu tunen, wenn es nicht möglich ist, beispielsweise bei:

- SELECT * FROM tab1;
- SELECT * FROM tab1 CROSS TAB tab2;
- Laufzeit besteht hauptsächlich aus Datentransfer

Ein paar Begriffe

Ein Equi-Join ist ein Join mit einer oder mehreren Gleichheitsbedingungen. Equi-Joins arbeiten mit allen Techniken, die die Datenbank erlaubt.

Der Non-Equi-Join ist das Gegenteil und kommt zur Anwendung, wenn im

SQL beispielsweise Größer- oder Kleinerzeichen vorkommen oder Jokerzeichen im SQL-LIKE.

Ein Anti-Join ist eine Negativauswahl beim Joinen, also quasi ein umgekehrter Equi-Join. Sie bieten sich an, wenn die Menge an Ausprägungen bei der Negativauswahl deutlich geringer ist als beim Equi-Join. Der Nachteil ist, dass bis zum ersten Treffer immer alle Ausprägungen abgearbeitet werden müssen.

Der Bitmap-Index- oder Bitmap-Heap-Scan wird dann verwendet, wenn die Datenbank aus definierten Bereichen mögliche Treffer heraussuchen muss. Das ist leicht parallelisierbar.

Filter sind Einschränkungen bei der Suche, etwa in Form von Where- oder Except-Klauseln, von Limits bei einem Group by having oder auch bei einem Inner Join.

Full Scans, die auch Sequential Scans oder kurz Seq Scans genannt werden, durchsuchen meist viele Zeilen einer Tabelle und gleichen alle Daten mit dem Filterkriterium ab. Der Full Scan wird, wie auch bei Oracle, immer dann benutzt, wenn nach einem Nicht-Index-Attribut gescannt werden soll, wenn gar keine Einschränkung gemacht wird (wie bei einem kartesischem Produkt), und wenn es billiger ist als die Nutzung von Index und Heap. Letzteres ist oftmals bei sehr kleinen Tabellen der Fall.

Das Gegenstück dazu auf Indexebene wird Index-Scan genannt. Das kommt beispielsweise bei einem SQL-Like auf das erste Feld eines Indexes zustande, wenn die Jokerzeichen nicht am Anfang des Suchbegriffs stehen.

Hash-Joins, die nur bei Equi-Joins nutzbar sind, berechnen Hashwerte für beide zu joinenden Tabellen und vergleichen diese dann. Sie sind speicherschonend und werden daher benutzt, wenn viele Treffer erwartet werden.

Ein (Sort-) Merge-Join wird bei zwei Tabellen benutzt, die nach gleichem Kriterium vorsortiert sind. Das ist beispielsweise bei einem Index-Scan teilweise bereits der Fall. Dieser wird genutzt, wenn die Treffermengen recht groß sind und nicht mehr in das work_mem passen.

Der Nested-Loop-Join ist die allgemeinste Technik und die Einzige, die bei

```
SELECT UUID, L.MENGENKZ, T.TEXT
FROM KOMPONENTE K
JOIN LAGER L USING (UUID)
JOIN TEXTE T USING (UUID)
WHERE K.CLASS_TYPE='LAGER'
```

Listing 1: Das SQL für den Explainplan

```
                                   QUERY PLAN
----------------------------------------------------------------------------------------------------
 Nested Loop  (cost=11.88..2430.57 rows=1 width=57) (actual time=0.288..3.329 rows=281 loops=1)
   ->  Hash Join  (cost=11.32..34.76 rows=281 width=94) (actual time=0.258..0.569 rows=281 loops=1)
         Hash Cond: ((l.uuid)::text = (t.uuid)::text)
         ->  Seq Scan on lauflager l  (cost=0.00..22.30 rows=430 width=37) (actual time=0.022..0.104 rows=430 loops=1)
         ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.221..0.223 rows=281 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 33kB
               ->  Seq Scan on lauflager_typ t  (cost=0.00..7.81 rows=281 width=57) (actual time=0.013..0.101 rows=281 loops=1)
   ->  Index Scan using c_komponente_pk on komponente k  (cost=0.55..8.53 rows=1 width=37) (actual time=0.009..0.009 rows=1 loops=281)
         Index Cond: ((uuid)::text = (l.uuid)::text)
         Filter: ((class_type)::text = 'Lauflager'::text)
 Planning Time: 0.878 ms
 Execution Time: 3.414 ms
```

Abbildung 1: Das Ergebnis des Explain (Quelle: Wolfgang Wilhelm)

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33KB
              -> Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 2: Zeile 1 des Explains (Quelle: Wolfgang Wilhelm)

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33KB
              -> Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 3: Zeile 2 des Explains (Quelle: Wolfgang Wilhelm)

einem Non-Equi-Join neben einem Full-Scan geeignet ist. Dabei wird zuerst eine Tabelle durchgegangen, Treffer werden gesucht und diese anschließend mit Informationen aus der anderen Tabelle des Joins verglichen. Daraus ergibt sich dann die Treffermenge. Der Equi-Join eignet sich nicht nur für das gerade beschriebene Szenario, sondern auch für sehr kleine Datenmengen.

Ein Semi-Join ist ein Join, bei dem nur aus einer der beiden Tabellen Daten gelesen werden. Das kommt zum Beispiel dann vor, wenn eine In-Klausel mit Werten aus einem Select einer anderen Tabelle abgefragt wird.

Nachdem jetzt die Begriffe klar sein sollten, schauen wir als nächstes auf den Nicht-SQL-Standard-Befehl „Explain“.

Explain me explain

„Explain“ vor ein DML-Kommando geschrieben liefert Informationen über den Ausführungsplan, den die Datenbank für das SQL verwenden will. Dabei gibt es zwei Varianten, nämlich einmal die textuelle Erstellung des Ausführungsplans und die zweite Variante, dem explain analyze, die das Kommando tatsächlich ausführt und dazu dann Ausführungszeiten sowie tatsächliche Datenmengen liefert. In den meisten Fällen ist die zweite Variante wegen der zusätzlich gelieferten Aussagen die Vorzuziehende. Die erste sollte nur verwendet werden, wenn geprüft wird, ob ein Index verwendet wird, aus welchen Partitionen gelesen wird, wenn die Ausführung sehr lange dauert, oder für die reinen Datenschätzungen der PostgreSQL-Planners.

Als Beispiel für einen Explainplan nehmen wir die Tabelle Komponente mit 1,9 Millionen Sätzen, die über eine UUID auf zwei abhängige Tabellen mit Datensatzzahlen jeweils unter 500 Sätzen pro Tabelle zugreift *(siehe Listing 1)*.

Vor das SQL wird dann ein „explain analyse“ geschrieben und dann kommt Folgendes als Ergebnis heraus *(siehe Abbildung 1)*.

Zu lesen ist nun der Explain von „innen“, das sind die am meisten eingerückten Zeilen, nach „außen“, also bis zur obersten Zeile. Einiges kann direkt gelesen werden, etwa die Tabellennamen mit Alias und manche der verwendeten Attribute. In *Abbildung 2* finden sich auch wieder die Begriffe, die im vorderen Teil des Artikels angesprochen wurden. Aus der innersten Zeile, dem Seq Scan auf die Texts-Tabelle, deren Zwischenergebnis 33 kB Platz braucht, folgt das Hashing und der Join mit der Tabelle Lager. Die Datenbank dreht also die Reihenfolge der Joins im SQL um. Dann werden mittels eines Index-Scans über den PK der Tabelle Komponente passende Werte gesucht und über einen Nested Loop herausgesucht, ob die passende Information, nämlich das Nicht-Index-Feld Class_type, dem Zwischenergebnis des Joins aus den Tabellen Texts und Lager zugeordnet werden kann.

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33kB
              ->  Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 4: Zeile 3 des Explains (Quelle: Wolfgang Wilhelm)

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33kB
              ->  Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 5: Zeile 4 des Explains (Quelle: Wolfgang Wilhelm)

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33kB
              ->  Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 6: Zeile 5 bis 7 des Explains (Quelle: Wolfgang Wilhelm)

In allen Zeilen außer den Zeiten und der Anzahl der ausgegebenen Zeilen am Ende des Explains steht eine feste Reihenfolge von der Information über das, was dort geplant wurde und dazu gehörender, zusätzlicher Information. Der geklammerte Ausdruck „(cost…)" liefert immer die Kosten in relativen Einheiten. Diese ergeben sich aus den Einstellungen aus den cost-Parametern in der postgresql.conf. Die Zahlen dahinter sind die Kosten für das Ermitteln des ersten passenden Datensatzes, die zweite Zahl des letzten Satzes. „width" ist die Anzahl der Zeichen. Da wir einen Explain Analyze durchgeführt haben, erfolgt die verbrauchte Zeit in Millisekunden wieder für die erste und letzte Zeile. Hinter „rows" steht die Anzahl der ermittelten Zeilen. Das Ergebnis des obigen SQLs sind also 281 Zeilen *(siehe Abbildung 2)*. Da es sich um einen Explain Analyze handelt, der hier betrachtet wird, sind das die echten Zeilen. Ein Explain hätte hier nur eine Schätzung der Datenbank geliefert. Wenn diese Schätzung von der tatsächlichen Datenlieferung deutlich abweicht, dann ist Handlungsgebot geboten. Oftmals reicht es, die Statistiken (neu) zu ziehen. Das geschieht mittels eines Vacuums.

Die nächste Zeile zeigt das Ergebnis des Hash Joins zwischen den Tabellen Lager und Texts. Auch hier sind wieder relative Kosten und tatsächlich Zeiten angegeben *(siehe Abbildung 3)*.

Die nächste Zeile zeigt an, mit welchen Bedingungen der Hash Join durchgeführt wurde *(siehe Abbildung 4)*. Es steht der Join mit den verwendeten Aliasen und Attributnamen da. Für Oracle-Kenner wird die Syntax mit den doppelten Doppelpunkten seltsam aussehen. Es handelt sich dabei um eine Typumwandlung, ein „coalesce", das PostgreSQL auch kennt. Das Format „text" ist auf Oracle ein CLOB oder eher ein bei Oracle wegen der Länge nichtexistierendes VARCHAR2(1.073.741.824), also ein GB. Eine echte Umwandlung findet allerdings nicht statt, sondern ist für VARCHAR-Typen rein virtuell.

Der Explain informiert in der nächsten Zeile, dass die komplette Tabelle Lager mit allen 430 Datensätzen sequenziell eingelesen wird. Der Durchschnittsplatzbedarf sind 45 Zeichen. Der Aufwand beträgt bis zum Lesen des letzten Satzes 0,6 ms *(siehe Abbildung 5)*. Dabei ist zu beachten, dass die Daten in drei verschiedenen Bereichen liegen können, nämlich dem shared memory, also dem Teil, den die Datenbank selbst verwaltet, im Betriebssystemcache oder auf der Disk. Je weiter die Daten von den Zugriffen der Datenbank entfernt sind, desto länger braucht das Einlesen. Das führt dann dazu, dass die zweite Verarbeitung der gleichen Daten schneller geht, wenn sie nun im shared memory liegen und vorher von Disk gelesen werden mussten.

Nachfolgende drei Zeilen sind am weitesten eingerückt und zeigen den ersten Arbeitsgang an. Es werden alle 281 Zeilen der Tabelle „Texts" per Seq Scan eingelesen. Dafür braucht die Datenbank gerade mal 0,28 ms *(siehe Abbildung 6)*. Die Daten werden in einen eigenen Speicherbereich des shared memory geschrieben und das verbraucht 33 kB. Für die Zeilen wird ein Hash ermittelt. Das zusammen mit den 0,28 ms ist in 0,4 ms erledigt. Die Hashermittlung für diese Zeilen geht sehr schnell, was sich aus dem Unterschied zwischen der ersten Zeile in 0,405 ms und der letzten Zeile in 0,406 ms ergibt.

Mit den folgenden beiden Zeilen beschreibt der Explain, was dazu führt, dass das bisherige Zwischenergebnis mit der größten im Join verwendeten Tabelle ergänzt wird *(siehe Abbildung 7)*. Wie zu lesen ist, verwendet die Datenbank einen Index Scan über das Constraint „komponente_pkey", dem einfach zu erkennenden Primary Key der Tabelle Komponente mit dem Alias „k". Das Ganze braucht 0,55 relative Einheiten für den ersten Datensatz und 8,53 Einheiten für den letzten. Das erscheint viel, die echten Zeiten relativieren dies und das liegt daran, dass die Tabelle bereits im shared memory lag. Daher ist der Aufwand auch minimal. Der Join holt sich aus der Datenmenge genau die 281 Sätze, die ein passendes Gegenstück aus den anderen Arbeitsstücken finden. Hash-Vergleiche sind ausschließlich CPU-basiert und daher sehr schnell. Anders sieht es aus, wenn die Datenbank erst frisch gestartet ist und kein pg_prewarm läuft. Dann muss die Datenbank tatsächlich von Disk lesen und die Zeiten sind deutlich länger. Der Hash-Vergleich ist zwar immer noch sehr schnell, aber das Einlesen der Daten schlägt sich in der „actual time" nieder.

Als Abschluss der Informationen zum SQL kommt noch die Filterbedingung für den Index *(siehe Abbildung 8)*. Im Index wird das Index-Attribut UUID mit dem aus dem vorherigen Join verglichen.

Als letzte Information des Explains sind noch die Zeiten für die Erstellung und die Zeit für die Ausführung des Plans angegeben *(siehe Abbildung 9)*. Wie immer gibt PSQL die Anzahl der ausgegebenen Zeilen als letzte Information an.

Zu beachten ist, dass sich die Zeiten nicht aufaddieren. Das liegt daran, dass die Parallelisierung zwar für einen einzelnen Prozess bedeutet, dass dieser eine bestimmte Zeit braucht, die für den User interessante Gesamtzeit aber vom Start des ersten Prozesses bis zur Ausgabe der letzten Zeile berechnet wird – ob das sequenziell gemacht wird oder parallelisiert, interessiert wahrscheinlich kaum eine Person, die auf eine Antwort von der Datenbank wartet. Das, was hier ausgegeben wird, ist nur ein momentanes Bild. Wenn viel mehr auf der Datenbank los ist, dann kommen nicht solche Zeiten zustande. Das macht Tests auch schwierig. Wie vergleicht man Zeiten, wenn die äußeren Umstände nicht dieselben sind?

Wozu ist das Ganze denn gut?

PostgreSQL plant überhaupt nicht mehr, wenn eine bestimmte Tabellenzahl in einem SQL überschritten wird. Das steht in der globalen Variable join_collapse_limit. Die Voreinstellung ist acht. Acht Tabellen mögen auf einem OLTP-System viel erscheinen, bei einem DWH mit einer Data-Vault-Architektur ist durch Hubs, Satelliten und Linktabellen schnell dieses Limit erreicht. Das Limit ist nicht fest, sondern kann pro Transaktion oder für eine Session geändert werden. Es ist dabei aber zu beachten, dass der Planungsaufwand nach der Formel Fakultät der Anzahl der Tabellen berechnet wird. Das sind bei acht Tabellen bereits über 40.000 Varianten, die sich der Planer anschauen könnte.

Wenn das Limit überschritten wird, dann führt der Planer das SQL von der

CLOUD NATIVE FESTIVAL

im Heide Park Soltau

CloudLand
www.cloudland.org

19. – 22. MAI 2026

Aufregende Atmosphäre eines Freizeitparks!

Neueste Trends & Innovationen rund um Cloud Native!

Ein Treffen mit den Hyperscalern AWS, Microsoft Azure & Google Cloud!

Spannende Vorträge, interaktive Workshops & kreatives Networking!

#CLOUDLAND2026

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33kB
              ->  Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 7: Zeile 8 und 9 des Explains (Quelle: Wolfgang Wilhelm)

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33kB
              ->  Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 8: Zeile 10 des Explains (Quelle: Wolfgang Wilhelm)

```
Nested Loop  (cost=11.88..2430.57 rows=1 width=65) (actual time=0.866..12.707 rows=281 loops=1
  ->  Hash Join  (cost=11.32..34.76 rows=281 width=102) (actual time=0.780..1.332 rows=281 loops=1)
        Hash Cond: ((l.uuid)::text = (t.uuid)::text)
        ->  Seq Scan on lager l  (cost=0.00..22.30 rows=430 width=45) (actual time=0.336..0.606 rows=430 loops=1)
        ->  Hash  (cost=7.81..7.81 rows=281 width=57) (actual time=0.405..0.406 rows=281 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 33kB
              ->  Seq Scan on texts t (cost=0.00..7.81 rows=281 width=57) (actual time=0.103..0.282 rows=281 loops=1)
  ->  Index Scan using komponente_pkey on komponente k (cost=0.55..8.53 rows=1 width=37) (actual time=0.040..0.040 rows=1 loops=281)
        Index Cond: ((uuid)::text = (l.uuid)::text)
        Filter: ((class_type)::text = 'Lager'::text)
Planning Time: 6.439 ms
Execution Time: 12.779 ms
(12 rows)
```

Abbildung 9: Zeilen 11 und 12 des Explains (Quelle: Wolfgang Wilhelm)

ersten angegebenen Tabelle abarbeitend bis zur letzten aus. Das ist aber nur ein Weg, wie eigene Ideen erzwungen werden können. Die anderen sind CTEs, „common table expressions“, also WITH-Queries *[1]*. Wer nicht (mehr) weiß, wie diese aussehen, findet in *Listing 2* die allgemeine Form.

Dort wird normalerweise bei nur einer Ansprache der WITH-Query ein Inlining durchgeführt, also ein Ersatz der Ansprache durch das SQL-Statement der WITH-Query. Erst wenn zwei oder mehr Ansprachen vorkommen, materialisiert PostgreSQL die Datenermittlung für die WITH-Query normalerweise im shared memory, falls dort genug Platz vorhanden ist. Dieses Verhalten kann auch schon bei einer Ansprache erzwungen werden, wenn man in der WITH-Query zusätzlich MATERIALIZED als Option angibt. Umgekehrt kann die Materialisierung durch NOT MATERIALIZED ausgeschlossen werden. Die Voraussetzung für diese Optionen ist PostgreSQL 12 oder höher. Aus Sicht des Planers sind die Daten wie in einer temporären Tabelle vorhanden. Dadurch kann bei der Erstellung des SQLs eine Vorgabe gemacht werden, was der Planer tun soll. Es wird also nicht mehr deklarativ gearbeitet, sondern imperativ.

Es gibt noch einen weiteren Weg, dem Planer Anweisungen zu geben, oder eher Verbote aufzustellen. Nicht nur die oben angesprochene Variable kann an eigene Bedürfnisse angepasst werden, sondern eine Menge mehr. Die Variablen *[2]* beginnen alle mit „enable_“ und diese können auch ausgeschaltet werden.

Gibt es da auch etwas von SQL?

Bisher haben wir noch kein einziges SQL-Kommando zum Tuning anderer benutzt. Das wird hiermit nachgeholt.

Mit SET *[3]* werden Variablen gesetzt, mit SHOW ausgelesen. SET LOCAL gilt nur für eine Transaktion.

PostgreSQL baut zwar Statistiken, aber untersucht keine Abhängigkeiten zwischen den Attributen einer Tabelle. Dafür gibt es das Kommando CREATE STATISTICS *[4]*.

Falls das immer noch nicht reicht, dann muss schon langsam an ein CREATE INDEX gedacht werden, wobei der richtige Typ und gegebenenfalls ein INCLUDE anzugeben ist, aber das wäre ein weiterer Artikel.

```
WITH klauselname AS ( sql-statement )
[, weitere WITH-Klauseln]
SELECT attribut1, attribut2…
FROM klauselname…
```

Listing 2: Die allgemeine Form einer WITH-Query

Quellen

[1] *https://www.postgresql.org/docs/current/queries-with.html*
[2] *https://postgresqlco.nf*
[3] *https://www.postgresql.org/docs/current/sql-set.html*
[4] *https://www.postgresql.org/docs/18/sql-createstatistics.html*

Über den Autor

Wolfgang Wilhelm ist Senior Consultant bei der Opitz Consulting Deutschland GmbH. Seine Datenbankkarriere begann auf einem BS2000-Großrechner. Seit 2000 beschäftigt er sich mit PostgreSQL, seit 2012 intensiv mit Oracle.

Wolfgang Wilhelm
wolfgang.wilhelm@opitz-consulting.com


VERTICAL SPEED
NAV
COM/VLOC
CDI
OBS
125.20

Automatisierte Health Checks für SQL-Server: Transparenz, Sicherheit und Stabilität auf Knopfdruck

Ilgar Machmudov und Andreas Ströbel, Opitz Consulting Deutschland

Immer wieder wird von Datenbanken berichtet, die gehackt, beschädigt oder sogar zerstört wurden. Dabei gehen oft sogar kritische Daten verloren, und manchmal ist auch die Rede von fehlenden funktionsfähigen Backups. Doch liegt das tatsächlich nur an mangelnder Sorgfalt? In der Praxis zeigt sich immer wieder, dass selbst äußerst gewissenhaft arbeitende IT-Teams mit versteckten Problemen, Konfigurationsabweichungen und unerkannten Risiken konfrontiert sind. Regelmäßige, strukturierte System Checks können hier entscheidend zur Stabilität beitragen – insbesondere dann, wenn sie automatisiert und skalierbar erfolgen.

Die Gründe für regelmäßige System Checks sind vielfältig: Das Ziel ist die Sicherstellung eines zuverlässigen und stabilen Betriebs. System Checks helfen, Fehler frühzeitig zu erkennen und zu beheben – sei es im Betriebssystem, in der Datenbankkonfiguration, in Treibern, Anwendungen oder in der Hardware. Gerade nach Upgrades oder Updates, bei regelmäßiger (z. B. bei nächtlicher Massendatenverarbeitung) oder saisonal hoher Last (z. B. beim Monats- oder Jahresabschluss), oder in heterogenen Landschaften ist eine valide Überprüfung unerlässlich. Auch die Frage nach der Kompatibilität aller eingesetzten Komponenten und Versionen spielt eine zentrale Rolle. Kurz gesagt: System Checks sind ein elementarer Baustein der Qualitätssicherung und Leistungsoptimierung.

Betrachtet man das nun konkret bezogen auf Microsoft-SQL-Server-Datenbanken, dann stellen sich weitere Fragen. Gibt es generelle Probleme? Funktionieren die Schnittstellen, das Backup, und vor allem das Recovery? Gibt es ein Disaster- Recovery-Konzept? Im Bereich Monitoring sollte überprüft werden, ob alle für einen sicheren Betrieb relevanten Metriken erfasst werden und ob die Schwellwerte passend konfiguriert sind. Und nicht zuletzt muss das Thema Security betrachtet werden – gibt es Sicherheitslücken, die zu patchen sind? Außerdem verändert sich in jedem System im Lauf der Zeit die Performance, meist verschlechtert sie sich durch gewachsene und sich verändernde SQL-Server-Landschaften, zwischenzeitlich veraltete Infrastruktur und nicht oder nur unzureichend Performance-optimierte Datenbanken.

In vielen Unternehmen existieren zudem SQL-Server-Instanzen, die außerhalb der zentralen IT entstanden sind – etwa in Fachabteilungen, die kurzfristig Lösungen benötigen. Diese „Schatten-IT“ entzieht sich häufig den gängigen Sicherheits-, Backup- und Compliance-Prozessen. Ein strukturierter System Check hilft, solche Umgebungen sichtbar zu machen und in die zentrale Governance zu integrieren.

Besonders dort, wo die Umgebungen komplexer werden, gibt es darüber hinaus weitere Fragen, vor allem zu Sicherheit und Compliance. Wer ist wofür verantwortlich? Werden Unternehmensstandards eingehalten? Sind alle Systeme korrekt lizenziert. Dazu kommt, dass gegebenenfalls mehr Ressourcen gebraucht werden als eigentlich nötig.

Alle diese Aspekte sollten in System- oder Health Checks regelmäßig überprüft werden. Bei einem geeigneten Vorgehen und guter Tool-Auswahl lassen sich darüber hinaus weitere Ergebnisse damit erzielen. So kann für die jeweils aktuelle Systemlandschaft automatisch die Dokumentation erzeugt werden, und die erwähnte, eben mal schnell im Fachbereich entstandene „Schatten-IT“ wird aufgefunden.

Wiederkehrende Checks: manuell oder automatisiert?

Eine wiederkehrende Überprüfung soll Antworten auf die genannten Punkte liefern. Doch wie kann man dabei möglichst effektiv vorgehen, insbesondere vor dem Hintergrund, dass diese Prüfungen regelmäßig und wiederholt vorgenommen werden sollen? Ein manueller Ansatz ist in der Regel aufwändiger, führt aber ge-

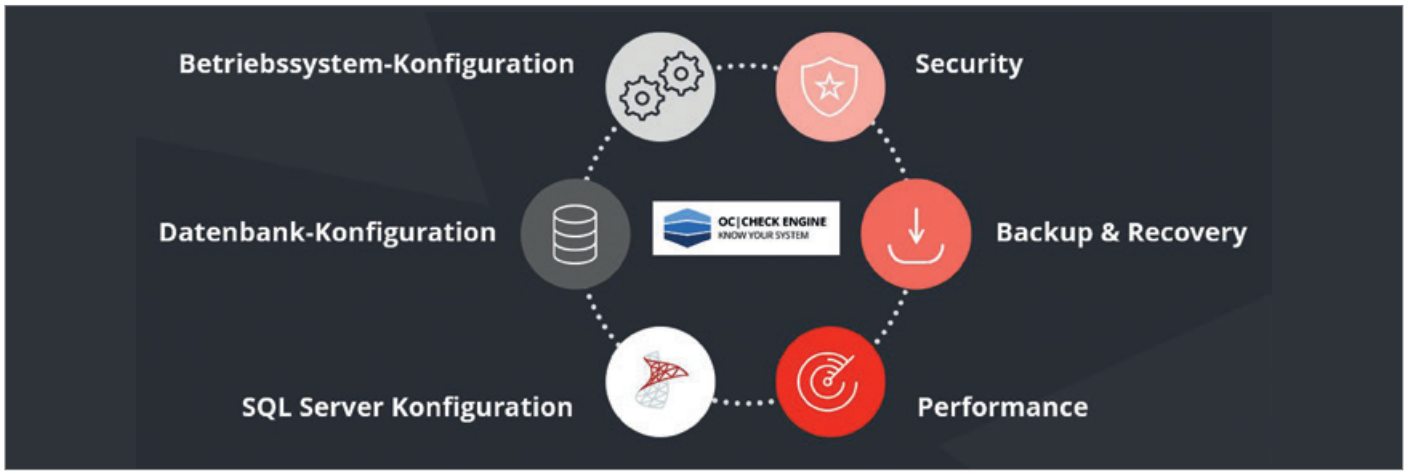


Abbildung 1: Analysebereiche der OC|Check Engine (Quelle: Opitz Consulting)

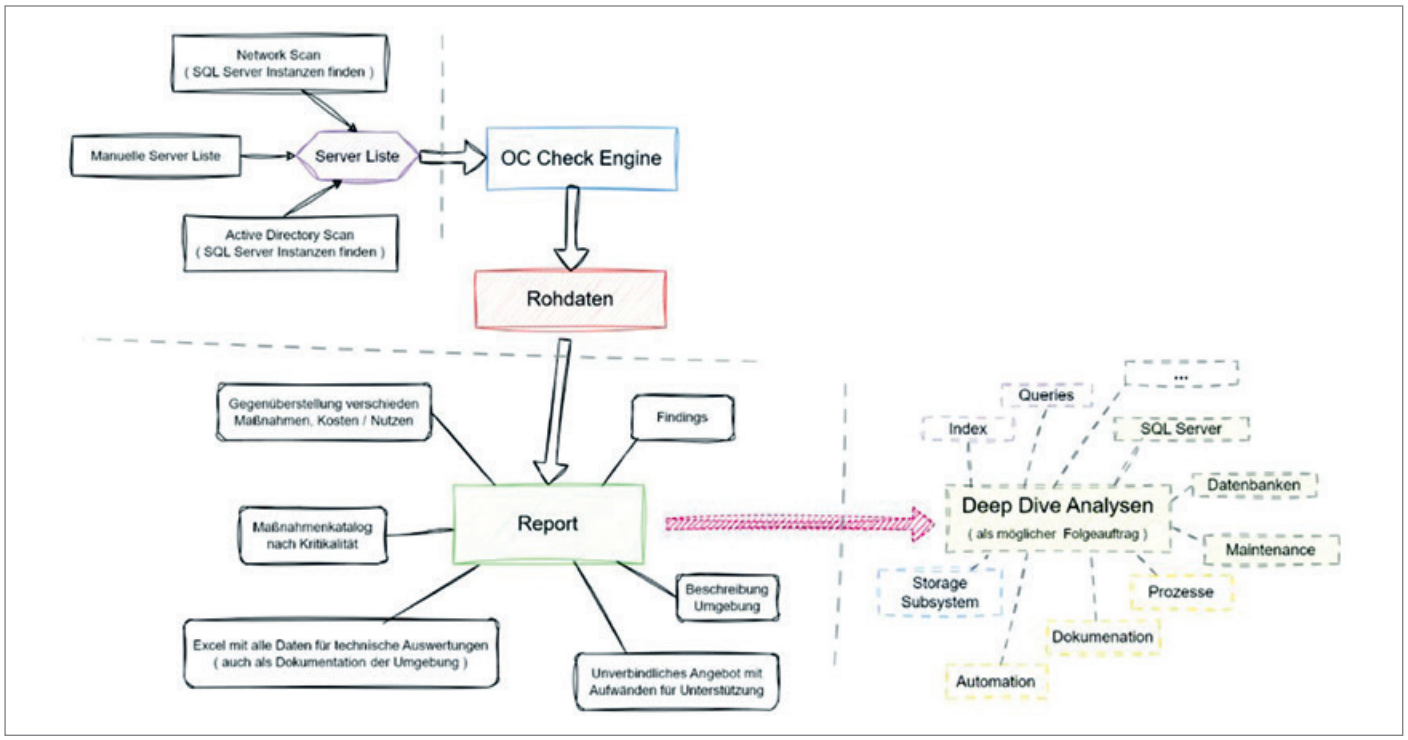


Abbildung 2: Workflow bei der Durchführung des System Checks (Quelle: Opitz Consulting)

gebenenfalls zu punktgenaueren und detaillierteren Ergebnissen. Die Durchführung automatisierter Checks kann hingegen breit eingesetzt werden und bietet einen besseren Gesamtüberblick über die vollständigen Landschaften. Und gerade unter dem Aspekt der Wiederholbarkeit bietet die Automatisierung entscheidende Vorteile.

Vergleich: manueller vs. automatisierter Systemcheck

In der folgenden Tabelle *(siehe Tabelle 1)* sind beide Varianten gegenübergestellt. Unter Umständen ergänzen manuelle Checks die automatisierte Vorgehensweise, wenn diese konkrete Anhaltspunkte liefert, dass eventuell Findings behoben und analysiert werden müssen.

Automatisierte Analyse mit der OC|Check Engine

Mit der OC|Check Engine stellt Opitz Consulting ein agentenloses, PowerShell-basiertes Framework bereit, das SQL-Server-Umgebungen automatisiert erkennt, analysiert und bewertet. Über 90 Prüfmodule mit mehr als 80 Konfigurationsparametern decken alle relevanten Bereiche ab – von Betriebssystem, Lizenzierung und AlwaysOn-Konfiguration über Backup und Recovery sowie Performance bis hin zu Security. Die Ergebnisse werden strukturiert klassifiziert, priorisiert und in übersichtlichen Reports bereitgestellt. Unternehmen erhalten damit eine vollständige, reproduzierbare und objektive Sicht auf ihre SQL-Server-Landschaft sowie klare Handlungsempfehlungen *(siehe Abbildung 1)*.

Abbildung 2 zeigt den generellen Workflow bei der Durchführung des System Checks. Als Input wird eine Liste aller zu untersuchenden Server benötigt,

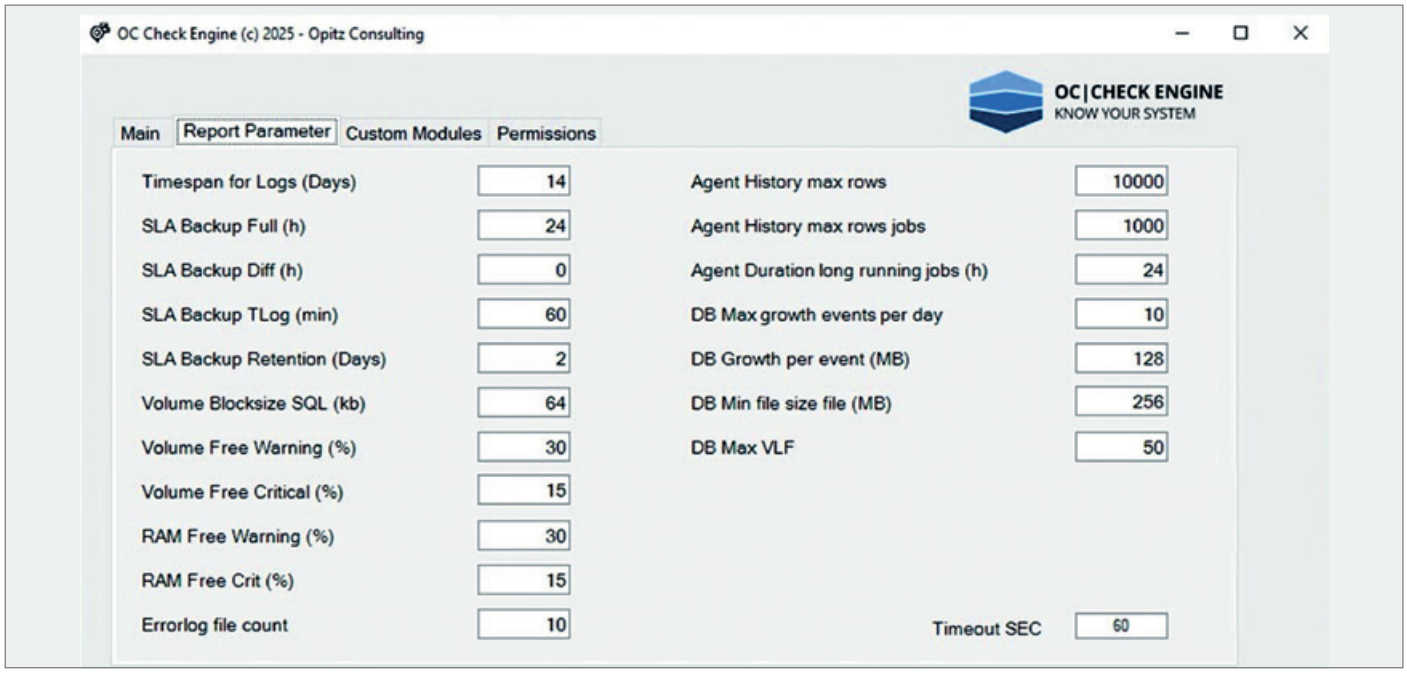


Abbildung 3: Parametrisierung (Quelle: Opitz Consulting)

Kriterium	manueller Check	automatisiertes Vorgehen
Analysefokus	tiefgehende Analyse **einzelner Systeme** mit spezifischen Problemschwerpunkten	breite Analyse der **gesamten SQL-Server-Umgebung** mit standardisierten Prüfmodulen
Erfassungsmethode	detaillierte, manuelle Untersuchung mit Expertenwissen	automatisierte, skalierbare Erfassung, Auswertung durch Experten
Anpassungsfähigkeit	hohe Flexibilität bei der Untersuchung **spezieller Probleme**	**standardisierte Prüfungen** mit vordefinierten Modulen
Problemidentifikation	Identifikation komplexer, kontextabhängiger Probleme	effiziente Erkennung bekannter Konfigurationsprobleme und Best-Practice-Verstöße
Empfehlungsqualität	maßgeschneiderte, **fallspezifische** Handlungsempfehlungen	**individuelle Lösungsansätze** mit konkreten Aufwandschätzungen für die gesamte Umgebung, inklusive priorisierter Handlungsempfehlungen
Ergebnisdokumentation	umfassende Berichte	strukturierte, kategorisierte Findings mit verschiedenen Ausgabeformaten und Zusammenfassung
Einsatzszenarien	optimal für **kritische Einzelsysteme** und **spezifische Problemstellungen**	ideal für **große SQL-Server-Umgebungen** und regelmäßige Überprüfungen
Kostenstruktur	hohe Kosten pro System	günstiger bei größeren Umgebungen

Tabelle 1: Überblick Manueller vs. automatisierter Systemcheck

die entweder manuell erfasst oder automatisch erzeugt werden kann, zum Beispiel anhand einer Configuration Management Database (CMDB) oder durch einen entsprechenden Network Scan. Die OC|Check Engine überprüft dann alle Systeme aus der Liste. Die erzeugten Rohdaten werden in strukturierte Reports verwandelt, die alle Ergebnisse in einer übersichtlichen Darstellung zusammenfassen. Dies ermöglicht dann gegebenenfalls weitere, tiefgehende Analysen, die durch entsprechende Experten durchgeführt werden können.

Vor der Durchführung der Checks ist es erforderlich, sich über passende Kriterien und die daraus resultierenden Parameter im Klaren zu sein *(siehe*

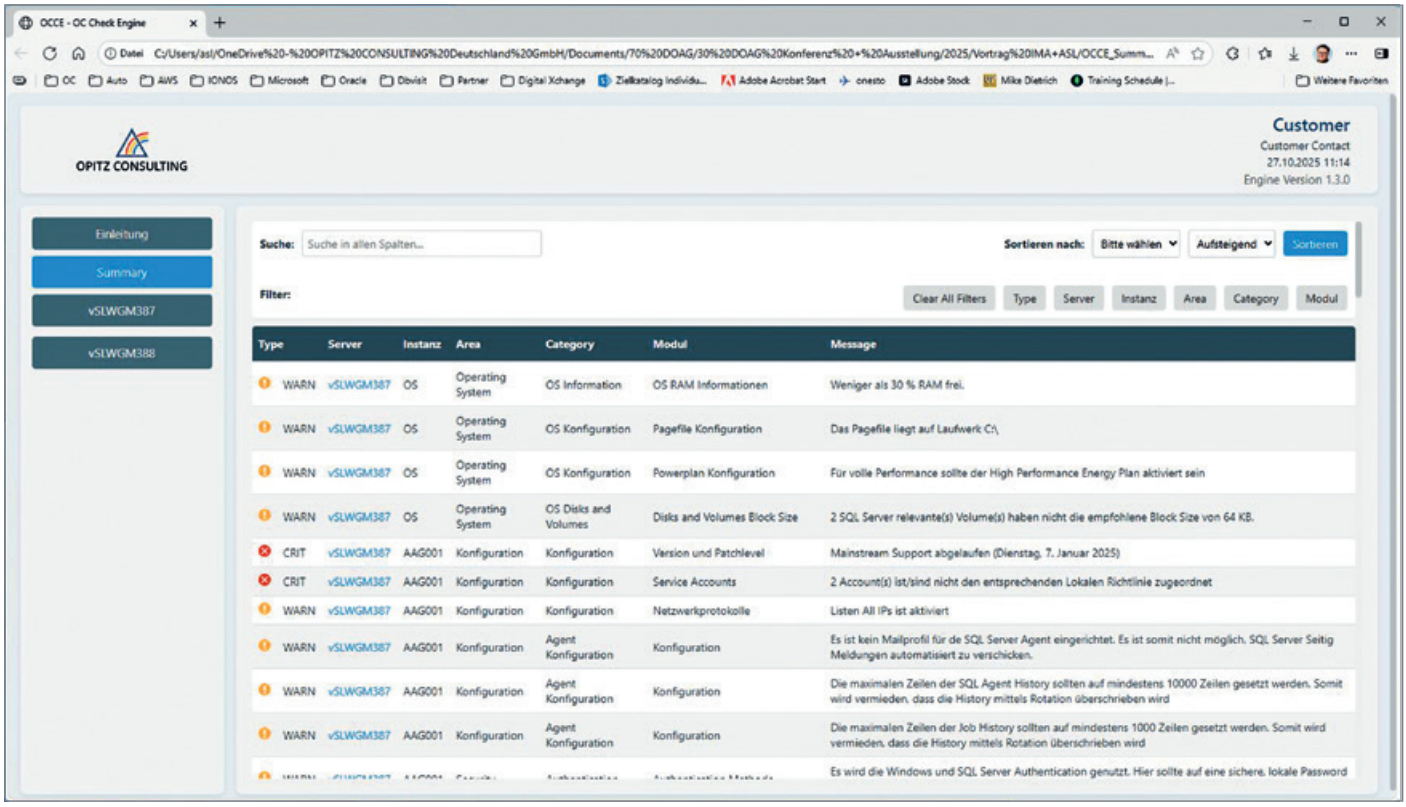


Abbildung 4: Ergebnis-Report (Quelle: Opitz Consulting)

Abbildung 3). Wie viel Platz muss in der Storage verfügbar sein? Wie alt darf ein Backup werden? Wie sieht eine sinnvolle Speichernutzung aus?

Die Ergebnisse selbst liegen zunächst in einem JSON-Format vor (Rohdaten). Daraus werden die Reports erstellt, die im HTML-Format betrachtet oder auch als Excel-Dateien weiterverarbeitet und analysiert werden können. In beiden Varianten sind übersichtliche Filtermöglichkeiten gegeben. Die Darstellung erfolgt zum einen in einer zusammenfassenden Übersicht (Management Summary), dann aber auch detailliert für jedes System, jede Datenbank und jede Instanz.

Jetzt ermöglichen die zuvor definierten Parameter eine farbliche Hervorhebung der Ergebnisse, insbesondere bei Abweichungen von den vorgegebenen Werten. Bereits in der Übersicht wird so ein erster Systemstatus erkennbar *(siehe Abbildung 4)*.

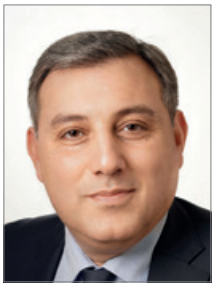
Neben den reinen Statusinformationen zeigen die Reports die konkreten, bereits vorpriorisierten Findings aus den Umgebungen. Darüber hinaus werden Handlungsempfehlungen dargestellt, um die jeweiligen Systeme zu optimieren.

Fazit: Mehr Transparenz und Stabilität durch Automatisierung

Regelmäßige, idealerweise automatisierte System Checks sind ein wirksames Mittel, um Risiken frühzeitig zu erkennen und Ausfälle zu vermeiden. Sie schaffen Transparenz in der gesamten SQL-Server-Landschaft, unterstützen die Einhaltung von Unternehmensstandards und ermöglichen planbare, nachvollziehbare Optimierungsschritte. Automatisierte Ansätze wie die OC|Check Engine kombinieren Skalierbarkeit mit konsistenten Ergebnissen und geben IT-Teams eine verlässliche Grundlage für Betrieb, Planung und Weiterentwicklung.

Über die Autoren

Die Autoren sind bei Opitz Consulting in der Division Systems für Design, Aufbau, Betrieb inklusive Managed Services für Datenbanksysteme verantwortlich. Die Schwerpunkte liegen dabei auf Oracle, Microsoft SQL Server, PostgreSQL und MongoDB, jeweils on-premises und in Cloud-Umgebungen.



Ilgar Machmudov
ilgar.machmudov@opitz-consulting.com

Andreas Ströbel
andreas.stroebel@opitz-consulting.com

Oracle 23ai/26ai-Funktionalitäten in 19c-Datenbanken

Christian Pfundtner, DB Masters

Oracle portiert immer wieder Funktionalitäten neuerer Datenbank Releases in ältere Versionen zurück. Beispielsweise wurden SQL Macros, Blockchain und Immutable Table von Oracle 21c in Oracle 19c eingebaut. Mit den aktuellen Release Updates beginnt Oracle auch einige Oracle 23ai-Funktionalitäten – seit Oktober 2025 in Oracle 26ai umbenannt – für die Version 19c bereit zu stellen. Schauen wir uns das einmal genauer an!

PL/SQL Package DBMS_DEVELOPER

Das PL/SQL Package DBMS_DEVELOPER bietet eine für Entwickler einfache Möglichkeit, die Metadaten für Datenbank-Objekte wie Tabellen und Indizes im JSON-Format abzuholen. Bisher ging das nur mit dem Package DBMS_METADATA, allerdings liefert das nur DDL oder XML und kein JSON-Format. Nicht immer haben Entwickler Zugriff auf DBMS_METADATA.

Mittels DBMS_DEVELOPER können die Metadaten für folgende Objekt-Typen im JSON- Format dargestellt werden:

- Tabellen oder Views, die der Benutzer auch lesen (SELECT oder READ) darf.
- Indizes auf Tabellen, die der Benutzer sehen kann.
- Synonyme, die der Benutzer sieht.

Aktuell gibt es im Package nur eine Funktion: GET_METADATA *(siehe Listing 1)*.

Die ersten drei Parameter sind selbsterklärend. Mit LEVEL wird definiert, wie ausführlich die Information im JSON-Dokument sein soll. Dafür stehen die Werte BASIC, TYPICAL und ALL zur Verfügung. BASIC gibt nur die notwendigsten Informationen: Die Spaltennamen und Datentypen bei Tabellen. Bei TYPICAL werden noch weitere Metadaten ausgegeben, beispielsweise Eigenschaften von Views und Constraint-Informationen. Mit ALL gibt es weitere Details wie beispielsweise die vergebenen Objektberechtigungen.

ETAG ist ein eindeutiger Identifier, der als Checksumme für das gesamte JSON-Objekt berechnet wird. Wenn das aktuelle Objekt denselben Wert wie dieser Parameter hat, ist das Ergebnis des Aufrufs NULL. Auf diese Weise kann einfach geprüft werden, ob sich die Definition eines Objekts im Vergleich zu einem vorherigen Aufruf geändert hat.

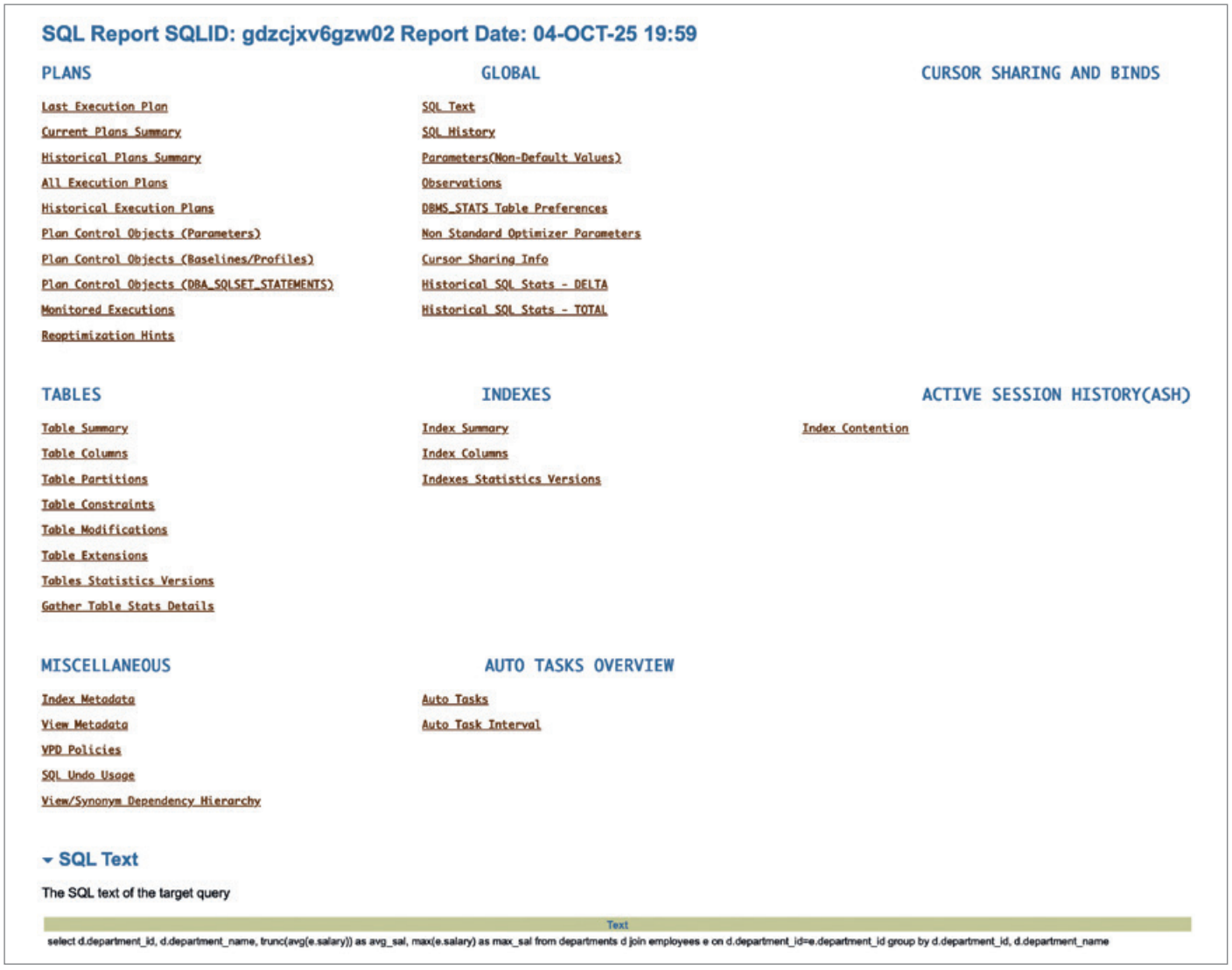


Abbildung 1: Beispiel Output my_sqldiag.html (Quelle: Christian Pfundtner)

```
DBMS_DEVELOPER.GET_METADATA (
name            IN VARCHAR2,
schema          IN VARCHAR2 DEFAULT NULL,
object_type     IN VARCHAR2 DEFAULT NULL,
level           IN VARCHAR2 DEFAULT 'TYPICAL'
etag            IN RAW      DEFAULT NULL)
RETURN JSON;
```

Listing 1: Definition DBMS_DEVELOPER.GET_METADATA

Der Output sieht typischerweise (verkürzt) wie in *Listing 2* aus.

Referenzen:

- DBMS_DEVELOPER: the new developer friendly metadata retrieval for database objects *https://blogs.oracle.com/coretec/post/dbmsdeveloper-the-new-developer-friendly-meta-data-retrieval-in-23ai*
- Oracle-23ai-Funktionalität in Oracle 19c: DBMS_DEVELOPER PL/SQL Package *https://www.database-blog.at/2025/10/09/oracle-23ai-funktionalitaet-in-oracle-19c-dbms_developer-pl-sql-package/*

IF [NOT] EXISTS Syntax-Support

Scripts, die versuchen, Objekte anzulegen, die schon existieren, liefen bisher in einen Fehler wie beispielsweise „ORA-00955: name is already used by an existing object“.

Die Überprüfung, ob Fehler eines Scripts relevant sind, oder nur auf Grund schon existierender Objekte entstehen, war dementsprechend aufwändig, zumal anhand der Fehlermeldung nicht zu erkennen war, welches Objekt gemeint ist. Andere Datenbanken können schon seit längerem eine Syntaxvariante mit IF NOT EXISTS, um genau solche Fehlermeldungen zu unterdrücken. Diese Funktionalität wurde in Oracle 23ai eingebaut und steht jetzt auch für Oracle 19c zur Verfügung. Ein Beispiel für das Erzeugen einer Tabelle mittels Scripts ist in *Listing 3* dargestellt.

Wird dieses Script erstmalig ausgeführt, gibt es „Table created“ aus. Ab dem nächsten Aufruf gibt es aber den Fehler „ORA-00955: name is already used by an existing object“ aus.

Wird das Script so angepasst *(siehe Listing 4)*, gibt es immer nur „Table created“ zurück, egal, ob die Tabelle schon exis-

tierte oder neu angelegt wurde. Damit wird die Überprüfung des Script-Logfiles auf relevante Fehler viel einfacher.

Der Zusatz IF NOT EXISTS darf bei den meisten CREATE-Statements von Datenbank-Objekten verwendet werden, einzig die Kombination mit CREATE OR REPLACE ist verständlicherweise nicht erlaubt. Ebenso funktioniert IF EXISTS bei den meisten DROP-Statements.

Ein Problem kann damit aber nicht gelöst werden: Was passiert, wenn beispielsweise bei einer Tabelle eine neue Spalte hinzugekommen ist? Der Check prüft nur, ob das Objekt existiert, nicht, ob es ident ist. Dieses Problem gibt es aber auch bei allen anderen Datenbankherstellern.

Referenzen:

- IF [NOT] EXISTS DDL Clause in Oracle Database 23ai
  https://oracle-base.com/articles/23/if-not-exists-ddl-clause-23
- Oracle 23c Preview: DDL und Administration
  https://www.database-blog.at/2022/10/26/oracle-23c-preview-ddl-und-administration/

Annotations

Annotations erlauben das Speichern zusätzlicher Metadaten zu Objekten in Form von Key/Value-Paaren oder einfach nur Namen. Mit diesen Annotations können nun zentral in der Datenbank Datenklassifikationen und Darstellungsempfehlungen abgelegt werden, damit – sofern diese Informationen von den Applikationsentwicklern auch entsprechend berücksichtig werden – die Darstellung über mehrere Applikationen hinweg konsistent ist.

Schauen wir uns Beispiele dazu an. Im ersten Beispiel sehen wir ANNOTATIONS auf einem Datenbank-Objekt. Hier können beliebige Key/Value-Paare definiert beziehungsweise beliebiger Freitext (Kundendaten) spezifiziert werden *(siehe Listing 5)*.

Dadurch können Anwendungen und Tools Entscheidungen treffen, die auf diesen Anmerkungen basieren. Beispielsweise dürfen auf Grund des Datenschutzes Informationen aus dieser Tabelle in der Applikation nur mit entsprechenden Berechtigungen angezeigt werden. Alle anderen Benutzer bekommen anonymisierte Informationen. Welche Abteilungen den Zugriff auf diese Daten erhalten dürfen, kann hier ebenfalls spezifiziert werden. Dadurch können interaktive Tools (zum Beispiel Query Builder) sicherstellen, dass nur berechtigte Benutzer auf die Daten zugreifen dürfen.

Ein weiteres Beispiel für den Einsatz von Annotations auf Spalten könnte wie in *Listing 6* aussehen.

```
{
  "objectType" : "TABLE",
  "objectInfo" :
  {
    "name" : "EMPLOYEES",
    "schema" : "HR",
    "columns" :
    [
      {
        "name" : "EMPLOYEE_ID",
        "notNull" : true,
        "dataType" :
        {
          "type" : "NUMBER"
        }
...
      }
    ]
  },
  "etag" : "9ADC6D08AE0D5C82D0188E53F2ECD5B1"
}
```

Listing 2: Beispieloutput im JSON-Format

```
CREATE TABLE FOO (X NUMBER);
```

Listing 3: Beispiel für ein CREATE TABLE

```
CREATE TABLE IF NOT EXISTS FOO (X NUMBER);
```

Listing 4: CREATE TABLE mit neuer Syntax

```
CREATE TABLE konto (....)
  ANNOTATIONS ( Datenschutz 'Hoch',
                Abteilungen 'Vertrieb, Buchhaltung, Vorstand',
                Kundendaten );
```

Listing 5: Beispiel CREATE TABLE mit Annotation

Damit können Informationen hinterlegt werden, wie die Spalten benannt werden sollen (Display) oder, dass sie überhaupt nicht angezeigt werden dürfen (UI_Hidden).

Die Datenbank selbst interpretiert Annotations nicht, das ist ausschließlich Aufgabe der Applikation.

Referenzen:

- Oracle 19c Schema Annotations
  https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/

DOAG EVE

JAN	01 Neujahr Do 1.	Fr 2.	Sa 3.	So 4.	02 Mo 5.	Di 6.	Mi 7.	Do 8.	Fr 9.	Sa 10.	So 11.	03 Mo 12	Di 13.	Mi 14.	Do 15.
FEB	So 1.	06 Mo 2.	Di 3.	Mi 4.	Do 5.	Fr 6.	Sa 7.	So 8.	07 Mo 9.	Di 10.	Mi 11.	Do 12.	Fr 13.	Sa 14.	So 15.
MAR	So 1.	10 Mo 2.	Di 3.	Mi 4.	Do 5.	Fr 6.	Sa 7.	So 8.	11 Mo 9.	Di 10.	Mi 11.	Do 12.	Fr 13.	Sa 14.	So 15.
APR	Mi 1.	Do 2.	Karfreitag Fr 3.	Sa 4.	Oster-sonntag So 5.	15 Oster-montag Mo 6.	Di 7.	Mi 8.	Do 9.	Fr 10.	Sa 11.	So 12.	16 Mo 13.	Di 14.	Mi 15.
MAI	Tag der Arbeit Fr 1.	Sa 2.	So 3.	19 Mo 4.	Di 5.	Mi 6.	Do 7.	Fr 8.	Sa 9.	So 10.	20 Mo 11.	Di 12.	Mi 13.	Christi Himmelfahrt Do 14.	Fr 15.
JUN	23 Mo 1.	Di 2.	Mi 3.	Do 4.	Fr 5.	Sa 6.	So 7.	24 Mo 8.	Di 9.	Mi 10.	Do 11.	Fr 12.	Sa 13.	So 14.	25 Mo 15.
JUL	Mi 1.	Do 2.	Fr 3.	Sa 4.	So 5.	28 Mo 6.	Di 7.	Mi 8.	Do 9.	Fr 10.	Sa 11.	So 12.	29 Mo 13.	Di 14.	Mi 15.
AUG	Sa 1.	So 2.	32 Mo 3.	Di 4.	Mi 5.	Do 6.	Fr 7.	Sa 8.	So 9.	33 Mo 10.	Di 11.	Mi 12.	Do 13.	Fr 14.	Sa 15.
SEP	Di 1.	Mi 2.	Do 3.	Fr 4.	Sa 5.	So 6.	37 Mo 7.	Di 8.	Mi 9.	Do 10.	Fr 11.	Sa 12.	So 13.	38 Mo 14.	Di 15.
OKT	Do 1.	Fr 2.	Tag der deutschen Einheit Sa 3.	So 4.	41 Mo 5.	Di 6.	Mi 7.	Do 8.	Fr 9.	Sa 10.	So 11.	42 Mo 12.	Di 13.	Mi 14.	Do 15.
NOV	So 1.	45 Mo 2.	Di 3.	Mi 4.	Do 5.	Fr 6.	Sa 7.	So 8.	46 Mo 9.	Di 10.	Mi 11.	Do 12.	Fr 13.	Sa 14.	So 15.
DEZ	Di 1.	Mi 2.	Do 3.	Fr 4.	Sa 5.	So 6.	50 Mo 7.	Di 8.	Mi 9.	Do 10.	Fr 11.	Sa 12.	So 13.	51 Mo 14.	Di 15.

NTKALENDER

Fr 16.	Sa 17.	So 18.	04 Mo 19.	Di 20.	Mi 21.	Do 22.	Fr 23.	Sa 24.	So 25.	05 Mo 26.	Di 27.	Mi 28.	Do 29.	Fr 30.	Sa 31.
08 Mo 16.	Di 17.	Mi 18.	Do 19.	Fr 20.	Sa 21.	So 22.	09 Mo 23.	Di 24.	Mi 25.	Do 26.	Fr 27.	Sa 28.			
12 Mo 16.	Di 17.	Mi 18.	Do 19.	Fr 20.	Sa 21.	So 22.	13 Mo 23.	Di 24.	Mi 25.	Do 26.	Fr 27.	Sa 28.	So 29.	14 Mo 30.	Di 31.
Do 16.	Fr 17.	Sa 18.	So 19.	17 Mo 20.	Di 21.	Mi 22.	Do 23.	Fr 24.	Sa 25.	So 26.	18 Mo 27.	Di 28.	Mi 29.	Do 30.	
Sa 16.	So 17.				Do 21.	Fr 22.	Sa 23.	Pfingst-sonntag So 24.	22 Pfingst-montag Mo 25.	Di 26.	Mi 27.	Do 28.	Fr 29.	Sa 30.	So 31.
Di 16.	Mi 17.	Do 18.	Fr 19.	Sa 20.	So 21.	26 Mo 22.	Di 23.	Mi 24.	Do 25.	Fr 26.	Sa 27.	So 28.	27 Mo 29.	Di 30.	

Do 16.	Fr 17.	Sa 18.	So 19.	30 Mo 20.	Di 21.	Mi 22.	Do 23.	Fr 24.	Sa 25.	So 26.	31 Mo 27.	Di 28.	Mi 29.	Do 30.	Fr 31.
So 16.	34 Mo 17.	Di 18.	Mi 19.	Do 20.	Fr 21.	Sa 22.	So 23.	35 Mo 24.	Di 25.	Mi 26.	Do 27.	Fr 28.	Sa 29.	So 30.	32 Mo 31.
Mi 16.	Do 17.	Fr 18.	Sa 19.	So 20.	39 Mo 21.	Di 22.	Mi 23.	Do 24.	Fr 25.	Sa 26.	So 27.	40 Mo 28.	Di 29.	Mi 30.	
Fr 16.	Sa 17.	So 18.	43 Mo 19.	Di 20.	Mi 21.	Do 22.	Fr 23.	Sa 24.	So 25.	44 Mo 26.	Di 27.	Mi 28.	Do 29.	Fr 30.	Sa 31.
47 Mo 16.	2026 DOAG Konferenz + Ausstellung				Sa 21.	So 22.	48 Mo 23.	Di 24.	Mi 25.	Do 26.	Fr 27.	Sa 28.	So 29.	49 Mo 30.	
Mi 16.	Do 17.	Fr 18.	Sa 19.	So 20.	52 Mo 21.	Di 22.	Mi 23.	Heilig-abend Do 24.	1.Weih-nachtstag Fr 25.	2.Weih-nachtstag Sa 26.	So 27.	53 Mo 28.	Di 29.	Mi 30.	Silvester Do 31.

OAG e.V. (Stand: 07.01.2026)
lin | 2026 | Alle Angaben ohne Gewähr

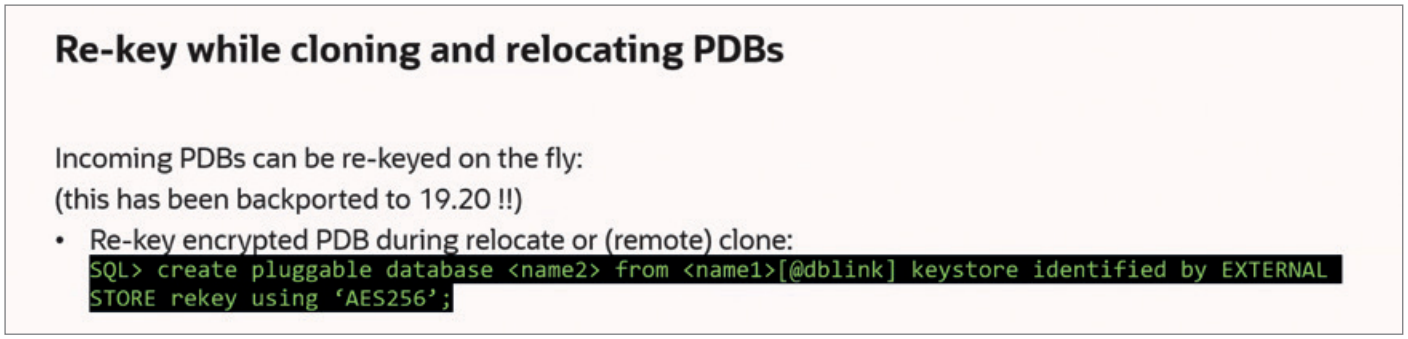


Abbildung 4: Ergebnis-Report (Quelle: Opitz Consulting)

```
CREATE TABLE mitarbeiter
( oid    NUMBER(10)   ANNOTATIONS (Identity, Display 'Mitarbeiternummer', Group 'MA_Info'),
  name   VARCHAR2(30) ANNOTATIONS (Display 'Mitarbeitername', Group 'MA_Info'),
  gehalt NUMBER       ANNOTATIONS (Display 'Gehalt', UI_Hidden)
) ANNOTATIONS ( Display 'Mitarbeitertabelle');
```

Listing 6: Beispiel CREATE TABLE mit Annotations auf allen Ebenen

```
set tab off
set feedback on sql_id
SELECT d.department_id, d.department_name,
       trunc(avg(e.salary)) AS avg_sal,
       max(e.salary) AS max_sal
  FROM departments d join employees e ON d.department_id=e.department_id
 GROUP BY d.department_id, d.department_name;

DEPARTMENT_ID DEPARTMENT_NAME                  AVG_SAL    MAX_SAL
------------- ------------------------------ ---------- ----------
           10 Administration                       4400       4400
           20 Marketing                            9500      13000
           30 Purchasing                           4150      11000
           40 Human Resources                      6500       6500
           50 Shipping                             3475       8200
           60 IT                                   5760       9000
           70 Public Relations                    10000      10000
           80 Sales                                8955      14000
           90 Executive                           19333      24000
          100 Finance                              8601      12008
          110 Accounting                          10154      12008

11 rows selected.

SQL_ID: gdzcjxv6gzw02
```

Listing 7: Beispiel mit SET FEEDBACK ON SQL_ID

```
set feedback on
var report clob;
exec :report := dbms_sqldiag.report_sql('gdzcjxv6gzw02');

PL/SQL procedure successfully completed.
```

Listing 8: Erzeugen eines CLOBs mit dem Output von DBMS_SQLDIAG.REPORT_SQL

application-data-usage.html#GUID-D5D5615C-BB2C-4833-A9AF-6BAF0BF-9CEC0

- Oracle 23ai-Funktionalität in Oracle 19c: Annotations
  https://www.database-blog.at/2025/10/13/oracle-23ai-funktionalita-et-in-oracle-19c-annotations/
- Oracle 23c Preview: DDL und Administration
  https://www.database-blog.at/2022/10/26/oracle-23c-preview-ddl-und-administration/

SQL Diagnostic Report

Im Package DBMS_SQLDIAG gibt es eine neue Funktion mit dem Namen REPORT_SQL. Diese Funktion liefert alle relevanten Informationen zu einem SQL-Statement, gesammelt in einem CLOB, das ein HTML-File mit allen Infos enthält.

Gehen wir einmal Schritt für Schritt vor. Wir benötigen ein SQL-Statement inklusive dessen SQL_ID – am einfachsten direkt im SQLPlus *(siehe Listing 7)*.

Für diese SQL_ID erzeugen wir jetzt einen Report als CLOB *(siehe Listing 8)*.

Und den Inhalt des CLOBs schreiben wir nun in ein HTML-File *(siehe Listing 9)*.

In diesem Report *(siehe Abbildung 1)* findet man jetzt alle relevanten und interessanten Details zu dem SQL-Statement, den betroffenen Tabellen und deren Indizes, ob das Statement schon in Auto Tuning Tasks aufgefallen ist und vieles

mehr. Bei den ersten Tests bei komplexeren Statements sind noch einige kleine Fehler aufgefallen. Beispielsweise sind die Ausführungspläne aktuell unleserlich, weil die Zeilenumbrüche fehlen. Dass SQL-Statements nach 1000 Zeichen abgeschnitten werden, ist auch nicht hilfreich. Beide Fehler sind bei Oracle bekannt und werden voraussichtlich in einem der nächsten RUs behoben.

Hinweis: Wenn man versuchen würde, den Report direkt in ein File zu schreiben, endet das mit einem Fehler *(siehe Listing 10)*.

Der Grund ist, dass in der Funktion REPORT_SQL das CLOB mittels DML zusammengebaut wird – und das ist in einem SELECT nicht erlaubt.

Benötigte Privilegien und Oracle-Lizenzen:

Für die Nutzung von DBMS_SQLDIAG benötigt man die Rolle ADVISOR. Da Informationen ausgegeben werden, auf die man nur mit einer Lizenz für das Diagnostic Management Pack zugreifen darf – beispielsweise Active Session History sowie DBA_HIST Views – ist davon auszugehen, dass für die Nutzung ebenfalls diese Management-Pack-Lizenz benötigt wird.

Referenzen:

- Oracle 23ai Funktionalität in Oracle 19c: SQL Diagnostic Report
  https://www.database-blog.at/2025/10/14/oracle-23ai-funktionalitaet-in-oracle-19c-sql-diagnostic-report/
- Diagnose SQL performance with DBMS_SQLDIAG
  https://blogs.oracle.com/coretec/post/diagnose-sql-performance-with-dbms-sqldiag

Instance Parameter SESSION_EXIT_ON_PACKAGE_STATE_ERROR

Wenn Datenbank Packages globale Variablen und Konstanten nutzen (deklarieren), dann spricht man von Package State. Wird ein Package – warum auch immer –invalidiert und dann neu recompiliert, werden die Inhalte der Variablen und Konstanten potenziell mit anderen Werten versehen, als in der Session genutzt wurden. Dies kann zu unvorhersagbaren Problemen in der Benutzer-Session führen.

```
set trimspool on
set trim on
set pagesize 0
set linesize 32767
set long 10000000
set longchunksize 10000000

spool my_sqldiag.html replace
SELECT :report report FROM dual;
spool off
```

Listing 9: Ausgabe des CLOBs in ein HTML-File

Der Instance Parameter SESSION_EXIT_ON_PACKAGE_STATE_ERROR = TRUE sorgt dafür, dass solche Sessions automatisch terminiert werden.

Das bisherige Verhalten war, dass in der Session die Fehlermeldung aufgetreten ist, die aber von vielen Applikationen nicht korrekt abgefangen und behandelt wurde *(siehe Listing 11)*.

Hinweis: Diese Funktionalität steht schon seit Oracle 19.23 zur Verfügung.

Referenzen:

- Oracle-Dokumentation zu Package State
  https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/package-state.html#LNPLS-GUID-08E1FC04-9EF3-4396-83C6-4812F8ECABF4
- Oracle 23ai-Funktionalität in Oracle 19c:SESSION_EXIT_ON_PACKAGE_STATE_ERROR
  https://www.database-blog.at/2025/10/08/oracle-23ai-funktionalitaet-in-oracle-19c-session_exit_on_package_state_error/

Automatic SQL Plan Management

Mit Oracle 19.22 hat Oracle das bisher für Exadata Only lizenzierte Automatische SQL Plan Management für alle Oracle-19c-Datenbanken freigegeben und entsprechend zurückportiert.

Das automatische SQL Plan Management erweitert das schon länger verfügbare SQL Plan Management um einige Automatismen, die eine End-zu-End-Lösung für SQL-Statement-Performancedegeneration möglich machen. Wird vom Cost Based Optimizer ein neuer Ausführungsplan erstellt, der sich danach als schlechter erweist, wird automatisch eine SQL Plan Baseline mit dem früheren Ausführungsplan erzeugt.

Dadurch gelingt in vielen Fällen ein schnellerer Rückstieg auf bessere Ausführungspläne, ohne dass jemand eingreifen muss (oder ohne dass ein Evolve Task notwendig wäre).

Wie der Name schon suggeriert: „Automatic" bedeutet, dass hier nichts zu tun ist. Es sollte nur sichergestellt werden, dass Oracle 19.22 oder höher mit einer Oracle Enterprise Edition zum Einsatz kommt. Für die Oracle Standard Edition gibt es diese Funktionalität auf Grund der Beschränkung in Bezug auf SQL Plan Management leider nicht.

Referenzen:

- License Change for Automatic SQL Plan Management
  https://blogs.oracle.com/optimizer/post/license-change-for-auto-spm
- Automatic SQL Plan Management in der Oracle Dokumentation
  https://docs.oracle.com/en/database/oracle/oracle-database/23/tgsql/overview-of-sql-plan-management.html#GUID-44CC1FEB-9259-49FD-82A6-FCE3AEB18A39

Mehrfaktor-Authentisierung (MFA) für Oracle-Datenbanken

Etwas außer Konkurrenz, da es für Oracle 23ai (23.9) und Oracle 19c (19.28) gleichzeitig eingeführt wurde. Oracle bietet nun die Möglichkeit zur Datenbank-Anmeldung mit MFA. Aktuell werden zwei Anbieter von MFA unterstützt: Oracle Mobile Authenticator (OMA) sowie Cisco Duo.

```
SELECT dbms _ sqldiag.report _ sql('gdzcjxv6gzw02') FROM dual;
ERROR:
ORA-14551: cannot perform a DML operation inside a query
ORA-06512: at "SYS.DBMS _ SQLDIAG _ INTERNAL", line 1111
ORA-06512: at "SYS.DBMS _ SQLDIAG _ INTERNAL", line 1651
ORA-06512: at "SYS.DBMS _ SQLDIAG", line 2650
ORA-06512: at line 1
```

Listing 10: Fehlermeldung, wenn man versucht, DBMS_SQLDIAG.REPORT_SQL direkt zu selektieren.

```
ORA-04068: existing state of package has been discarded
```

Listing 11: Fehlermeldung, wenn Package State verloren geht.

```
ALTER USER scott ADD FACTOR 'oma _ push' AS 'scott@example.com';
```

Listing 12: Datenbankbenutzer auf MFA mittels E-Mail umstellen

Da beide auch kostenlose Möglichkeiten anbieten, kann diese Funktionalität ohne Kauf getestet werden. Im Fall vom OMA muss in der Oracle OCI Cloud das Identity Management entsprechend konfiguriert werden.

Die Vorbereitungen zur Nutzung bestehen aus folgenden Schritten:

- Einrichten und Konfigurieren von OCI IAM oder Cisco Duo
- Anpassen der Instance Parameter für die Anbindung des MFA-Anbieters
- Einrichten eines Wallets mit den benötigten Credentials für den MFA-Anbieter (Hinweis: Im Fall von Oracle-19c-Datenbanken wird das Programm orapki der Oracle 23ai Free Datenbank zum Konfigurieren des Wallets benötigt.)
- MFA für den Datenbank-Benutzer konfigurieren *(siehe Listing 12)*.

Hinweis: Für diese E-Mail-Adresse muss im IAM eine Registrierung für die OMA-App vorliegen. Dies erfolgt durch eine E-Mail mit einem QR-Code an die hinterlegte E-Mail-Adresse, der mit der OMA-App eingescannt werden muss.

Bei der Anmeldung mit *sqlplus scott/tiger@mydb* erhält man eine Push Notification auf die OMA-App.

Referenzen:

- MFA Arrives for Local Oracle Database Users
  https://blogs.oracle.com/database/post/new-mfa-jul2025
- Enable MFA for Local User in Oracle Database 23ai using OMA
  https://docs.oracle.com/en/learn/mfa-db23ai-oma/index.html#task-1-oci-identity-and-access-management-configuration

Fazit

Oracle liefert regelmäßig neue Funktionalitäten in Oracle 19c, die teilweise Rückportierungen von neueren Versionen darstellen. Es ist definitiv sinnvoll, von Zeit zu Zeit nachzuschauen, welche Schätze dabei zu finden sind. Neben den hier vorgestellten Funktionalitäten gibt es noch einige andere, auf die ich aus Platzgründen nicht mehr eingegangen bin (zum Beispiel: Re-key von PDBs beim Clonen über Datenbanklinks, DBMS_CRYPTO Erweiterung mit Asymmetric Key Operations, Erweiterungen in der RADIUS Unterstützung und viele andere)

Über den Autor

Christian Pfundtner ist seit über 30 Jahren aktives Mitglied der Oracle-Datenbank-Community. Seine Schwerpunkte liegen auf Hochverfügbarkeit, Performance-Optimierung und Sicherheit. Er war einer der ersten vier Oracle Certified Master in Europa und hat alle Oracle-Datenbankzertifikate (beginnend mit Oracle 7.3) erworben. Seit vielen Jahren ist er Mitglied der Oracle ACE Community als einziger Oracle ACE Director Datenbank in Österreich. Er ist Inhaber und Geschäftsführer der DB Masters GmbH.

Christian Pfundtner
cp@dbmasters.at
www.dbmasters.at
www.database-blog.at

JavaLand

im EUROPA PARK in Rust

10. – 12. MÄRZ 2026

Die Java-Community-Konferenz

Keynote-Speaker

Venkat Subramaniam

Patrick Baumgartner

Bruno Borges

Ed Burns

Hanno Embregts

Milena Fluck

Miriam Greis

Ivar Grimstad

Sophie Küster

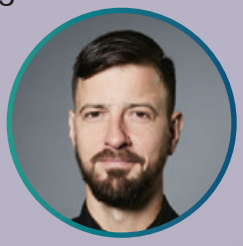
Josh Long

Thomas Much

Nicolai Parlog

Marc Philipp

Mario-Leander Reimer



Henning Schwentner

Marit van Dijk

Freut euch auf diese und viele weitere Speaker sowie jede Menge Wissensaustausch, Networking und noch mehr Highlights.

JAVALAND.EU

Präsentiert von: iJUG Verbund · heise medien · DOAG

Veranstalter: JavaLand

GoldenGate Microservices – Von 0 auf Replikation in 45 Minuten

Amin Farvardin, Hyand Solutions

In modernen IT-Landschaften steigt der Bedarf nach hochverfügbaren, konsistenten und jederzeit zugänglichen Daten exponentiell. Unternehmen stehen vor der Herausforderung, geschäftskritische Systeme nicht nur performant, sondern auch ausfallsicher und flexibel zu betreiben – on-premises, hybrid oder vollständig in der Cloud. Oracle GoldenGate hat sich in den vergangenen Jahren als führende Lösung für Echtzeit-Datenreplikation etabliert. Mit der Einführung der GoldenGate Microservices Architecture (MSA) ist aus einer technisch anspruchsvollen Replikationsplattform ein modularer, cloudfähiger Service geworden, der sich erstaunlich schnell implementieren lässt.

Dieser Beitrag zeigt, warum Microservices der entscheidende Fortschritt gegenüber der klassischen GoldenGate-Architektur sind und wie Unternehmen heute in unter 45 Minuten eine voll funktionsfähige Replikationsstrecke aufbauen können. Dabei vereinen wir praktische Erfahrungen aus Kundenprojekten mit dem Know-how des Hyand-Teams und zeigen, wie Organisationen die Vorteile dieser Architektur nachhaltig nutzen können.

Warum Microservices? Moderne Anforderungen erfordern moderne Architekturen

Traditionelle GoldenGate-Setups basierten auf einer monolithischen Struktur: Extract, Pump, Replicat, Parameterdateien, Konfigurationsdateien und meist komplexe Installationspfade. Die Architektur war leistungsfähig, aber nicht immer intuitiv, insbesondere für heterogene oder dynamische Umgebungen.

Mit der Microservices Architecture verfolgt Oracle einen konsequent modernen Ansatz:

- Serviceorientiert (jeder Bestandteil hat eine klare Aufgabe),
- API-getrieben (REST-Schnittstellen für Automatisierung und DevOps),
- Webbasiert verwaltbar,
- Cloud-ready,
- und besonders wichtig: deutlich leichter zu installieren und zu betreiben.

Die wichtigsten Dienste in MSA:

- Administration Service – zentrale Konfiguration, Benutzer, Zertifikate, Monitoring
- Distribution Service – ersetzt Pump-Prozesse, Datenströme werden als Microservice gesteuert
- Extract/Replicat Services – transaktionssichere Erfassung und Wiedergabe
- Receiver Service – empfängt replizierte Datenströme am Ziel
- Security Service – integrierte Zertifikatsverwaltung und Verschlüsselung

Damit wird GoldenGate zu einem produktionsreifen, modularen Plattformdienst, der sich wesentlich besser in moderne Betriebsmodelle integriert.

Installation/Konfiguration in Minuten statt Stunden

Ein wesentlicher Vorteil von GoldenGate Microservices ist die drastisch vereinfachte Installation. In klassischen Umgebungen musste ein erfahrener DBA mehrere Schritte manuell durchführen.

Mit MSA ist die Installation weitgehend:

- Wizard-gesteuert,
- standardisiert,
- dokumentiert,
- und innerhalb weniger Minuten abgeschlossen.

Besonders auf Engineered Systems oder in der Oracle Cloud (OCI) zeigt die Microservices-Architektur ihre Stärke. Die Services starten einzeln und unabhängig voneinander, was die Fehlersuche erleichtert und Administratoren entlastet.

Von 0 auf Replikation in 45 Minuten – der Ablauf im Überblick

1. Vorbereitung und Installation

GoldenGate Microservices wird auf Quell- und Zielsystem installiert. Der Service Manager dient als zentrale Schaltstelle und ermöglicht den Zugriff auf UI, Logs und Service-Status.

Benutzer, Ports, Trails und Credentials werden über die Weboberfläche angelegt – keine Parameterdateien mehr, kein diffiziles Editieren im Dateisystem.

2. Datenbank-Verbindungen einrichten

Jede Quelle und jedes Ziel wird über den Credential Store registriert. Die verschlüsselte Speicherung der Zugangsdaten erfolgt automatisch. Anders als in der klassischen Architektur sind alle Schritte vollständig über die Weboberfläche oder REST-API steuerbar.

3. Extract-Prozess konfigurieren

Der Extract-Service wird so konfiguriert, dass alle relevanten Änderungen aus der Quelle erfasst werden. Ob LogMiner-basiert, im Integrated-Modus oder mit besonderen Optimierungen – die MSA macht die Konfiguration deutlich transparenter.

Ein Extract lässt sich meist innerhalb von wenigen Sekunden starten.

4. Distribution- und Receiver-Service verbinden

Statt klassischer Pump-Prozesse übernimmt der Distribution Service die Übertragung der Daten – zuverlässig, verschlüsselt und automatisch wiederverbindend. Auf der Zielseite sorgt der Receiver Service für den reibungslosen Empfang der Daten.

Die Vorteile sind erheblich:

- keine manuellen Trail-Verknüpfungen,
- automatische Wiederaufnahme bei Netzwerkunterbrechungen,
- flexible Skalierung,
- saubere Trennung der Verantwortlichkeiten.

5. Replicat einrichten und starten

Replicat übernimmt die Wiedergabe der Daten auf der Zielseite – vollständig transaktionssicher.

Über die UI lassen sich Mappings, Transformationen und Fehlerbehandlungen komfortabel definieren.

Schon wenige Augenblicke nach dem Start fließen die Daten im Zielsystem ein.

Damit ist die Grundkonfiguration der Replikation abgeschlossen – häufig in 30 bis 45 Minuten, je nach Komplexität der Umgebung.

Praxisnutzen: Warum Unternehmen auf Microservices umsteigen

Unternehmen profitieren von GoldenGate Microservices auf mehreren Ebenen:

- Schnellere Implementierung
  Neue Replikationsstrecken lassen sich viel schneller aufbauen – ideal für Migrations- und Modernisierungsprojekte.
- Höhere Betriebssicherheit
  Durch klare Service-Trennung und integriertes Monitoring lassen sich Störungen schneller identifizieren und beheben.
- Ideal für moderne Betriebsmodelle
  MSA ist API-driven – perfekt für DevOps, Automatisierung, CI/CD und Infrastructure-as-Code.

- Cloud- und Hybrid-Fähigkeit
  Ob On-Premises, Engineered System, OCI oder Multi-Region – GoldenGate Microservices funktioniert überall.
- Zukunftssichere Architektur
  Oracle investiert primär in MSA, neue Funktionen entstehen fast ausschließlich dort.

In zahlreichen Kundenprojekten, in denen GoldenGate Microservices für Hochverfügbarkeitslösungen, Rechenzentrumsumzüge oder Zero-Downtime-Migrationen eingesetzt wurde, zeigen sich wiederkehrende Muster und Vorgehensweisen. Aus diesen praktischen Erfahrungen lassen sich einige klare Best Practices ableiten:

- Microservices bevorzugt für neue Installationen einsetzen
- Trails logisch trennen (z. B. je Anwendung oder Schema)
- Datenwege über den Distribution Service standardisieren
- Monitoring über Administration Service und REST-APIs automatisieren
- In RAC-Umgebungen mehrere Receiver Services einplanen
- Replikation mit Data Guard, FSFO oder Observer kombinieren (für Active/Passive-Szenarien)

Diese Empfehlungen haben sich in der Praxis als besonders robust und effizient erwiesen und unterstützen den stabilen Betrieb moderner Replikationsumgebungen.

Fazit

GoldenGate Microservices verändert die Art und Weise, wie Datenreplikation betrieben wird. Was früher ein komplexes Setup mit vielen manuellen Schritten war, lässt sich heute binnen weniger Minuten einrichten – stabil, übersichtlich und sicher.

Für Unternehmen, die auf Echtzeitdaten, hohe Verfügbarkeit und moderne Cloud-Archtitekturen angewiesen sind, bietet MSA enorme Vorteile.

Durch die Kombination aus technischer Klarheit, moderner Architektur und schneller Implementierung empfiehlt sich GoldenGate Microservices nicht nur für neue Projekte, sondern auch als strategischer Ersatz für klassische GoldenGate-Umgebungen.

Hyand Solutions GmbH unterstützt Unternehmen dabei, diese Potenziale vollständig auszuschöpfen – von der Architektur über die Installation bis hin zum Betrieb und der Automatisierung.

Quellen

[1] Oracle Corporation (2023): Oracle GoldenGate Microservices Architecture – Concepts Guide. Oracle Press, Redwood Shores.
[2] Oracle Corporation (2023): Oracle GoldenGate Installation and Upgrade Guide for Microservices. Oracle Press, Redwood Shores.
[3] Oracle Corporation (2022): Oracle GoldenGate – High Availability and Disaster Recovery Best Practices. Oracle Press, Austin.
[4] Oracle Corporation (2021): Zero Downtime Migration Using Oracle GoldenGate. Oracle Press, Redwood Shores.
[5] Oracle Corporation (2023): Oracle GoldenGate REST API Reference Guide. Oracle Press, Redwood Shores.

Über den Autor

Ich bin Senior Datenbank- und Infrastruktur-Spezialist mit langjähriger Erfahrung in Oracle-Technologien, Hochverfügbarkeit und Replikationslösungen. Meine Schwerpunkte liegen auf Oracle-Datenbanken, Engineered Systems, RAC, GoldenGate, Data Guard, Backup/Recovery und Cloud-Migrationsprojekten. Bei Hyand unterstützen wir Unternehmen dabei, komplexe Datenbankumgebungen stabil, modern und zukunftssicher zu gestalten.

Amin Farvardin
amin.farvardin@hyand.com

RMAN aufgeräumt: Standardisierte und automatisierte Backups ohne Skript-Chaos

Dr. Thomas Petrik, Sphinx IT Consulting

Routinearbeit raus, Standards rein: RMAN kann weit mehr als nur Backups – wenn man ihn richtig einbindet. Automatisierung und Standardisierung sind die Schlüssel für einen stabilen Betrieb – das gilt auch für RMAN-Backups. Ein generischer Kern wird lediglich durch das DB-Environment individualisiert – manuelle RMAN-Skripte gibt es nicht. Sie müssen dafür auch kein Cloud Control installieren, das wäre in den meisten SE2-Umgebungen wohl ein kostspieliger Overhead. Auch ein übersichtliches und verständliches Reporting gehört nicht zu den Stärken des RMAN, ist aber gerade in kritischen Situationen unerlässlich. Zudem stellen wir die klassischen RMAN-Backup-Strategien in den Kontext mit kompletten VM-Backups durch Storage Snapshots: Brauchen wir den RMAN da überhaupt noch und wenn ja, wie müssen wir die Strategien anpassen, und können wir sie sogar vereinfachen?

```
…
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 30 DAYS;
CONFIGURE BACKUP OPTIMIZATION ON;
CONFIGURE DEFAULT DEVICE TYPE TO DISK;
CONFIGURE CONTROLFILE AUTOBACKUP ON;
…
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/app/oracle/backup/%d/%F';
CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO BACKUPSET PARALLELISM 4;
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT   '/app/oracle/backup/%d/%s_%p_%T_%u';
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/app/oracle/data/C000D_A/snapcf.f';
```

Listing 1: Typische RMAN-Konfiguration einer CDB für Backups in 4 parallelen Streams auf ein Disk Share

```
-- maintenance
crosscheck archivelog like '/app/oracle/fra/%' device type disk;
delete expired archivelog like '/app/oracle/fra/%';
crosscheck copy of database like '/app/oracle/fra/%' device type disk;
crosscheck copy of controlfile like '/app/oracle/fra/%' device type disk;
delete expired copy like '/app/oracle/fra/%';
catalog recovery area noprompt;
crosscheck foreign archivelog like '/app/oracle/fra/%' device type disk;
delete expired foreign archivelog like '/app/oracle/fra/%';
crosscheck backup device type disk;
delete expired backup;
delete obsolete;
-- backup
backup incremental level 0 device type disk database
include current controlfile
plus archivelog ;
```

Listing 2: Generisches RMAN-Skript für Maintenance und Differential Level 0 Backup

```
$ rman -h
USAGE:
       run a backup (optional maintenance)
         rman [-D db_name] [-Q] -b {full|cum0|cum1|diff0|diff1}
              [-d dest] [-m] [-t] {RMAN-options}
       run an archivelog backup (optional maintenance)
         rman [-D db_name] [-Q] -a [-d dest] [-m] [-t] {RMAN-options}
       run maintenance
         rman [-D db_name] [-Q] [-t] -m {RMAN-options}
       list backup jobs
         rman [-D db_name] -j [days]
       list backup summary by tags
         rman [-D db_name] -s [days]
       interactive command
         rman [-D db_name] {RMAN-options}
       dest(ination):
       TAPE ............... backup to tape
       FRA ................ backup to flash recovery area
                            (valid for -b only)
       FRA_TAPE .......... backup to flash recovery area and migrate to tape
                            (valid for -b only)
       DISK:<filename> ... backup to disk using <filename> (any format string)
```

Listing 3: Beispiel für die Syntax eines generischen RMAN-Skripts

```
$ crontab -l
0 1 * * 0   /app/oracle/local/bin/rman -D C000D -b diff0 -m -Q
0 1 * * 1-6 /app/oracle/local/bin/rman -D C000D -b diff1 -m -Q
```

Listing 4: Beispiel für die einfache Einrichtung der Backup via Cron

RMAN & Storage Snapshots

Storage Snapshots können auf unterschiedlichen Ebenen gezogen werden:

- für eine einzelne VM
- auf Hypervisor-Ebene für mehrere VMs (speziell, wenn ZFS oder Ceph als Storage Layer verwendet werden)
- im Storage-System für einzelne LUNs oder eine Gruppe von LUNs

Allen Varianten ist eines gemeinsam: sie stellen eine Momentaufnahme dar und sind crash-consistent. Das ist also so, als würden Sie den Stecker ziehen.

Die Granularität des Snapshots entscheidet über die Konsistenz des Systems im Fall einer Wiederherstellung: Wird der Snapshot für eine Gruppe von VMs gezogen, sind diese nach einem Re-Apply des Snapshots zueinander konsistent – etwas, was mit klassischen Backups auf VM-Ebene nie erreicht werden kann.

Der klassische Einwand – „Snapshots are not Backups" – stimmt natürlich grundsätzlich, daher ist es unerlässlich, die Snapshots auf eine remote Destination zu transferieren. Passiert dies verschlüsselt und wird aus diesen Snapshots eine Blockchain gebildet (das heißt, Koststenz und Unveränderlichkeit sind implizit gegeben), kann man so damit sogar Ransomware-resistente Systeme bauen. Ein Beispiel dafür stellt die Blueboxx der Sphinx IT dar *(siehe Abbildung 1)*.

Braucht man den RMAN dann überhaupt noch? Die Antwort ist klar: Dort, wo RMAN für online Backups bisher im Einsatz war, ist er auch in Zukunft unverzichtbar. Ein Point-in-Time Recovery oder ein DB-Cloning kann mit Snapshots allein nicht funktionieren.

Doch sehen wir uns die Szenarien genauer an:

a) NOARCHIVELOG Mode:
   Hier kommt der RMAN natürlich nicht zum Einsatz für Online-Backups. Regelmäßige Snapshots aber sind von großem Nutzen, weil dadurch zumindest ein Rückstieg auf die Backup-Zeitpunkte möglich wird – also eine klare Verbesserung der Situation.
b) RMAN-Backups in die Flash Recovery Area (FRA):
   Dieses Szenario wird erst durch die Kombination mit Snapshots (inklusive remote Shipping) zu einem vollwertigen Backup. Der Vorteil: es ist sehr performant (speziell im Restore-Fall) und die Backup-Historie kann kurzgehalten werden – sie muss im Grunde nur den Zeitraum zwischen zwei Snapshots abdecken.
   Aber kein Vorteil ohne Nachteil: Durch die lokalen Backups steigt die Zahl der geänderten Blöcke und somit die Größe der Snapshots (die inkrementell sind).
   Dieses Szenario eignet sich also nur für kleinere Systeme mit mäßiger Transaktionslast.
c) RMAN-Backups auf ein Share oder Tape:
   Der Klassiker – belastet die Snapshots nicht, benötigt aber längere Historien und ist abhängig von Bandbreiten und I/O-Performance des Backup-Stores.

RMAN-Automation ohne Cloud Control

Die Zeiten, in denen das Verhalten hinsichtlich Parallelisierung („allocate channel …") oder Destination (Disk, Tape, …) direkt im RMAN-Script codiert werden musste, sind längst vorbei. All diese Parameter können pro CDB individuell mit den RMAN „configure"-Commands hinterlegt werden (sinnvollerweise gleich im Zuge der Provisionierung einer neuen CDB). *Listing 1* zeigt ein Beispiel dafür.

Für das eigentliche RMAN-Script verbleibt eine Hand voll Standardkommandos, die vollkommen DB-unabhängig sind. Ein Beispiel für ein Differential Level 0 Backup mit vorgelagerter Maintenance (das heißt Löschen alter Backups, Prüfung auf fehlende Backups, Katalogisierung allenfalls unbekannter Backups und vieles mehr) findet sich in *Listing 2*.

Der einzig spezifische Teil in diesem Beispiel bleibt der Pfad zur FRA und dieser kann durch ein generisches Skript direkt aus der DB ausgelesen werden (`select db_recovery_file_dest from v$parameter`).

Ansonsten wird das generische RMAN-Skript durch klassische Commandline-Parameter und allenfalls noch durch Environment-Variablen, die wiederum spezifisch pro CDB im jeweiligen CDB-Environment gesetzt sein können, gesteuert. *Listing 3* gibt ein Beispiel für die Syntax eines derartigen Skripts.

Damit reduziert sich die Einrichtung der erforderlichen Backups auf 2 Zeilen in der Crontab *(siehe Listing 4)*.

Da sich mit jedem Major Oracle Release die RMAN-Syntax leicht ändern kann, sollte das Skript natürlich mit unterschiedlichen Versionen umgehen können.

RMAN Security

Passwörter in RMAN-Skripts – das muss wirklich nicht sein. RMAN benötigt üb-

```
mkstore -wrl $TNS_ADMIN -createCredential rman rman <password> -nologo
```

Listing 5: Hinterlegen der RMAN Catalog Credentials im Wallet

```
WALLET_LOCATION = (SOURCE =
  (METHOD = FILE)
  (METHOD_DATA = (DIRECTORY = /app/oracle/admin/network)))
SQLNET.WALLET_OVERRIDE = TRUE
```

Listing 6: Notwendige Einträge in sqlnet.ora

```
SELECT oldest_backup_time
      ,newest_backup_time
      ,num_backupsets
      ,ROUND (output_bytes / 1024 / 1024 / 1024, 3)     AS size_gb
  FROM v$backup_set_summary;
SELECT start_time
      ,end_time
      ,status
      ,input_type                                       AS TYPE
      ,ROUND (elapsed_seconds / 60, 2)                  AS elapsed_min
      ,ROUND (output_bytes / 1024 / 1024 / 1024, 3)     AS size_gb
      ,ROUND (compression_ratio, 1)                     AS compression_ratio
  FROM (SELECT start_time
              ,end_time
              ,status
              ,input_type
              ,(end_time - start_time) * 86400     AS elapsed_seconds
              ,output_bytes
              ,compression_ratio
          FROM (  SELECT start_time
                        ,end_time
                        ,status
                        ,input_type
                        ,(end_time - start_time) * 86400     AS elapsed_seconds
                        ,output_bytes
                        ,compression_ratio
                    FROM v$rman_backup_subjob_details
                   WHERE operation = 'BACKUP'
                ORDER BY end_time DESC))
```

Listing 7: Ein Schneller Überblick über die letzten RMAN-Jobs

```
…
with base as (
SELECT r.start_time
      ,r.end_time
      ,r.input_type    AS TYPE
      ,CASE
           WHEN s.backup_type = 'L' THEN 'ARCHIVELOG'
           WHEN s.controlfile_included = 'YES' THEN
                CASE WHEN sp.set_stamp IS NULL THEN 'CONTROLFILE'
                     ELSE 'SPFILE' END
           WHEN s.backup_type = 'D' THEN 'DB FULL ' || NVL (s.incremental_level, 0)
           WHEN s.backup_type = 'I' THEN 'DB INCR ' || s.incremental_level
           ELSE s.backup_type
       END             AS subtype
      ,s.completion_time
      ,p.tag
      ,p.bytes
      ,s.elapsed_seconds
  FROM v$rman_backup_subjob_details  r
       JOIN v$backup_set_details s ON r.session_stamp = s.session_stamp
       JOIN v$backup_piece_details p
           ON p.session_stamp = r.session_stamp
              AND p.set_stamp = s.set_stamp
              AND p.set_count = s.set_count
       LEFT JOIN v$backup_spfile_details sp
           ON sp.session_stamp = r.session_stamp
              AND sp.set_stamp = s.set_stamp
              AND sp.set_count = s.set_count
 WHERE r.operation = 'BACKUP' AND p.status = 'A'
)
…
```

Listing 8: Die ausgeführten Backups nach Typ im Detail (Auszug)

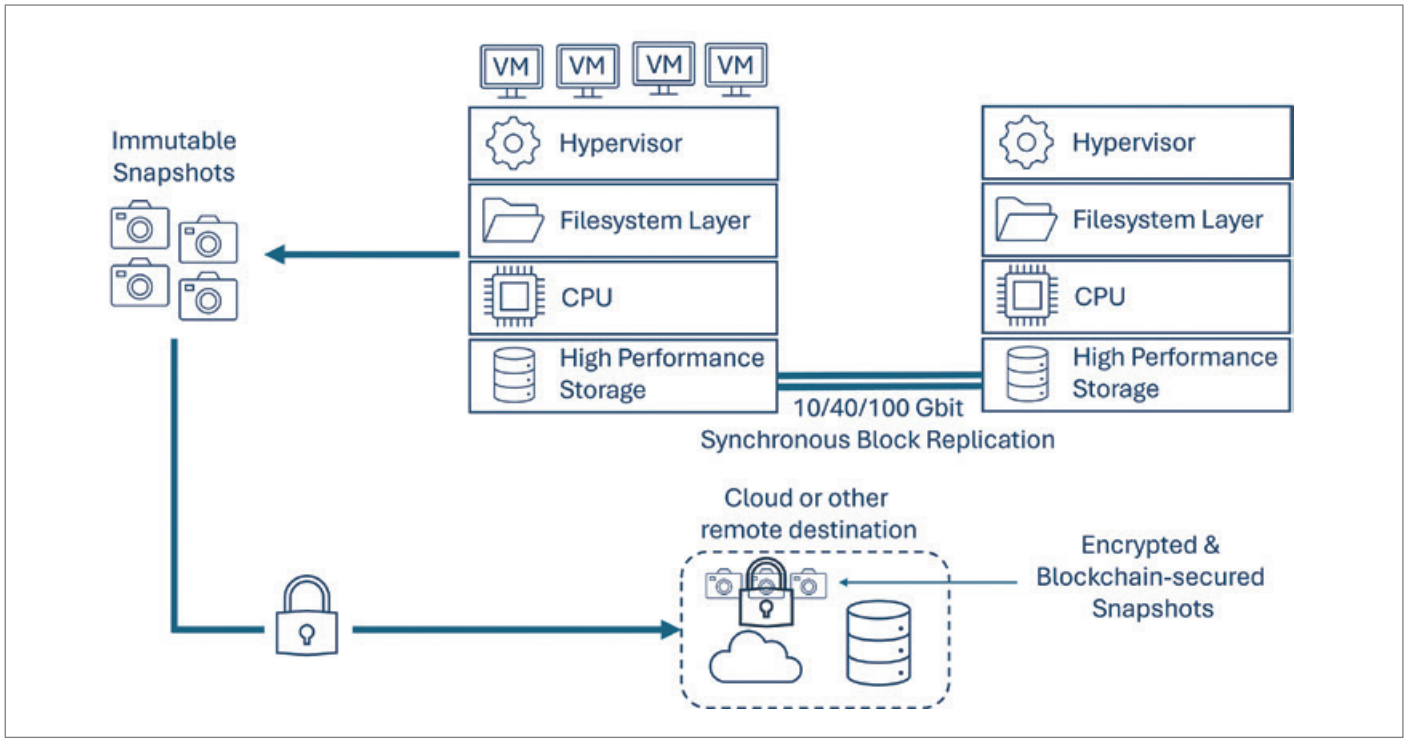


Abbildung 1: Blueboxx (Näheres unter www.sphinx.at/blueboxx) (Quelle: Thomas Petrik)

Logtime	FRA used %	FRA recl.%	FRA avail.%	Archivelogs not saved %	estimated overflow	Backup needed
18.11.2025 08:40	79,78	52,92	73,14	6,86	2025-11-19 16:42	N
18.11.2025 08:41	79,78	52,92	73,14	6,86	2025-11-19 09:47	N
18.11.2025 08:42	79,2	49,78	70,58	9,42	2025-11-19 04:28	N
18.11.2025 08:43	79,2	49,78	70,58	9,41	2025-11-19 01:05	N
18.11.2025 08:44	78,08	46,68	68,6	11,4	2025-11-18 22:15	N
18.11.2025 08:45	78,99	46,68	67,69	12,31	2025-11-18 20:08	N
18.11.2025 08:46	78,99	46,68	67,69	12,31	2025-11-18 18:38	N
18.11.2025 08:47	76,03	43,71	67,68	14,43	2025-11-18 17:31	N
18.11.2025 08:48	78,14	43,71	65,57	14,43	2025-11-18 16:28	N
18.11.2025 08:49	93,77	52,45	58,68	17,32	2025-11-18 15:13	N
18.11.2025 08:50	97,08	43,69	46,61	23,39	2025-11-18 13:49	N
18.11.2025 08:51	93,59	39,31	45,72	32,37	2025-11-18 12:53	N
18.11.2025 08:52	95,6	23,23	27,63	32,37	2025-11-18 11:55	N
18.11.2025 08:53	90,59	11,55	20,96	32,37	2025-11-18 11:14	N
18.11.2025 08:54	90,59	11,55	20,96	32,37	2025-11-18 10:47	N
18.11.2025 08:55	89,37	4,23	14,86	38,47	2025-11-18 10:26	N
18.11.2025 08:56	89,37	4,23	14,86	38,47	2025-11-18 10:14	N
18.11.2025 08:57	89,37	4,23	14,86	38,47	2025-11-18 10:00	N
18.11.2025 08:58	89,37	4,23	14,86	38,47	2025-11-18 09:51	Y
18.11.2025 08:59	87,33	0,00	12,67	40,66	2025-11-18 09:44	R
18.11.2025 09:00	87,33	0,00	12,67	40,66	2025-11-18 09:39	R
18.11.2025 09:01	87,33	0,00	12,67	40,66	2025-11-18 09:34	R
18.11.2025 09:02	87,33	40,66	53,33	0,00		N
18.11.2025 09:03	87,33	40,66	53,33	0		N

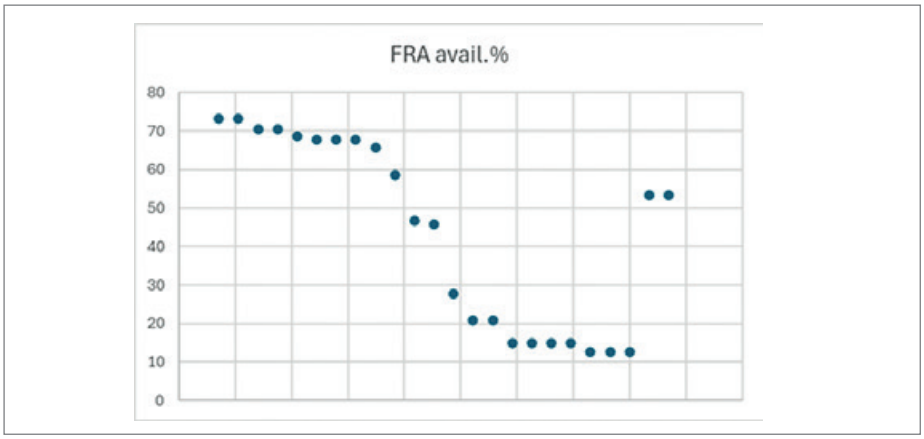


Abbildung 2: Monitoring der FRA und Regression zur Abschätzung des Overflows (Quelle: Thomas Petrik)

licherweise zwei Connections: Zur CDB selbst und zum RMAN-Katalog (falls ein solcher verwendet wird). In unserem Konzept läuft das generische RMAN-Skript im lokalen Datenbank-Environment – ORACLE_HOME und ORACLE_SID sind also gesetzt, ein Bequeath-Connect („/ as sysdba“) ist also vollkommen ausreichend.

Für den Connect zum Catalog hinterlegen wir die Credentials in einem Oracle Wallet.

Listing 5 zeigt exemplarisch die Anlage eines derartigen Wallet-Eintrags, *Listing 6* die notwendigen Parameter in sqlnet.ora.

Der RMAN-Aufruf reduziert sich so – ganz ohne explizite Angabe eines Passworts – auf folgende Zeile:

```
rman target / catalog /@rman
```

RMAN Reporting

Ein schneller Überblick auf Knopfdruck, gerade in hektischen Situationen, erleichtert das Leben des DBA massiv. RMAN selbst bietet da leider nahezu nichts. Aber es wäre nicht Oracle, wenn nicht ohnehin alle Informationen im Dictionary zu finden wären. Aus V$RMAN- und V$BACKUP-Views lässt sich in jedem Fall das passende Reporting bauen.

Listing 7 und *Listing 8* zeigen zumindest auszugsweise, wie man einen schnellen Überblick über die RMAN-Jobs und deren Inhalt erhält: Auch diese Funktionalität sollte in das generische RMAN-Skript integriert werden, sodass – wie in unserem Beispiel – ein simples `rman -j` oder `rman -s` die gewünschte Information übersichtlich aufbereitet liefert.

Hinweis:

Achten Sie darauf, dass die control_file_record_keep_time stets größer ist als die RMAN Retention Policy, sofern Sie keinen Catalog verwenden.

Automatisierte Archivelog Backups

Leider ein oft beobachtetes Problem: Die Datenbank steht, weil die FRA voll ist. Wie kann das passieren?

Es ist nicht notwendig (und für Restore-Szenarien sogar kontraproduktiv), Archivelogs zu löschen – Oracle kümmert sich bereits seit der Version 10 selbst darum. Dies trifft allerdings nur auf Archivelogs zu, die bereits gesichert wurden und somit als RECLAIMABLE gelten (siehe V$FLASH_RECOVERY_AREA_USAGE).

Es ist daher unerlässlich, den effektiven Füllgrad (also jenen Anteil an Files, die nicht gelöscht werden können – dazu zählen beispielsweise auch Flashbacklogs in einer Enterprise Edition) regelmäßig abzufragen, um aus diesem Verlauf mittels linearer Regression einen möglichen Overflow abschätzen zu können. Ein Archivelog Backup kann dann automatisch getriggert werden, wenn der Overflow absehbar ist (zum Beispiel laut Prognose in den nächsten 2 Stunden) oder wenn ein bestimmter Grenzwert an verfügbarem Platz (zum Beispiel 10%) unterschritten wird.

Abbildung 2 zeigt exemplarisch einen typischen Verlauf. In der Spalte „Backup Needed“ sieht man, wie das Flag von (N)o auf (Y)es und dann auf (R)unning umspringt. Tatsächlich wurde über den DBMS_SCHEDULER ein External Job gestartet, der das Kommando `rman -a` aufruft (eine weitere Spezifikation des Environments ist hier gar nicht notwendig, da der Prozess ohnehin im Kontext der CDB abläuft).

Für die Statistiker unter Ihnen sei hinzugefügt, dass das Quadrat des Korrelationskoeffizienten (r^2) für eine verlässliche Prognose größer als 0,7 sein sollte.

Fazit

Eine Automatisierung in der beschriebenen Art und Weise ist natürlich ein initialer Aufwand, der sich aber bereits ab der dritten Installation zu rentieren beginnt. Speziell im Fall vieler kleiner Kundenumgebungen, die aus rechtlichen Gründen gar nicht in einer Cloud-Control-Instanz vereint werden dürfen, ist diese Vorgehensweise der Schlüssel zu einer automatisierten und verlässlichen Betriebsführung.

Über den Autor

Dr. Thomas Petrik arbeitet seit fast 3 Jahrzehnten mit Oracle-Datenbanken und befasst sich ebenso lange mit den Themen Security, Performance und Effizienz in der Betriebsführung (Automatisierung). Es ist kein Zufall, dass aus dieser Tätigkeit im Laufe der Zeit Services und Produkte entstanden sind, die aus der Praxis für die Praxis geschaffen wurden und erfolgreich von den Kunden der Sphinx IT Consulting eingesetzt werden.

Dr. Thomas Petrik
thomas.petrik@sphinx.at
www.sphinx.at

DOAG Datenbank Kolumne: Im Alltag immer etwas dazulernen

Dagmar Förster

Die Arbeitstage sind meist gut angefüllt. Dennoch versuche ich, mich im Alltag weiterzubilden und dazuzulernen.

Meine ganz persönlichen Vorschläge (können Spuren von Werbung enthalten):

Die Blogs von Mike Dietrich und Daniel Overby Hansen lesen. Als Abonnentin bekomme ich ungefähr je eine Mail pro Woche mit wertvollen Tipps, mal bleiben sie im Hinterkopf, mal stelle ich gleich meine Befehle um wegen der Hidden Column cdb$name in den CDB-Views.

Fehlermeldungen als Ansporn. Ich kann den neuen Service nicht anlegen? Fehler sagt etwas mit "serverpool". Was ist nochmal ein Serverpool? Und wenn ich schon dabei bin, schaue ich mir gleich die Clusterware an.

Das "Oracle Database Monthly News Team" veröffentlicht monatlich Neuigkeiten zu DB, Releases, Veranstaltungen und die Quick Links Postings auf ihrem Database & Cloud Technology Blog.

Bash-Skripte schreiben (mit viel Googeln wegen der oft gruseligen Syntax) und damit tägliche Aufgaben automatisieren, standardisieren und vereinfachen und nebenbei Logik üben.

An einstündigen, kostenlosen DOAG Datenbank DBTalks und Sessions teilnehmen, z.B. am 25.02.26 zum Thema „PostgreSQL hochverfügbar betreiben", immer um 17 Uhr nach der Hauptbürozeit.

Die DOAG Datenbank Kolumnen lesen, vielleicht den Autor anschreiben und um die Skripte bitten.

Mit Kolleginnen und Kollegen diskutieren, gerne aus anderen Abteilungen. Ach, Ihr benutzt jetzt DBeaver, zeig mal. Wie war nochmal der Befehl, um nachzuschauen, welches Linuxpaket installiert ist?

Ab und zu DBSAT (Doc ID 2138254.1 in MOS) und Exachk / ORAchk ausführen. Daraus ergeben sich oft weitere Fragestellungen.

Ein bisschen Zeit findet sich immer, wenn nicht diese Woche, dann in der nächsten. Mir macht die Arbeit so mehr Spaß und die kleinen Häppchen Wissen zahlen sich früher oder später aus.

Mehr Cloud für weniger Geld: Ein praxisnaher Leitfaden zur Kostenoptimierung in der Oracle Cloud Infrastructure

Marcus Schröder, Oracle Deutschland

Mehr Cloud für weniger Geld zu erreichen, erfordert in der Oracle Cloud Infrastructure (OCI) eine Kombination aus Transparenz, Governance und kontinuierlicher Optimierung. Cloud-Nutzung ist ein strategischer Hebel für Innovation und Geschwindigkeit, doch ohne klare Verantwortlichkeiten und datengestützte Entscheidungen steigen die Ausgaben schneller als der Nutzen. Kostenoptimierung in der OCI ist deshalb ein wiederkehrender Prozess, der Technik, Finanzen und Produktstrategie verbindet, FinOps verankert und Automatisierung gezielt einsetzt. Ziel ist es, Kapazitäten präzise am Bedarf auszurichten, Risiken zu reduzieren, Budgetdisziplin zu stärken und gleichzeitig Innovation zu fördern.

Grundlagen der Kostenoptimierung in OCI

Der Ausgangspunkt ist ein tiefes Verständnis der Workloads. Wer Lastspitzen, Leerlaufphasen und Wachstumsmuster kennt, dimensioniert Ressourcen passgenau, vermeidet Überprovisionierung und definiert wirkungsvolle Skalierungsregeln. Auf Basis systematischer Analysen zur Ressourcennutzung werden Rightsizing-Entscheidungen getroffen und Auto-Scaling so gestaltet, dass Performance-Ziele und Kostengrenzen gleichermaßen eingehalten werden. Kontinuierliches Performance-Monitoring sorgt dafür, dass die Infrastruktur laufend an neue Anforderungen angepasst wird – technisch und wirtschaftlich.

Einführung in FinOps

Neben der Technik bildet FinOps den zweiten Grundpfeiler. Der FinOps-Zyklus aus Informieren, Optimieren und kontinuierlicher Steuerung verankert Transparenz und wirtschaftliche Entscheidungsfindung. Die Rollen sind klar verteilt: Ingenieurwesen verantwortet kosteneffiziente Architekturen und die operative Optimierung, FinOps-Praktiker etablieren Reporting, Tags, Budgets, Alerts und Governance, und Produkt-Owner richten Ausgaben an Roadmap, SLAs und Business-Metriken aus. Dieses Zusammenspiel verhindert einseitige Entscheidungen, hält Kosten, Geschwindigkeit und Qualität in Balance und schafft die Grundlage für eine nachhaltige Kostenkultur.

OCI-Tools für Kostenmanagement – Transparenz als Voraussetzung

Transparenz liefert OCI durch zentrale Werkzeuge: Kostenanalysen und Nutzungsberichte machen Treiber und Trends sichtbar, Budgets mit Alerts schaffen finanzielle Leitplanken und Frühwarnsysteme, und einheitliches Tagging ermöglicht eine verursachungsgerechte Allokation bis auf Produkt-, Team- oder Umgebungs-Ebene. Die Kombination aus Analyse, Budgets und Tags macht Kosten sichtbar, antizipierbar und steuerbar – ohne Innovation auszubremsen. Empfehlenswert ist, in neuen Tenants auf Root-Compartment-Ebene früh Default-Tags wie „Erstellungsdatum“ und „aktueller Benutzer“ zu aktivieren, um Ressourcen eindeutig den Besitzern zuordnen zu können *(siehe Abbildung 1)*.

Compute-Ressourcen

In der Praxis beginnt die Optimierung häufig bei Compute *(siehe Abbildung 2)*.Rightsizing auf Basis realer Auslastungsdaten stellt sicher, dass CPU, RAM, IOPS und Netzwerkbandbreite den Bedarf decken, ohne dauerhaft darüber zu liegen. Flexible Compute-Shapes unterstützen eine fein abgestimmte Anpassung von OCPU/ECPU und Arbeitsspeicher und reduzieren Overprovisioning. Auto-Scaling passt Kapazitäten dynamisch an, idealerweise metrisch gesteuert (etwa durch CPU-Auslastung, Queue-Länge oder Latenz), mit Cooldown-Perioden gegen Flapping, klaren Ressourcenlimits zum Schutz des Budgets und integrierten Load Balancern für gleichmäßige Auslastung. Günstige Preemptible Instances eignen sich als starker Hebel für fehlertolerante, nichtkritische Workloads wie Batch-Verarbeitung, CI/CD, Entwicklung/Test, stateless Webanwendungen oder ML-Training – vorausgesetzt, Checkpointing, Wiederanlaufstrategien, Monitoring und automatisierte Neuplatzierung sind etabliert.

Storage- und Netzwerk-Optimierung

Storage ist oft ein schleichender Kostentreiber, weil er dauerhaft Kosten verursacht. Wirksam sind die bewusste Wahl der Speicherklassen entlang des Datenlebenszyklus – häufig genutzte Daten in Object Storage, selten benötigte oder revisionsrelevante Daten im Archive Storage – sowie Lifecycle-Policies, die Daten nach Kriterien wie Alter, Zugriffshäufigkeit oder Klassifizierung automatisch verschieben, komprimieren oder löschen. Das senkt Kosten, verbessert Betriebseffizienz und unterstützt Datenhygiene, Compliance und Backups. Auch im Netzwerk lassen sich Kosten senken, insbesondere bei Egress und interregionalen Transfers. Kostenfallen werden durch optimierte Datenlokalität, CDN- und Caching-Strategien, Komprimierung und Datenaggregation adressiert. Hilfreich sind Monitoring und Analyse von Traffic-Strömen und Hotspots; aktuell ist die SKU für Netzwerk-Traffic B88327. Zudem sollten Richtlinien für Datenübertragungen zwischen Regionen/Compartments hinterfragt werden und Caching die wiederholte Abfrage entlasten.

Implementierung von Tagging und Budgetierung

Tagging und Budgetierung bilden die Governance-Schicht über allen Maßnahmen. Standardisierte Pflicht-Tags wie cost_center, environment, product und owner schaffen Verantwortlichkeit und saubere Allokation; Kostentracking-Tags verbinden operative Nutzung mit Finanzsicht. Budgets mit Benachrichtigungen etablieren ein Frühwarnsystem und ermöglichen proaktives Handeln. Kontinuierliches Monitoring von Infrastruktur-, Anwendungs- und Wirtschaftsmetriken – von CPU und Latenz über Fehlerraten und SLO-Erfüllung bis zu Kosten pro Transaktion oder Kunde – verhindert lokale Optima und liefert die Grundlage für Right- und Auto-Scaling. Automatisierung vom Provisioning bis zum Lifecycle-Management standardisiert Best Practices, verringert Fehlerquellen und erhöht die Reaktionsgeschwindigkeit bei Last- und Kostenabweichungen.

Von der Theorie zur Praxis – ein FinOps-Betriebsmodell

Ein wirksames FinOps-Betriebsmodell verbindet Inventarisierung, Standardisierung, Instrumentierung, Optimierung, Operationalisierung sowie Messen und Lernen. Zuerst wird vollständige Sicht auf Tenants, Compartments, Ressourcen, Tags und Kostenströme hergestellt. Darauf folgen Tagging-Konventionen, Budgetregeln, Alert-Schwellen, Namenskonventionen und Richtlinien. Mit aktivierten Kosten- und Nutzungsberichten, produktionsreifen Alerts und Team-Dashboards wird die Steuerung operationalisiert. Im nächsten Schritt werden Rightsizing, Auto-Scaling, Preemptible-Workloads, Speicherklassen

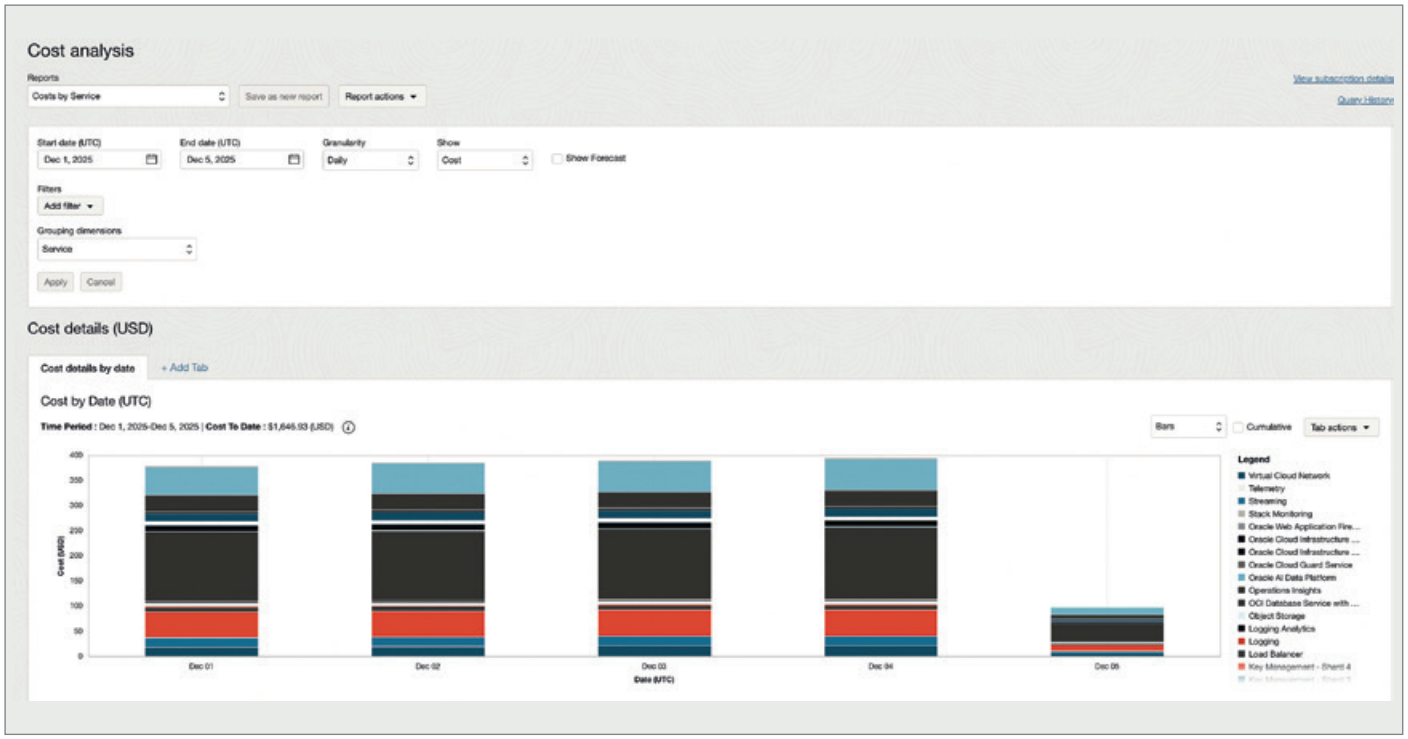


Abbildung 1: OCI Cost Analyzer (Quelle: Marcus Schröder)

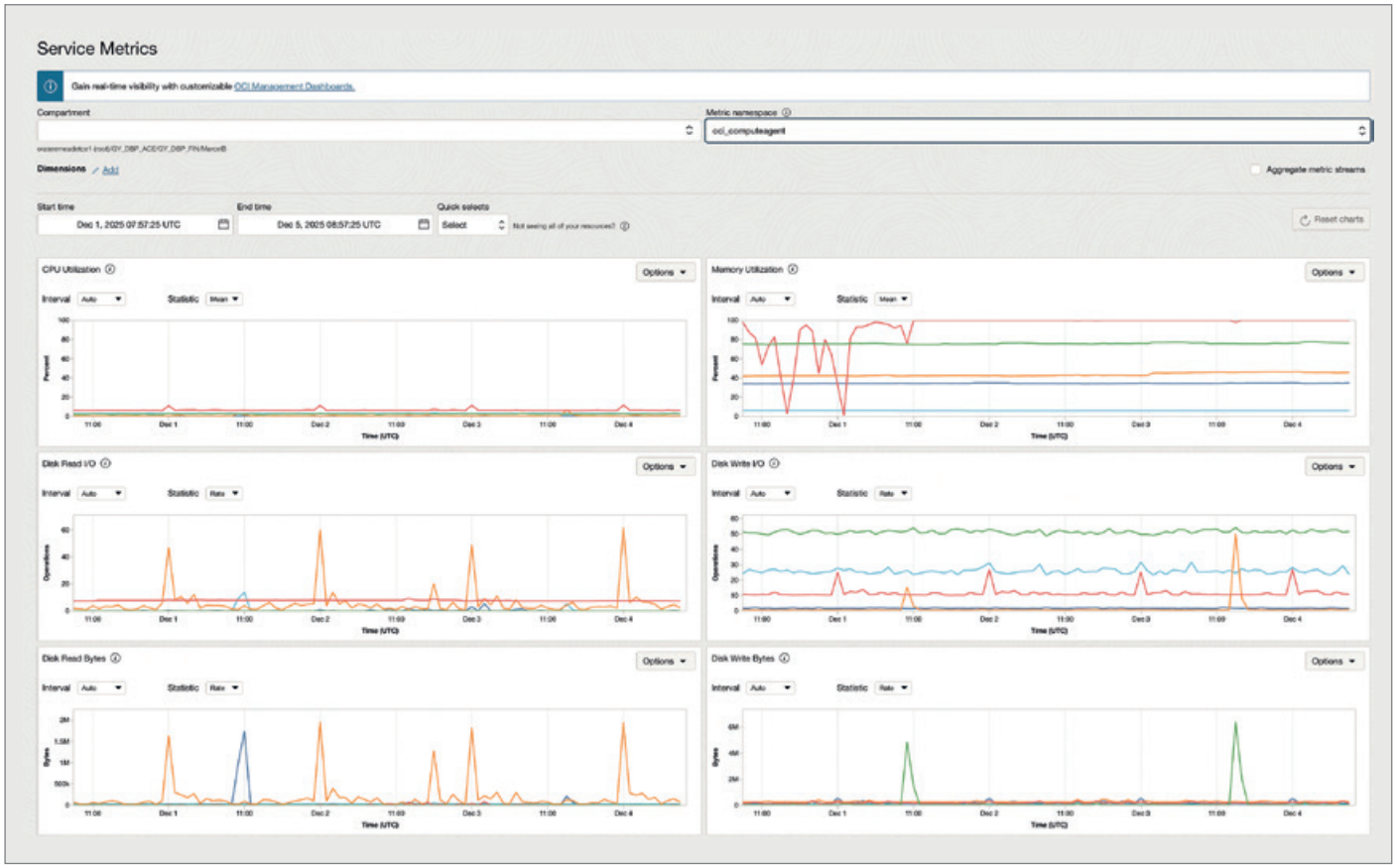


Abbildung 2: Service-Metriken von Compute-Instanzen (Quelle: Marcus Schröder)

und Lifecycle-Policies umgesetzt sowie die Netzwerkarchitektur auf Egress-Kosten geprüft. Regelmäßige Reviews, SLO- und Budget-Drilldowns, Postmortems bei Budgetverletzungen und eine quartalsweise Roadmap halten den Verbesserungsprozess in Gang; klar definierte Erfolgsmetriken – etwa minus 30 Prozent Leerlaufkosten, plus 20 Prozent Auslastung oder weniger als zwei Prozent Budgetabweichung – machen Fortschritt messbar.

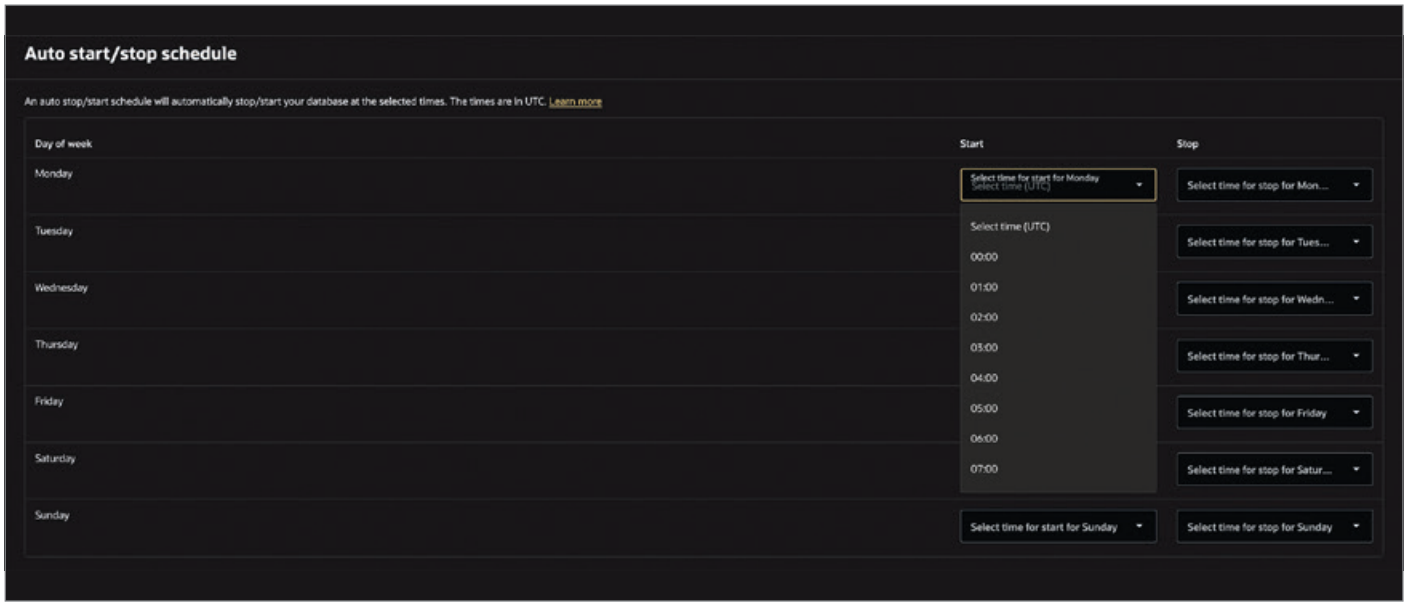


Abbildung 3: Automatisches Herunterfahren von Autonomous AI Database (Quelle: Marcus Schröder)

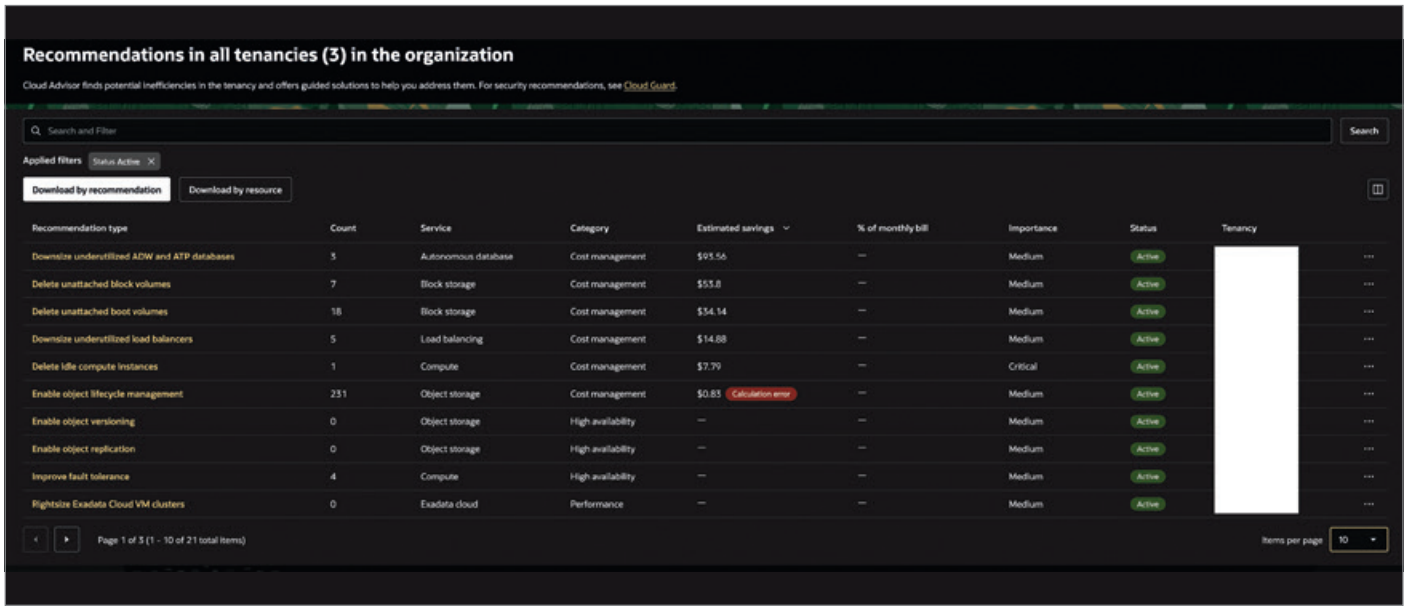


Abbildung 4: Cloud-Advisor-Vorschläge zu Kosteneinsparungen (Quelle: Marcus Schröder)

Fallbeispiele, Ergebnisse und Stolpersteine

Persönliche Erfahrungen zeigen die Skalierbarkeit strukturierter FinOps-Programme: Neue interne Tenants mit Budget-Belegung über jeweils mehr als 100 Accounts wurden erfolgreich eingeführt; in einem bestehenden Tenant wurden über 50 Accounts nachgerüstet – mit den erwartbaren Hürden, wenn FinOps-Prinzipien zuvor nicht konsequent etabliert waren. Die Lehren daraus: Ohne klare Standards, rechtzeitige Berichte und Automatisierung drohen Blindflüge, Überdimensionierung und unkontrollierte „Zombie"-Ressourcen. Gegenmaßnahmen sind verbindliche Tagging-Standards, hohe Automatisierung, klare Verantwortlichkeit und eine Kultur, die Kosten- und Leistungsziele gleichwertig behandelt.

Detaillierte Handlungsanleitung nach Domänen

Konkrete Handlungsfelder nach Domänen setzen auf belastbare Datengrundlagen, Perzentil-basierte Dimensionierung, metrische Auto-Scaling-Policies mit Cooldowns und Limits sowie den gezielten Einsatz von Preemptible-Instanzen für geeignete Workloads. In der Speicher-Domäne helfen Datenklassifizierung, dokumentierte Richtlinien zur Zuordnung von Speicherklassen, Lifecycle-Policies, Wachstums-Alerts und konsequente Datenhygiene gegen verwaiste Objekte. Im Netzwerk stehen Traffic-Kartierung, Datenlokalität, CDN/Caching, Komprimierung und Freigabeprozesse für interregionale Transfers im Vordergrund. Governance-seitig sind Pflicht-Tags mit validierten Werten, „No tag, no deploy", Budgets und Alerts auf 50/75/90/100 Prozent, wöchentliche Reviews, monatliche Executive-Summaries, quartalsweise

Strategie-Updates, klare Service-Owner-Zuordnung und Schulungen entscheidend *(siehe Abbildung 3)*.

Metriken, KPIs und Zielbilder

Messbare KPIs machen Fortschritt steuerbar: Auslastung über Durchschnitt und 95./99. Perzentile, Leerlaufquote, Kosten pro Transaktion/Nutzer/Feature, Tag-Abdeckung über 95 Prozent, korrekt allokierte Kosten über 98 Prozent, Budgetabweichung unter zwei bis fünf Prozent sowie der Automatisierungsgrad, etwa Anteil automatisch skalierter Ressourcen oder policy-konformer Deployments. Sicherheit und Kosten gehen dabei Hand in Hand: Ein harter Tagging-Governor verhindert Ressourcenerstellung ohne Pflicht-Tags, Least Privilege schützt Budget- und Alert-Objekte, automatisierte Aufräumjobs beseitigen verwaiste Ressourcen, und Audits prüfen regelmäßig Allokation, Lifecycle-Policies und Netzwerkpfade.

Organisatorische Verankerung und Kulturwandel

Damit Kostenverantwortung nachhaltig wirkt, muss sie organisatorisch verankert werden. Teams integrieren Kosten-KPIs in OKRs und SLAs, fördern Transparenz über Dashboards und kurze Statusberichte, machen Erfolge sichtbar und reinvestieren Einsparungen in Resilienz oder Features. Eine Lernkultur mit Postmortems bei Budgetverletzungen, Brown-Bag-Sessions zu Best Practices und Communities of Practice stärkt die Akzeptanz. Ein 90-Tage-Fahrplan erleichtert den Einstieg:

- In den ersten 30 Tagen Sichtbarkeit und Standards schaffen, Tagging durchsetzen, Budgets und Alerts setzen und Quick Wins heben.
- In den Tagen 31 bis 60 technisch optimieren mit Rightsizing, Auto-Scaling, Speicher-Lifecycle-Policies und Netzwerk-Optimierungen.
- In den Tagen 61 bis 90 Preemptible-Piloten ausrollen, Governance härten, KPIs reviewen und die Roadmap für komplexere Optimierungen planen.

Häufige Fragen und klare Antworten

- Müssen wir Performance für geringere Kosten opfern? Nein. Ziel ist die „Kosten pro Outcome“ zu optimieren. Mit Auto-Scaling und flexiblem Rightsizing lassen sich SLAs sichern und zugleich Leerlaufkosten verringern.
- Lohnt sich der Aufwand für Tagging wirklich? Ja. Ohne Tags fehlen Allokation, Verantwortlichkeit und belastbare Analysen. Tagging ist das Fundament für alle weiteren Maßnahmen.
- Wie gehen wir mit unvorhersehbaren Spitzen um? Metrikbasierte Autoscaling-Policies, Puffer über Max-Limits und Lasttests. Für unkritische Teile Preemptible einsetzen, um den Preis pro Peak zu senken.
- Wie vermeiden wir „Kostenblindheit“? Wöchentliche Reviews, klare KPIs, Alerts, gemeinsames Dashboarding von Technik, FinOps und Produkt.
- Wie kann ich möglichst schnell Kosten sparen? Ein guter Einstieg ist der Cloud Advisor, hier werden automatisiert mögliche Kosteneinsparpotentiale gesammelt, aufbereitet und angezeigt *(siehe Abbildung 4)*.

Fazit

Zusammengefasst führt ein datengestütztes, rollenbasiertes und automatisiertes Vorgehen zu „mehr Cloud für weniger Geld“: Workloads werden verstanden, FinOps wird verankert, die OCI-Werkzeuge werden systematisch genutzt, Skalierungsstrategien werden wirksam eingesetzt, der Datenlebenszyklus wird aktiv gemanagt, das Netzwerk klug entworfen, Governance gestärkt und eine Kultur des Lernens etabliert. Das Ergebnis ist eine Cloud-Landschaft, die wirtschaftlich skaliert, Risiken reduziert und Innovation beschleunigt – genau das Versprechen moderner Architekturen in OCI.

Über den Autor

Marcus Schröder, Distinguished Account Cloud Engineer bei der Oracle Deutschland B. V. & Co. KG, kümmert sich um verschiedene Themen rund um Oracle Technology in und außerhalb der Cloud und verfügt über jahrelange Erfahrung speziell in den Bereichen Cloud, Betrieb, Middleware und Datenbank. Er studierte Technische Informatik und arbeitet seit über 25 Jahren bei Oracle.

Marcus Schröder
marcus.schroeder@oracle.com

Progressive Web-Apps & APEX: von Bluetooth bis NFC – mach die App zum Superhelden!

Daniel Horwedel, WGV Versicherungen

Progressive Web-Apps (PWA) verbinden die Flexibilität moderner Webanwendungen mit den Möglichkeiten nativer Apps. Sie lassen sich installieren, offline nutzen und greifen auf Gerätefunktionen wie Kamera, GPS oder Benachrichtigungen zu. APEX bietet dafür seit Version 21.1 deklarativen Support – etwa für Manifest, Service Worker oder Push Notifications. Noch nicht abgedeckte Funktionen lassen sich jedoch dank offener Browser-APIs einfach ergänzen und erweitern.

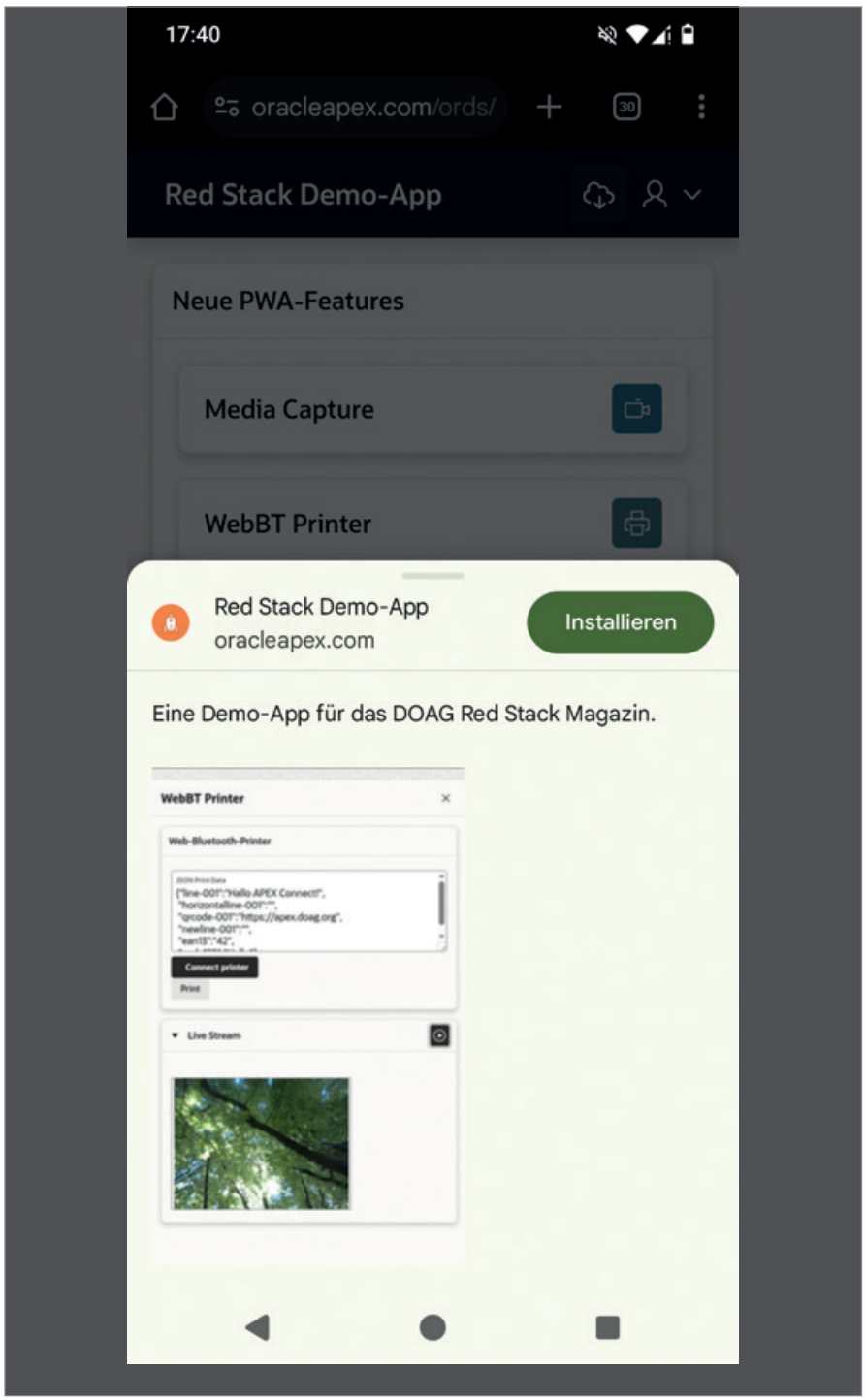


Abbildung 1: Installationsdialog der Progressive Web-App (Quelle: Daniel Horwedel)

PWA – was ist das überhaupt?

Progressive Web-Apps ermöglichen einer Web-App, Features zu implementieren und Funktionalitäten des Endgerätes zu nutzen, die üblicherweise nativen Anwendungen vorbehalten sind. Bei PWAs handelt es sich um keinen Standard, sondern um verschiedene Funktionalitäten, die teils noch als Draft in Entwicklung sind und daher auch nicht von jedem Browser beziehungsweise Endgerät unterstützt werden. Die Anwendungen sind üblicherweise als Responsive-Applikationen implementiert und in ihrem Look & Feel an native Apps angelehnt. Der Fokus der üblicherweise verwendeten PWA-Funktionalitäten bezieht sich meistens auf mobile Anwendungen, es lassen sich aber einige der Features auch auf Desktop-Geräten nutzen. Zentraler Bestandteil jeder PWA sind ein Manifest, das die grundlegenden Eigenschaften der App definiert und in einer standardisierten Form bereitstellt, sowie ein Service Worker, der sich um die Koordinierung der App-Features mit dem Browser kümmert und auch die Ausführung von asynchronen Background-Jobs ermöglicht.

Technische Voraussetzungen & Browser-Support

Die Unterstützung von PWAs durch die Browser ist an einige grundlegende Eigenschaften der Anwendungen gekoppelt – so ist es zum Beispiel erforderlich, dass eine PWA mit all ihren Komponenten durchgehend SSL-verschlüsselt betrieben wird.

Eine grundlegende Design-Entscheidung, die die Browser-Hersteller bei der Implementierung der Zugriffsmöglichkeiten auf das Endgerät eingeführt haben, ist die Zustimmung des Users zur Nutzung dieser Funktionen – so ist zum Beispiel kein Kamera-Zugriff ohne eine explizite Genehmigung des Users möglich. Die Abfrage dieser Erlaubnis erfolgt auch nicht durch einen Dialog in der Anwendung selbst, sondern durch einen hiervon – auch optisch – getrennten Dialog des Browsers.

Bei der Unterstützung von PWA-Features gibt es leider zwischen den verschiedenen Browsern und Betriebssystemen massive Unterschiede, die teils auch dazu führen, dass auf manchen Plattformen maximal eine rudimentäre Umsetzung möglich ist. Die beste Unterstützung für PWA-Features bietet Chrome unter Android. Auf dem Desktop ist die Unterstützung durch Chrome und die darauf basierenden Browser ebenfalls relativ gut und umfangreich, wohingegen Firefox-Nutzer schon einige Einschränkungen hinnehmen müssen. Unter iOS und Safari ist derzeit nur eine sehr rudimentäre Unterstützung gegeben, wodurch leider auf viele in diesem Artikel beschriebenen Funktionen verzichtet werden muss.

Ein umfangreicher Überblick über die Unterstützung auf den verschiedenen Plattformen ist auf *caniuse.com* zu finden.

PWA-Features in APEX

Mit der Version 21.1 wurde in APEX ein erster grundlegender PWA-Support im-

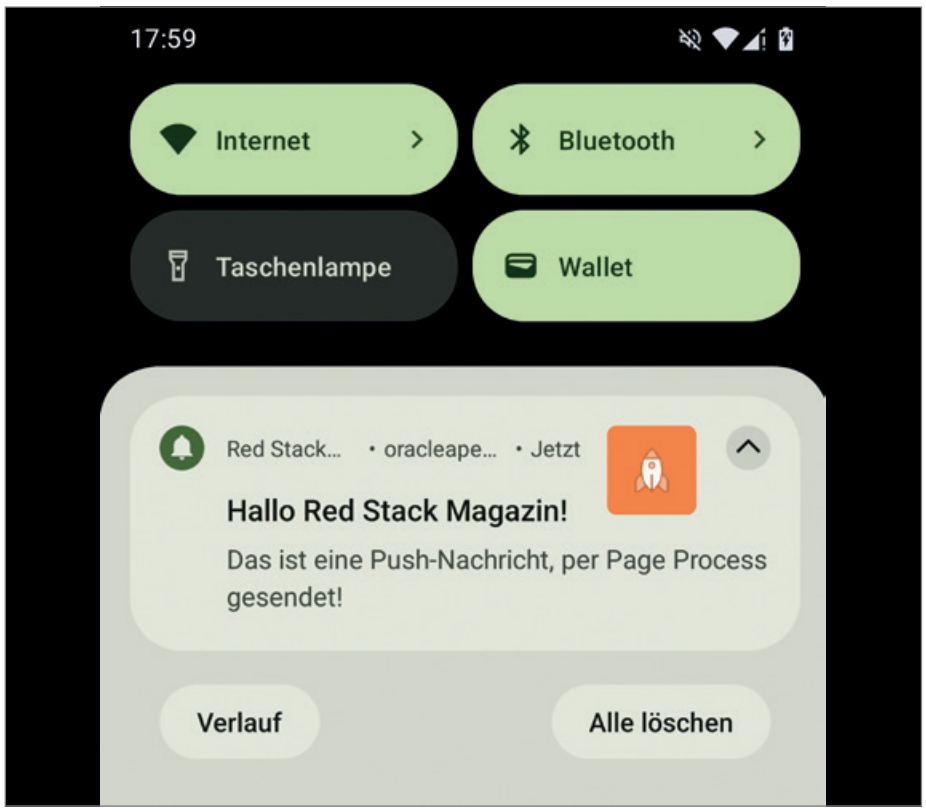


Abbildung 2: Push Notifications (Quelle: Daniel Horwedel)

```
{"share_target": {
    "action": "/pls/apex/r/redstack/pwa-demo/share-receiver",
    "method": "GET",
    "params": {
      "title": "P400_share_title",
      "text": "P400_share_text",
      "url": "P400_share_url"
    }
  }
}
```

Listing 1: Custom-Manifest-Erweiterung für ein Share Target

plementiert, der seitdem kontinuierlich erweitert wurde. APEX unterstützt deklarativ die Generierung eines PWA-Manifests, das um eigene Bestandteile erweitert werden kann – dadurch lassen sich sehr einfach auch (noch) nicht unterstützte Funktionalitäten implementieren. Ebenfalls möglich sind die Anpassung und Erweiterung des Service Workers um eigenen Code.

Zu den deklarativ nutzbaren PWA-Features, die APEX mitbringt, zählen unter anderem die Möglichkeit, eine App wie eine native App im Betriebssystem zu installieren, dadurch lässt sich diese direkt aus dem Launcher oder vom Homescreen starten und muss nicht erst durch die Eingabe einer URL oder das Aufrufen eines Lesezeichens im Browser geöffnet werden *(siehe Abbildung 1)*. Dadurch lassen sich damit außerdem auch – je nach System – Einstellungen wie das Deaktivieren des Energiesparmodus (z. B. für Hintergrund-Aktionen wichtig) sowie die Benachrichtigungseinstellungen des Systems steuern.

APEX bietet out-of-the-box ein an native Apps angelehntes Look & Feel – so lässt sich etwa die Titelleiste der Anwendung entsprechend den Systemstandards einfärben, die App kann im Fullscreen-Modus ausgeführt werden und es können eigene Icons für den Launcher hinterlegt werden. Auch eine Reaktion auf eine Änderung des Gerätes vom Hoch- ins Querformat erfolgt automatisch und die Inhalte werden dank des Responsive Design des Universal Theme automatisch angepasst.

Ein weit verbreitetes Feature im mobilen Umfeld ist die „Share"-Funktion, mit der Informationen wie beispielsweise Links, GPS-Koordinaten oder kurze Textinformationen mit anderen Apps geteilt werden können. APEX unterstützt hier das Teilen von Daten aus der APEX-Anwendung heraus in andere Apps.

Mit der Geolocation bietet APEX die Möglichkeit, deklarativ auf den GPS-Sensor im Smartphone zuzugreifen und darüber die aktuelle Position zu ermitteln. Hierfür steht die „Get Current Position"-Dynamic Action zur Verfügung, die die aktuellen Koordinaten wahlweise als Längen- und Breitengrad oder als GeoJSON mit weiterführenden Informationen wie Genauigkeit und Höhe zurückliefert.

Ebenfalls deklarativ nutzbar sind Push Notifications, die es ermöglichen, dem Nutzer Benachrichtigungen zukommen zu lassen, selbst wenn die App gerade nicht aktiv ausgeführt wird. Hierfür werden die Push Notification-APIs der jeweiligen Betriebssysteme verwendet, so dass die App nicht durchgehend ausgeführt werden muss. Die Notifications können mit einem eigenen Icon versehen werden und eine Aktion enthalten, die beim Klick auf die Benachrichtigung ausgeführt wird – zum Beispiel der Aufruf einer bestimmten Seite innerhalb der PWA *(siehe Abbildung 2)*.

Was fehlt in APEX derzeit noch?

APEX unterstützt aktuell nur eine Teilmenge der möglichen PWA-Funktionen – aufgrund der erweiterbaren Architektur des Frameworks lassen sich diese aber sehr einfach selbst ergänzen. Viele der PWA-Funktionen basieren auf APIs, die durch die Browser bereitgestellt werden und über JavaScript nutzbar sind. Dadurch lassen sich diese einfach und komfortabel in APEX-Anwendungen einbinden – für einen schnellen Proof of Concept direkt als JavaScript Function-Aufruf, optimalerweise aber als kleines APEX-Plugin, um eine Kapselung des Codes vorzunehmen und eine Wiederverwendbarkeit in mehreren Anwendungen zu ermöglichen. Im

Folgenden werden einige nützliche PWA-Funktionen und ihre mögliche Implementierung vorgestellt.

Web Share Target

Ein Web Share Target ist ein Endpunkt innerhalb der Anwendung, der als Empfänger für das Teilen von Inhalten aus anderen Apps fungiert. Damit bildet es das Gegenstück zur bereits in APEX vorhandenen Web-Share-Funktionalität und ermöglicht es, die APEX-PWA als Empfänger im Standard-Share-Dialog des Betriebssystems zu registrieren.

Zur Implementierung muss das durch APEX generierte PWA-Manifest um einen Custom Block erweitert werden. Das Manifest beschreibt die PWA-spezifischen Eigenschaften der Applikation und wird im JSON-Format erstellt. In diesem erweiterten Custom Manifest wird nun angegeben, welche Aktion beim Empfangen von Daten aus einer anderen App ausgeführt werden soll. Mögliche Daten, die aus anderen Apps empfangen werden können, sind URLs, Titel, Text sowie Dateien.

In *Listing 1* findet sich eine beispielhafte Ergänzung des Manifests. Als „action" wird die APEX-Seite definiert, die die Elemente enthält, die die übergebene Payload entgegennehmen. „title", „text" und „url" geben dann die Static IDs der Page Items an, in die die jeweiligen Werte geschrieben werden sollen. Die weitere Verarbeitung der empfangenen Daten erfolgt dann mittels den üblichen APEX-Prozessen *(siehe Listing 1)*.

Das Web Share Target lässt sich mit Safari und Firefox weder mobil noch auf dem Desktop verwenden.

Haptisches Feedback

Unter „haptischem Feedback" wird eine Interaktion mit dem Nutzer durch fühlbare Aktionen verstanden – zum Beispiel durch Vibrationsmuster. Die Vibrationsmuster lassen sich frei definieren, sodass auch unterschiedliche und eindeutige Patterns darstellbar sind.

Der Funktionsaufruf ist sehr simpel: es wird einfach eine JavaScript-Funktion aufgerufen und ein Array übergeben, das die Längen der einzelnen Vibrations- und Ruhephasen enthält *(siehe Listing 2)*.

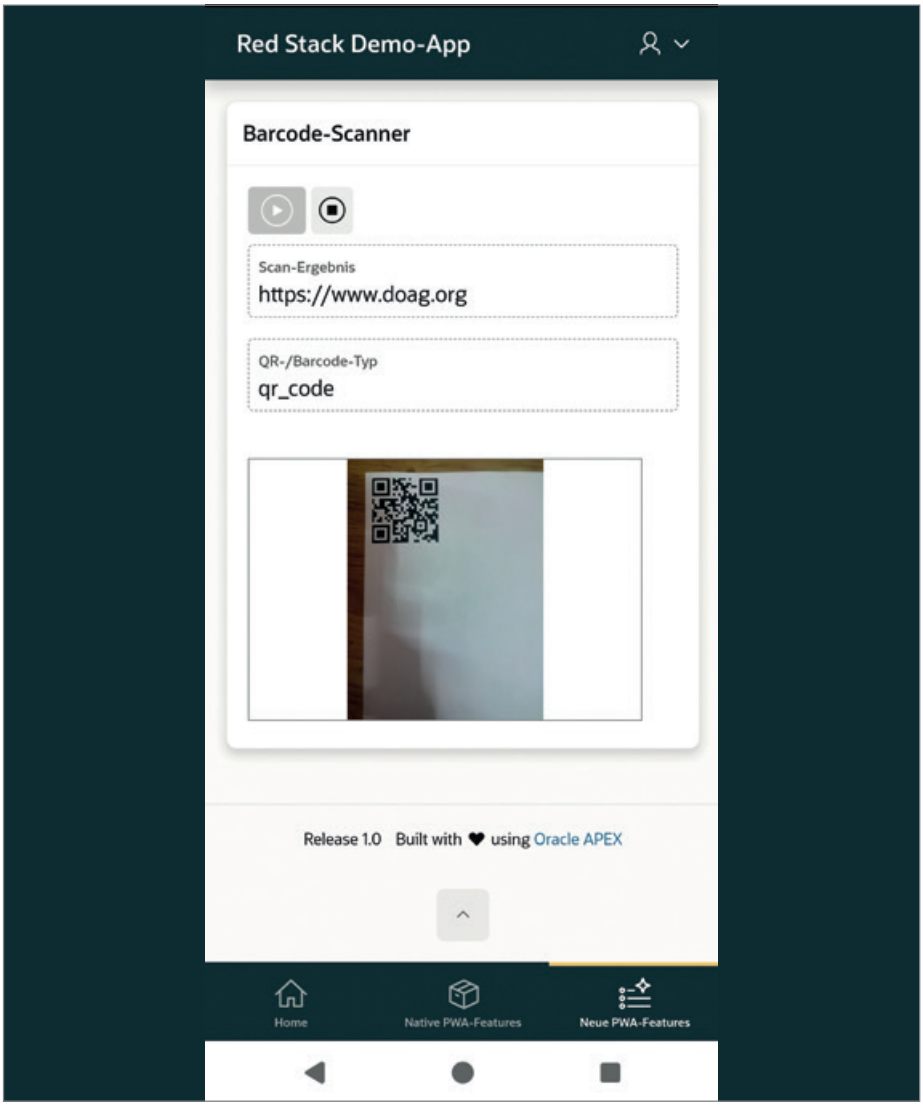


Abbildung 3: Barcode-Scanner in Aktion (Quelle: Daniel Horwedel)

Der Browser-Support ist bei dieser API deutlich umfangreicher als beim Web Share Target – allerdings wird dieses Feature von Safari ebenfalls nicht unterstützt. Aufgrund der üblicherweise auf Desktop-Geräten nicht vorhandenen Vibrationsmotoren, wird diese API hauptsächlich von mobilen Browsern unterstützt.

Media Capture

Mittels Media Capture lässt sich der Zugriff auf Kameras und Mikrofone ermöglichen – falls davon mehrere in einem Gerät verbaut sind, können diese auch gezielt ausgewählt und genutzt werden *(siehe Listing 3)*. Aus Sicherheits- und Datenschutzgründen wird dem Nutzer beim Zugriff auf diese Eingabegeräte durch den Browser ein Freigabe-Dialog angezeigt, der sich nicht aus der App heraus steuern lässt. Der Nutzer hat die Wahl, den Zugriff nur für die aktuelle Session oder dauerhaft zu erlauben.

Der Audio- oder Video-Stream lässt sich dann auf zwei verschiedene Arten verwenden: es ist sowohl eine direkte Verarbeitung möglich (z. B. für das Scannen von Barcodes oder Objekterkennung) als auch eine Aufzeichnung zur späteren Verwendung *(siehe Listing 4)*.

Media Capture wird sowohl auf dem Desktop als auch auf Mobilgeräten von allen gängigen Browsern unterstützt – eine Ausnahme bildet hier nur wieder Safari auf iOS-Geräten.

Barcode-Scanner

Durch die Kombination mehrerer PWA-APIs lassen sich auch fortgeschrittene Use cases umsetzen. Ein Beispiel hier-

```
navigator.vibrate([5, 200, 100, 200, 100, 400, 150, 150, 150, 500, 200, 200, 100, 200, 100, 600])
```

Listing 2: JavaScript-Code zum „Abspielen“ eines Vibrations-Patterns

```
<div>
<video id="videostream" width="300" height="220" style="border: 1px solid gray"></video>
</div>
```

Listing 3: Video-HTML-Element zur Darstellung des Kamera-Streams

```
window.addEventListener('load', function () {

    try {
        const startvideostream = document.getElementById(lStartButton); /* Static ID des "Start"-Buttons */
        const videostream = document.getElementById("videostream"); /* Static ID des Video-Elements */

        videostream.addEventListener("play", () => capture());

        startvideostream.addEventListener("click", () => {
          (async () => {
            const media = await navigator.mediaDevices.getUserMedia(
              {audio: false, video: {facingMode: "environment"}});
            videostream.srcObject = media;
            videostream.autoplay = true;

          })().catch(console.error);
        }, {once: true});
      } catch (err) {
        console.log(err);
      }
})
```

Listing 4: JavaScript-Code zum Starten des Videostreams

für ist die Implementierung eines QR- und Barcode-Scanners, der mittels der Media Capture-API über die Kamera eines Smartphones die Codes scannt. Im nächsten Schritt wird mit der in modernen Browsern bereitgestellten Barcode-Detector-API dieses Kamerabild analysiert und die darin hinterlegte Payload dekodiert *(siehe Abbildung 3)*.

Das Scannen von QR-Codes mittels JavaScript ist schon lange möglich – allerdings sind die hierfür benötigten Libraries ziemlich komplex und teilweise nicht besonders performant. Durch die Nutzung der Barcode-Detector-API des Browsers entfällt der Weg über die JavaScript-Libraries, wodurch sowohl die Performance als auch die Erkennungsrate deutlich optimiert werden *(siehe Listing 5)*.

Leider steht diese API im Safari-Browser unter iOS nicht zur Verfügung, hier ist aber das APEX-Region-Plugin „APEX-QR-Code-Scanner“ von Ronny Weiß eine gute Option.

Aufgrund der fehlenden Unterstützung für die Media Capture-API lässt sich ein Barcode-Scanner nicht für den mobilen Safari realisieren. Die Barcode-Detector-API wird grundsätzlich nur von mobilen Browsern unterstützt.

Interaktion mit Cyber-physischen Systemen

Im experimentellen Bereich der PWA-APIs finden sich einige interessante und vielversprechende Funktionen zur Interaktion mit Cyber-physischen Systemen. Dazu zählen zum Beispiel Bluetooth-Sensoren, per USB angebundene Makropads, aber auch Peripheriegeräte wie Thermodrucker oder NFC-Tags. Letztere sind ein Beispiel, die die einfach umsetzbare Integration nicht deklarativ unterstützter Features zeigen.

Web-NFC unterstützt hierbei lediglich einen Teil der NFC-Funktionen – es lassen sich ausschließlich die ID eines Tags sowie die Payload auslesen *(siehe Listing 6)*. Ein schreibender Zugriff – Hardwareseitige Unterstützung durch das Tag vorausgesetzt – ist lediglich auf die gespeicherte Payload möglich, die UUID des Tags lässt sich nicht verändern. Ebenfalls nicht möglich sind die Authentifizierung oder der Zugriff auf verschlüsselte NFC-Tags sowie die Durchführung von kontaktlosen Zahlungen mittels NFC-fähigen Kreditkarten. Hierfür gibt es die separate Payment-Request-API, die Kartenzahlungen mittels Wallets ermöglicht. Web-NFC wird ausschließlich von Chrome für Android unterstützt.

Debugging

Das Debugging von PWAs funktioniert auf dem Desktop wie bei klassischen Web-Anwendungen mittels der DevTools des jeweiligen Browsers *(siehe Abbildung 4)*.

```
window.addEventListener('load', function () {
  var lastScanResult;

        const start = document.getElementById(lStartButton);
        const video = document.getElementById("video");

        const bd = new BarcodeDetector();
        (async () => {

        })().catch(err => {
          apex.item(lResultItem).setValue(err);
        });

        const capture = async () => {
            const barcodes = await bd.detect(video);
            const bcode = barcodes.map(({rawValue}) => `${rawValue}`).join("\n");
            if (bcode != lastScanResult) {
                const bcodetype = barcodes.map(({format}) => `${format}`).join("\n");
                if (bcode) {
                    lastScanResult = bcode;
                    apex.item(lResultItem).setValue(bcode);
                    if (bcodetype) apex.item(lCodeTypeItem).setValue(bcodetype);
                }
                requestAnimationFrame(capture);
            }
        };

        video.addEventListener("play", () => capture());

        start.addEventListener("click", () => {
          start.disabled = true;
          (async () => {
            const media = await navigator.mediaDevices.getUserMedia(
              {audio: false, video: {

                facingMode: "environment"}});

            video.srcObject = media;
            video.autoplay = true;

          })().catch(console.error);
        }, {once: true});

})
```

Listing 5: Ausschnitt des JavaScript-Codes zur Integration des Barcode-Scanners. Der zugehörige Video-Container „video“ sowie der Start-Button mit der Static ID „lStartButton“ und das APEX-Item „lResultItem“ zur Rückgabe des gescannten Wertes müssen auf der zugehörigen APEX-Seite bereitgestellt werden.

```
async function readTag(targetItem) {
  if ("NDEFReader" in window) { /* Prüfen, ob WebNFC unterstützt wird */
    const reader = new NDEFReader();
      await reader.scan();
      reader.onreading = event => {
        const decoder = new TextDecoder();
        apex.item( "P1000_NFC_SERIAL" ).setValue(event.serialNumber); /* ID des Tags */
        apex.item( "P1000_NFC_PAYLOAD" ).setValue("");
        for (const record of event.message.records) {
          apex.item( this.action.attribute02 ).setValue(decoder.decode(record.data));
        } /* Die Payload ist in einem Record gespeichert */
      }
  }
}
```

Listing 6: JavaScript-Code zum Auslesen von NFC-Tags

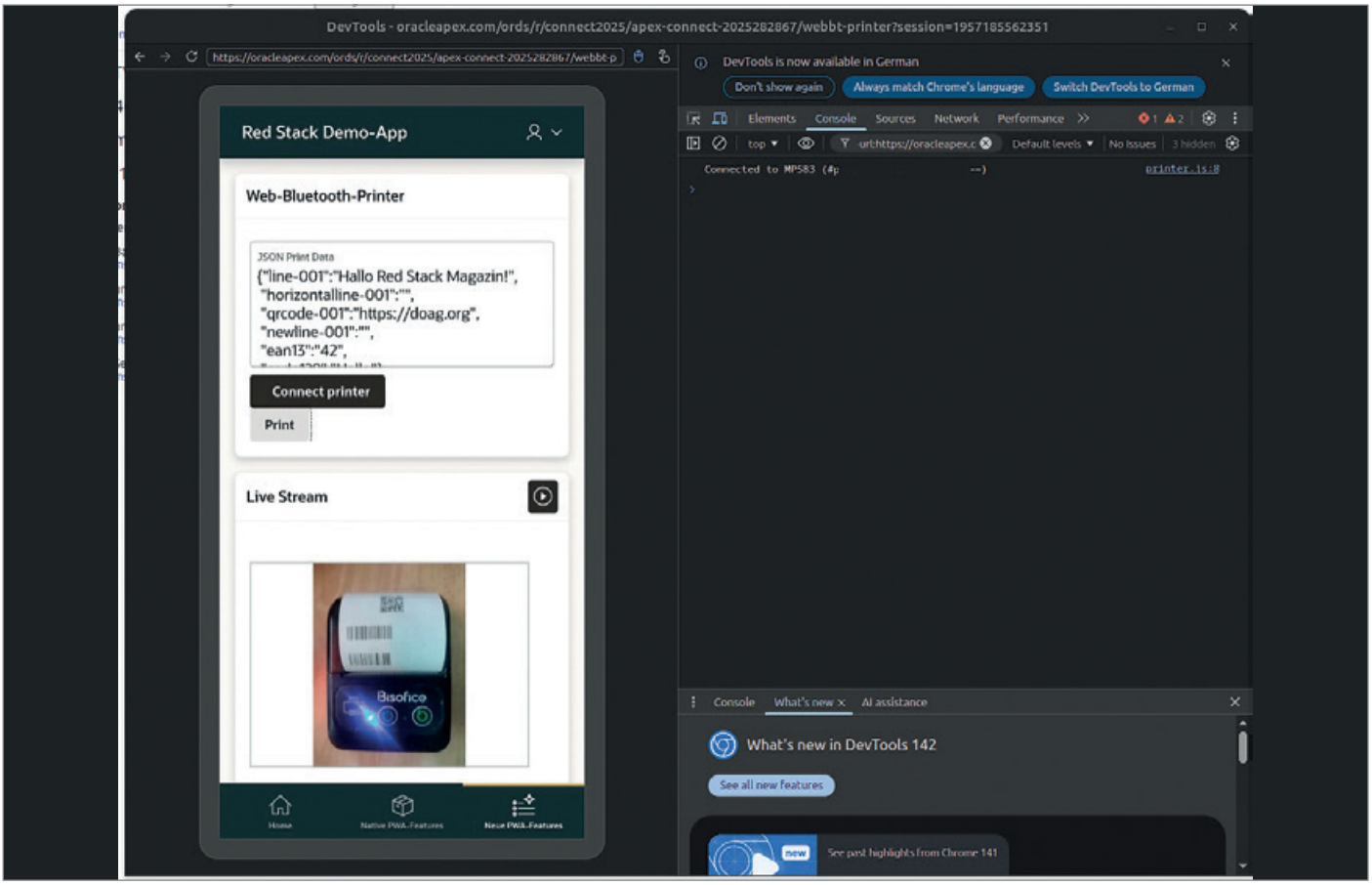


Abbildung 4: Chrome Developer Tools (Quelle: Daniel Horwedel)

Da PWAs aber überwiegend auf mobilen Endgeräten ausgeführt werden, wird auch eine Debugging-Möglichkeit für diese Geräteklassen benötigt. Hierzu stellen die Developer Tools der gängigen Browser mittlerweile komfortable Möglichkeiten bereit, um remote auf die Browser auf den Mobilgeräten zuzugreifen. Unter Android erfolgt dies mittels der Android Debug Bridge (ADB), wodurch eine Debug-Verbindung sowohl kabelgebunden als auch über WLAN aufgebaut werden kann. ADB muss in den Entwicklereinstellungen des Gerätes aktiviert werden. Mittels der Sonder-URL `chrome://inspect#devices` lassen sich zum Beispiel im Chrome die per ADB verbundenen Geräte anzeigen und fernsteuern, außerdem stehen die üblichen Funktionen der Entwicklertools nun auch für das verbundene Mobilgerät zur Verfügung.

Weiterführende Ressourcen

- Oracle APEX PWA Reference App mit Implementierungs- und Code-Beispielen. *https://apex.oracle.com/pwa*
- APEX to go Sample App: eine PWA, die als Showcase der aktuellen Möglichkeiten dient. *https://apex.oracle.com/pls/apex/r/apex_pm/apextogo*
- Can I Use – Feature-Support-Matrix zur einfachen Übersicht, welche Features mit welcher Betriebssystem-Browser-Kombination nutzbar sind. *Caniuse.com*
- Übersicht möglicher PWA-Features mit Beispielen. *whatpwacando.today*
- MDN Web Docs: Dokumentation der verfügbaren Browser-APIs. *https://developer.mozilla.org/en-US/docs/Web*
- APEX-QR-Code-Scanner Plugin von Ronny Weiß: *https://github.com/Ronny-Weiss/APEX-QR-Code-Scanner*
- Open Source Library zur Anbindung von Point of Sale-Hardware wie Thermodruckern, Laser-Barcodescannern oder externen Displays: *https://point-of-sale.dev/*

Über den Autor

Daniel Horwedel ist als Senior Software Developer und Architekt für APEX und ORDS-Applikationen bei der WGV Versicherung tätig. Zu seinen Tätigkeitsgebieten gehört die Entwicklung und Betreuung vielfältiger APEX-Applikationen im Versicherungsumfeld. Dabei entwickelt er auch gerne kreative Lösungen, die über die Standard-Möglichkeiten von APEX hinausgehen.

Daniel Horwedel
daniel.horwedel@wgv-informatik.de

Fighting Bad PL/SQL & SQL with VS Code

Philipp Salvisberg, Grisselbav

Der Erfolg eines Projekts oder Produkts hängt wesentlich von der Qualität des Codes ab. Doch wie verbessere ich Sicherheit, Wartbarkeit oder Performance? Und noch wichtiger: Wie verhindere ich, dass Code mit Qualitätsmängeln überhaupt in die Produktion gelangt? Neben Tests spielt die Codeanalyse eine zentrale Rolle. Heute lässt sich Code nicht nur in CI/CD-Pipelines prüfen, sondern bereits während der Entwicklung. Moderne IDEs wie Visual Studio Code erkennen Qualitätsprobleme automatisch und bieten konkrete Lösungsvorschläge (Quick Fixes) an. Ziel ist es den Feedback-Loop zu minimieren.

Softwarequalität messen

Stell dir vor, du arbeitest in einem Team, welches Code-Änderungen auf einem Feature- oder Bugfix-Branch erst nach einem Review in den Develop- oder Main-Branch mergen. Manchmal bist du der Autor und manchmal ein Reviewer. Ich arbeite derzeit in einem Team, das genauso arbeitet. Neben der Qualitätssicherung ist auch das Know-how-Sharing ein wichtiger Aspekt. Aber wie können wir die Code-Qualität beurteilen?

Das bekannte Meme von Thom Holwerda *[1]* sagt: „The only valid measurement of code quality is WTFs per minute“ *(siehe Abbildung 1)*. Dies scheint eine gute Metrik zu sein. Das Problem dieser Metrik ist:

- sie ist willkürlich;
- sagt nichts darüber aus, was geprüft wird;
- hilft neuen Team-Zugängen nicht sich einzuarbeiten;
- und sie lässt sich nicht sinnvoll automatisieren.

Früher oder später bleibt einem Team nichts anderes übrig, als sich auf ein Set von Regeln zu einigen. Informell oder formell. Typische Coding Guidelines behandeln Themen wie Namenskonventionen, Formatierung, Code-Komplexität und Regeln zur Sprachverwendung. Es geht darum, die Qualitätsmerkmale in den Bereichen Wartbarkeit, Zuverlässigkeit, Performance und Sicherheit zu definieren und diese einzuhalten. Schriftlich festgehaltene Coding Guidelines vereinfachen auch das Onboarding neuer Mitarbeiter.

Automatisierung

Die manuelle Überprüfung der Einhaltung von Coding Guidelines im Rahmen eines Reviews ist so aufwendig, dass sich das niemand leisten kann. Ergo müssen wir so viele Aspekte wie möglich automatisch überprüfen. Dies ist für viele Aspekte möglich. Die anderen müssen nach wie vor im Rahmen eines manuellen Reviews überprüft werden.

Mit anderen Worten, wir brauchen eine Tool-Suite zur automatischen Überprüfung von Regeln. dbLinter *[2]* ist eine solche Tool-Suite. Es handelt sich um ein neues Produkt, das von Grisselbav und United Codes entwickelt und stetig weiter verbessert wird. dbLinter verwendet den IslandSQL Parser *[8]*, der mit jedem Release-Update der Oracle AI Datenbank für SQL, PL/SQL, SQL*Plus und SQLcl aktualisiert wird.

dbLinter positioniert sich als Nachfolgeprodukt der dekommissionierten db* CODECOP Suite von Trivadis. dbLinter besteht aus einem zentralen Repository, einer Web GUI und Client-Tools wie einem CLI, einem SonarQube Plugin und einer VS Code Extension. Die VS Code Extension ist kompatibel mit allen IDEs, welche die Visual Studio Code Extension API *[3]* implementieren. Konkret: Visual Studio Code, VSCodium, Cursor, Windsurf, Antigravity and Theia IDE.

dbLinter for VS Code ist im Visual Studio Marketplace *[4]* und in der Open VSX Registry *[5]* verfügbar und kann darum bequem direkt aus der IDE installiert werden. Die Extension kann im Rahmen der Anonymous und Starter-Pläne kostenlos eingesetzt werden *(siehe [9])*.

Wichtig ist, dass der Code immer lokal analysiert wird, das heißt, dein Code verlässt das lokale Netzwerk niemals. Vom zentralen Repository wird die Konfiguration gelesen, damit der Client weiß, welche Regeln und welche Parameter aktiv sind. Mit dem Web GUI *[6]* werden unter anderem diese Konfigurationen verwaltet.

dbLinter bietet derzeit 186 Regeln an. Dazu gehören alle Regeln der ebenfalls dekommissionierten Trivadis PL/SQL & SQL Coding Guidelines. Die Konfiguration in *Abbildung 2* zeigt meine 14 Lieblingsregeln. Eine Verletzung einer dieser Regeln ist ein Bug. Hier gibt es kaum Interpretationsspielraum. Nachfolgend zwei Beispiele.

Top-n-Abfragen mit ROWNUM

Die Query in *Listing 1* basiert auf der berühmten EMP-Tabelle des Datenbank-Benutzers SCOTT. Die Query liefert zwei Datensätze zurück.

Das Ergebnis entspricht aber kaum den Anforderungen. Hier würde ich eher erwarten, dass die zwei Mitarbeiter mit dem größten Salär angezeigt werden. Warum ist das nicht der Fall? Weil die Abfrage so formuliert ist, dass willkürlich zwei Datensätze gelesen und diese dann nach dem Salär absteigend sortiert werden. Das Ergebnis ist vom Ausführungsplan und der physischen Speicherung der Datensätze abhängig.

In VS Code werden die Regelverletzungen unter „Problems“ angezeigt *(siehe Abbildung 3)*. Dort wird auch ein Link „Core G-3185“ angeboten, der die Details zur Regel im Web GUI öffnet.

Die Regel G-3185 im Web GUI wird basierend auf einem ähnlichen Fall im HR-Schema erklärt *(siehe Abbildung 4)*. Hier gibt es zwei Lösungen. Eine, die in allen Oracle-Datenbank-Versionen anwendbar ist, und eine, welche ab der Version 12c funktioniert. *Listing 2* zeigt die veraltete Lösung mit Hilfe einer Inline-View. Heutzutage würde ich immer eine Lösung mit der row_limiting_clause bevorzugen.

Wir sehen auch, dass die Regel in 6 Konfigurationen verwendet wird. Mit einem Klick auf „Details“ können in einem Pop-Up-Fenster Änderungen vorgenommen werden. So lassen sich Regeln direkt aus der IDE konfigurieren.

NVL vs. COALESCE

Eine häufig unterschätzte Regel ist “G-4230: Always use a COALESCE instead of a NVL command, if parameter 2 of the NVL function is a function call or a SELECT statement.“ Es geht hier um Performance und die ist für uns alle relevant, vor allem wenn sie schlecht ist. Darum ist diese Regel wichtig und notwendig.

Listing 3 zeigt einen Fall, in dem die Verwendung von NVL zu einer Ausführungszeit von etwa 2.8 Sekunden führt. Wird hingegen COALESCE statt NVL verwendet, beträgt die Ausführungszeit nur noch etwa 0.2 Sekunden. Der Grund dafür ist, dass NVL in jedem Fall den zweiten Parameter evaluiert, selbst wenn der erste Parameter nicht NULL ist. COALESCE funktioniert da anders, der zweite Parameter wird nur evaluiert, wenn der erste NULL ist. Da die EMP-Tabelle 14 Datensätze beinhaltet und nur ein Datensatz eine leere MGR zurückliefert, entsteht dieser Performance-Unterschied.

In diesem Fall ist die Verwendung von COALESCE definitiv die bessere Lösung. Sollte darum NVL immer durch COALESCE ersetzt werden? Nein. Es gibt folgende zwei Sonderfälle:

1. NVL unterstützt implizite Datenkonversion. „select nvl(mgr, '1') from emp“ funktioniert, aber „select coalesce(mgr, '1') from emp“ führt zu einem ORA 00932-Fehler. Allerdings sind implizite Datenkonversionen sowieso zu vermeiden und der Code entsprechend anzupassen.
2. Optimierungen von NVL, welche zu besseren Ausführungsplänen führen können. Solche Optimierungen existieren derzeit nur für NVL, aber nicht für COALESCE. In diesen Fällen ist es besser NVL weiterzuverwenden.

dbLinter kennt diese Spezialfälle und bietet entsprechende, alternative Lösungsvorschläge an *(siehe Abbildung 5)*.

Quick Fixes können von dbLinter für verschiedene Bereiche appliziert werden. Für einen einzelnen Verstoß einer Regel, für alle Verstöße einer Regel oder für alle Verstöße aller Regeln im Editor-Fenster.

Die @dbLinter ignore Marker sind spezielle Kommentare, welche dazu dienen, Regelverstöße zu ignorieren. Es

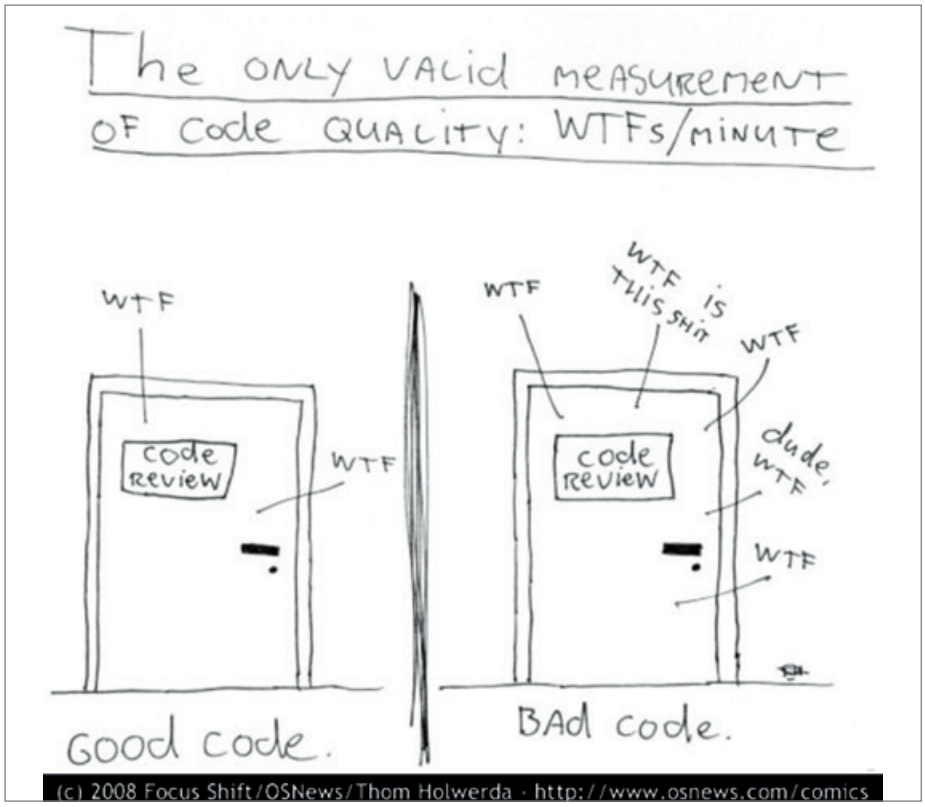


Abbildung 1: WTFs/m (Quelle: Philipp Salvisberg)

```
select ename,
       sal,
       hiredate,
       rownum as rank
  from emp
 where rownum <= 2
 order by sal desc;

ENAME             SAL HIREDATE         RANK
---------- ---------- ---------- ----------
ALLEN            1600 20.02.1981          2
SMITH             800 17.12.1980          1
```

Listing 1: Fehlerhafte Top-n-Query, pre-12c-style

gibt verschiedene legitime Gründe dies zu tun. Beispielsweise bei falsch-positiv-Meldungen (dbLinter Bugs) oder bei Ausnahmen von der Regel. Ein Kommentar wie „-- @dbLinter ignore(G-4230): NVL forever" kann folgende Anwendungsbereiche haben: Zeile, Statement oder Datei, je nachdem, wo der Kommentar positioniert wird.

Code-Analyse und künstliche Intelligenz

Die Lösungsvorschläge in *Abbildung 5* mit dem gelben Lampen-Symbol sind Quick Fixes von dbLinter. Die letzten beiden Menüoptionen stammen vom GitHub-Copilot.

Wenn der Copilot einen Fix für eine Regelverletzung anbietet, dann erhält er von dbLinter einen Kontext: die Datei, die genaue Position der Regelverletzung und welche Regel verletzt wurde. Damit steht ihm auch die Regeldefinition im Internet zur Verfügung und er kann in vielen Fällen einen vernünftigen Lösungsvorschlag liefern. Mit anderen Worten, dbLinter verbessert die Unterstützung der Tools wie Copilot, Cline oder Windsurf.

Regeln als Input für AI-Tools sind sinnvoll. Sie stecken den Rahmen ab und führen zu einem besseren Ergebnis. Allerdings ist es nicht garantiert, dass alle Regeln eingehalten werden. Am Ende wird auch der von AI generierte oder erweiterte Code zu unserer Code-Base. Darum ist es wichtig den von AI erzeugten Code nach den gleichen Regeln zu prüfen, wie den manuell erstellten Code.

Mit anderen Worten: AI setzt keine Standards durch, die Codeanalyse mit dbLinter hingegen schon.

Fazit

Softwarequalität ist eine Definitionssache. dbLinter bietet viele Regeln. Nicht alle sind für jedes Projekt sinnvoll und anwendbar. Die dbLinter-Regeln verstehen sich daher als ein großes Buffet. Man wählt aus, was einem schmeckt und was gesund ist. Aber nicht zu viel auf einmal. Nichtsdestotrotz gibt es einige Regeln, die aus meiner Sicht in jedem Projekt einzuhalten sind. Zwei davon habe ich ihn diesem Artikel vorgestellt.

Installiere dbLinter for VS Code, um herauszufinden, welche Regeln für deine Projekte hilfreich sind. Die Regeln in der *Abbildung 2* sind sicher ein guter Startpunkt.

Wenn du Fragen hast, melde dich gerne bei mir. Du kannst dich aber auch gerne direkt an das dbLinter GitHub-Repository *[7]* wenden.

Quellen

[1] Thom Holwerda, WTFs/m, *https://www.osnews.com/story/19266/wtfsm/*
[2] dbLinter, *https://grisselbav.github.io/dbLinter/*
[3] Visual Studio Code Extension API, *https://code.visualstudio.com/api*
[4] Visual Studio Marketplace, *https://marketplace.visualstudio.com/*
[5] Open VSX Registry, *https://open-vsx.org/extension/Grisselbav/dblinter*
[6] dbLinter Web GUI, *https://dblinter.app/*
[7] dbLinter GitHub Repository, *https://github.com/Grisselbav/dbLinter*
[8] IslandSQL Parser, *https://github.com/IslandSQL/IslandSQL*
[9] dbLinter Pricing Plans, *https://www.united-codes.com/products/dblinter/#pricing*

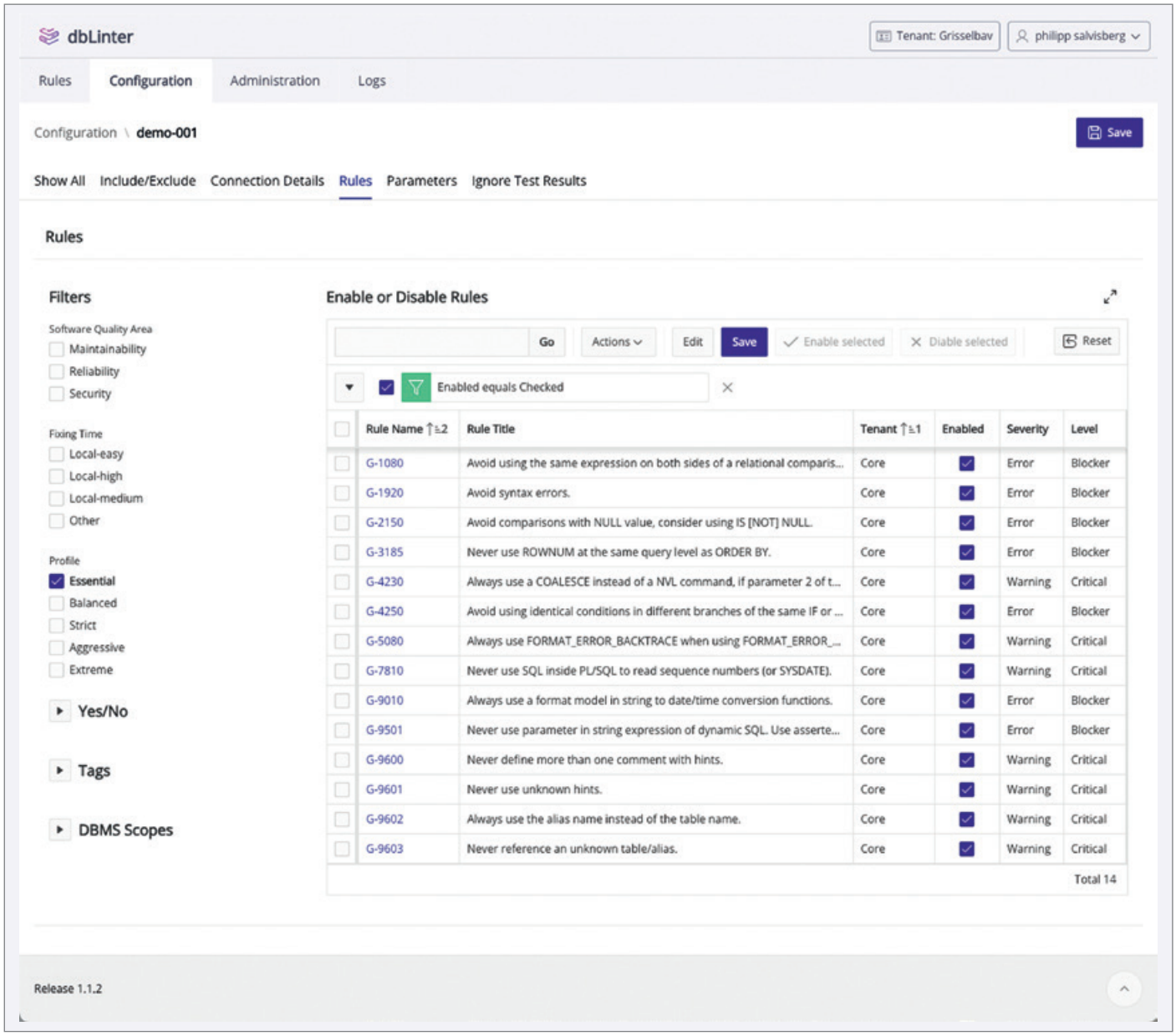
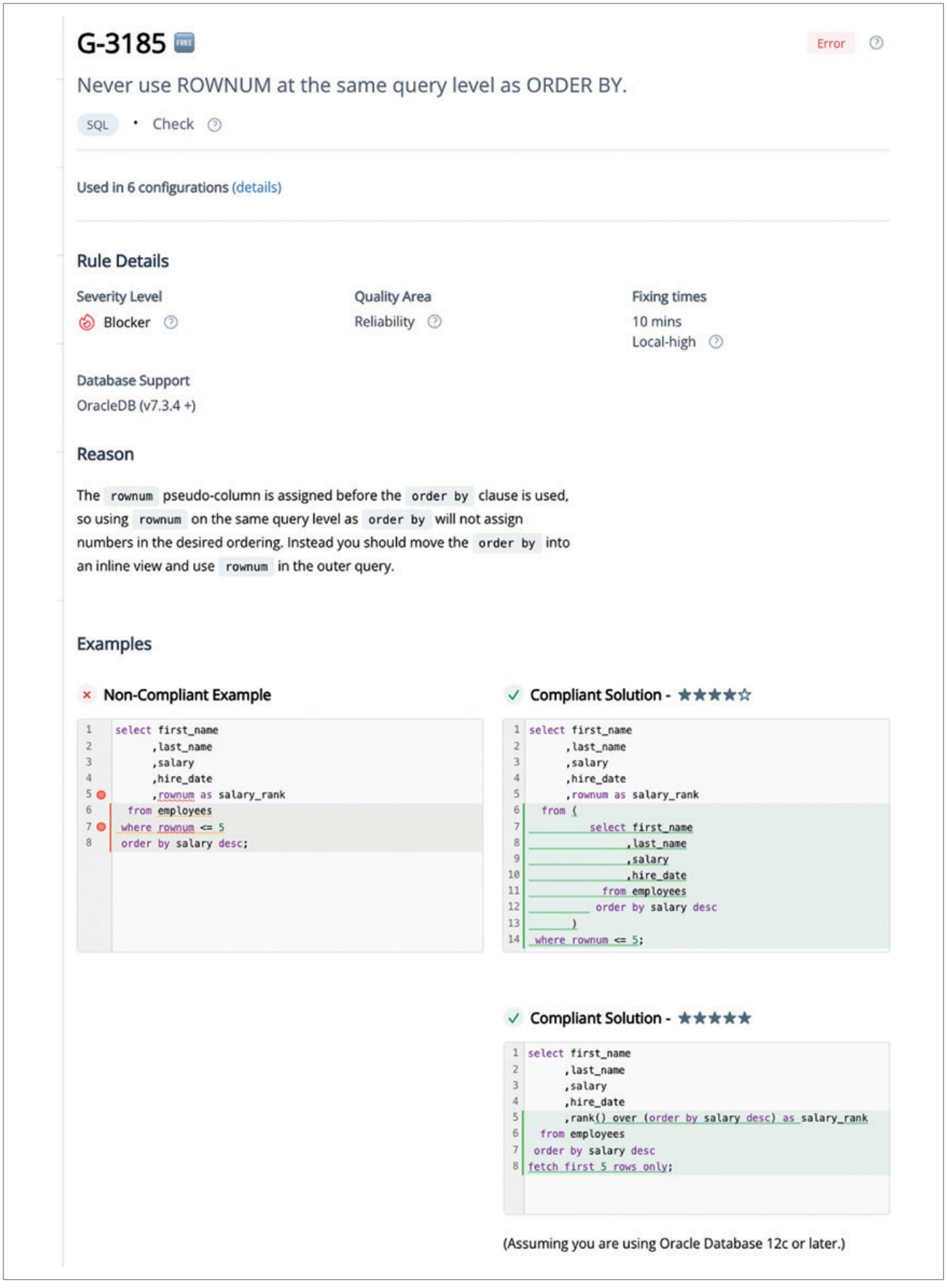


Abbildung 2: dbLinter Regel-Konfiguration im Web GUI (Quelle: Philipp Salvisberg)

Abbildung 3: G-3185 Regelverletzung in VS Code (Quelle: Philipp Salvisberg)



G-3185

Error

Never use ROWNUM at the same query level as ORDER BY.

SQL • Check

Used in 6 configurations (details)

Rule Details

Severity Level
Blocker

Quality Area
Reliability

Fixing times
10 mins
Local-high

Database Support
OracleDB (v7.3.4 +)

Reason

The `rownum` pseudo-column is assigned before the `order by` clause is used, so using `rownum` on the same query level as `order by` will not assign numbers in the desired ordering. Instead you should move the `order by` into an inline view and use `rownum` in the outer query.

Examples

Non-Compliant Example

```
1  select first_name
2       ,last_name
3       ,salary
4       ,hire_date
5       ,rownum as salary_rank
6    from employees
7   where rownum <= 5
8   order by salary desc;
```

Compliant Solution - ★★★★☆

```
1  select first_name
2       ,last_name
3       ,salary
4       ,hire_date
5       ,rownum as salary_rank
6    from (
7            select first_name
8                  ,last_name
9                  ,salary
10                 ,hire_date
11             from employees
12           order by salary desc
13         )
14  where rownum <= 5;
```

Compliant Solution - ★★★★★

```
1  select first_name
2       ,last_name
3       ,salary
4       ,hire_date
5       ,rank() over (order by salary desc) as salary_rank
6    from employees
7   order by salary desc
8  fetch first 5 rows only;
```

(Assuming you are using Oracle Database 12c or later.)

Abbildung 4: G-3185 Regel im dbLinter Web GUI (Quelle: Philipp Salvisberg)

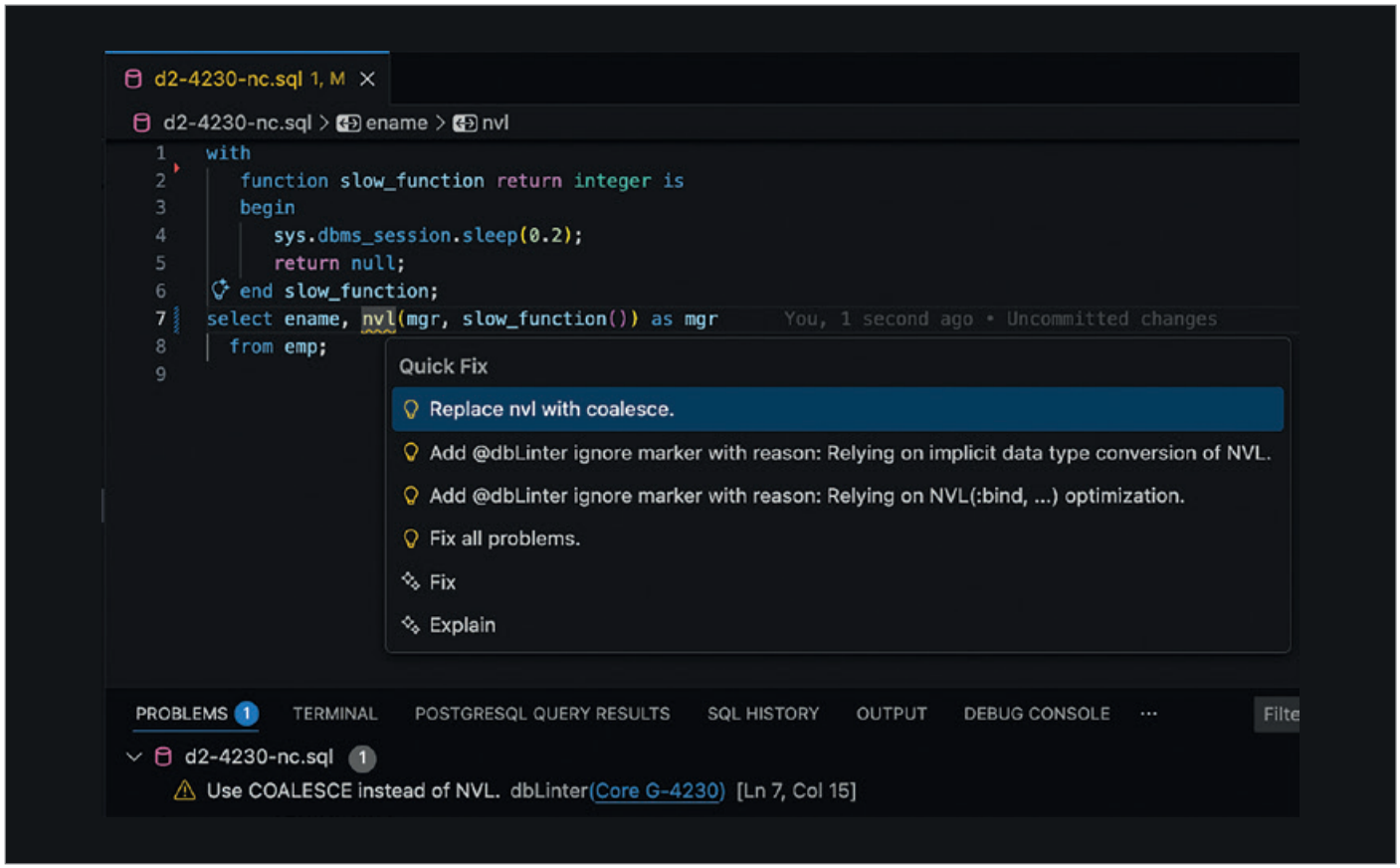


Abbildung 5: G-4230 Quick Fixes in VS Code (Quelle: Philipp Salvisberg)

```
select e.*,
       rownum as rank
  from (
          select ename,
                 sal,
                 hiredate
            from emp
           order by sal desc
       ) e
 where rownum <= 2;

ENAME             SAL HIREDATE         RANK
---------- ---------- ---------- ----------
KING             5000 17.11.1981          1
SCOTT            3000 19.04.1987          2
```

Listing 2: Korrekte Top-n-Query, pre-12c-style

```
with
   function slow_function return integer is
   begin
      sys.dbms_session.sleep(0.2);
      return null;
   end slow_function;
select ename, nvl(mgr, slow_function()) as mgr
  from emp;
```

Listing 3: Schlechte Performance mit NVL

Über den Autor

Philipp Salvisberg ist der Inhaber der Grisselbav GmbH und ein Oracle ACE Pro. Seit 1988 fokussiert er sich auf datenbankbasierte Lösungen und unterstützt Kunden beim Design, Aufbau und der Optimierung ihrer datenbankzentrischen Lösungen. Softwarequalität und Automatisierung haben dabei einen hohen Stellenwert. Philipp hat ein Faible dafür, möglichst viel in einer einzigen SQL-Anweisung auszuführen und ist an so ziemlich allem interessiert, um die Datenbank so effizient wie möglich zu nutzen.

Philipp Salvisberg
philipp.salvisberg@grisselbav.com

Programmieren nach Daten – Teil 3

Jürgen Sieben, ConDes

Dies ist eine kleine Reihe mit Anwendungsbeispielen, die zeigen, wie der Einsatz von SQL in der Programmierung der Datenbank nicht nur erhebliche Codemengen einsparen, sondern zudem auch die Effizienz und Funktionalität Ihrer Anwendungen erhöhen kann. Hier geht es nicht um abgefahrene Spezialfunktionen, sondern um eine Erweiterung des Blickwinkels in der Programmierung mit PL/SQL.

„Expertensystem"

In meinem letzten Artikel aus dieser Reihe habe ich erläutert, wie Entscheidungstabellen die Programmierung von Anwendungen flexibler und anwenderfreundlicher gestalten können. Ein häufig vorgebrachtes Gegenargument gegen solche Entscheidungstabelle ist aber, dass die Entscheidungen, die dort getroffen werden, nur schwer nachvollzogen werden könnten. Dies stellt ein Problem für das automatisierte Testen dar, zudem existieren Einsatzbereiche, die als Ergebnis der Abfrage genau diese Information benötigen, um überhaupt eingesetzt werden zu können.

Um das Problem zu verdeutlichen, zeige ich in *Listing 1* eine einfache Abfrage, die einen Vergleich zweier Tabellen mit Hilfe der `decode`-Funktion realisiert. Ich hatte dieses Beispiel in meinem Artikel zum Thema Delta-Views verwendet, diesmal aber die `where`-Klausel insofern abgewandelt, sodass die Prüfung über `or`-Verknüpfungen der Bedingungen realisiert wird.

Diese Schreibweise macht das Problem von Entscheidungstabellen deutlich: Welcher Zweig der or-Bedingungen hat dazu geführt, dass eine Entscheidung getroffen wurde? Sicher ist es trivial, wenn nur drei Prüfungen durchgeführt wurden, doch die Regeln können häufig umfangreich und komplex werden. Da die Prüfungen jedoch in der `where`-Klausel der Abfrage hinterlegt wurden, hat SQL in der select-Klausel keinen Zugriff auf diese Daten und kann deshalb nicht anzeigen, welche der drei Bedingungen für die Anzeige der aktuellen Zeile relevant war.

In diesem Artikel möchte ich daher das Konzept der Entscheidungstabelle so erweitern, dass die Auswertung uns auch die Information darüber gibt, warum eine Entscheidung getroffen wurde. Formal wird die Entscheidungstabelle so zu einer

möglichen Implementierung eines Expertensystems, doch scheue ich mich, diesen Begriff zu verwenden, denn die Anwendungsbeispiele dieses Artikels verdienen diesen Namen zwar nicht recht, sie sind aber dennoch höchst wirkungsvoll.

Der Trick

Wieder verwenden wir einen einfachen SQL-Trick, den jeder semantisch kennt, aber nur wenige in Bezug auf dieses Problem als Lösungsweg erkennen: Wir filtern die Tabelle nicht über die `where`-Klausel, sondern über einen Join. Das Join-Bedingungen Daten von Tabellen filtern können, wissen wir im Grunde, denn ein Inner Join kann Zeilen ausblenden, falls für ein Join-Kriterium keine Zeile in der referenzierten Tabelle gefunden wurde. Die Filterwirkung ist es, die uns zu Outer Joins greifen lässt. In Datenwarenhäusern ist dieser Weg zur Filterung von Daten deutlich häufiger anzutreffen, denn das spezielle Datenmodell vieler Datenwarenhäuser als Star-Schema beruht im Wesentlichen darauf, eine Faktentabelle über Joins auf Dimensionstabellen zu filtern, weil sich diese Dimensionstabellen einfacher über eine `where`-Klausel filtern lassen als die riesigen Faktentabellen.

Für Entscheidungstabellen ist dieser Weg aber ebenso sinnvoll einsetzbar. Wir werden also die Bedingungen aus der `where`-Klausel in die `join`-Klausel verschieben, müssen uns aber fragen, gegen welche Tabelle wir eigentlich einen Join ausführen möchten. Hier folgt der zweite Teil unserer Idee: Wir erstellen eine Tabelle mit den Bedingungen und geben den Bedingungen eine technische ID, einen erläuternden Text sowie optional ein Sortierkriterium mit, um eine Rangfolge der Bedingungen durchsetzen zu können, falls dies erforderlich ist. Die Bedingungen werden dann als Join-Kriterium verwendet. *Listing 2: Regeln des »Expertensystems« als View* zeigt eine solche Tabelle (Syntax nach Datenbankversion 21c), hier als View ausgeführt.

Bevor wir diese Tabelle nutzen können, müssen wir ein Problem lösen, denn natürlich ist es nicht möglich, die Inhalte einer Spalte als Join-Kriterium zu verwenden: der boolesche Ausdruck, der in der Bedingungsspalte steht, kann nur als Text interpretiert werden, nicht aber als Teil einer SQL-Anweisung. Um dieses Problem zu lösen, können wir über ein SQL-Makro oder einen Codegenerator nachdenken, um aus diesen Daten dynamisch die Join-Bedingung zu berechnen. Das halten wir als eine Erweiterung für später fest, ich möchte zunächst die Verwendung zeigen, indem ich diese Bedingungen von Hand in die Join-Bedingungen schreibe.

```
SQL> select coalesce(sap_id, best_id) best_id,
  2         sap_parent_id best_parent_id,
  3         sap_name best_name,
  4         case when best_id is null then 'I'
  5              when sap_id is null then 'D'
  6              else 'U' end best_dml_action
  7    from bestand
  8    full join sap
  9      on best_id = sap_id
 10   where decode(best_id, sap_id, 0, 1) > 0
 11      or decode(best_parent_id, sap_parent_id, 0, 1) > 0
 12      or decode(best_name, sap_name, 0, 1) > 0;
```

Listing 1: Beispielhafte Entscheidungstabelle

Listing 3 zeigt, wie die Erweiterung funktioniert. Der Trick der Abfrage besteht darin, dass die Join-Bedingungen ab Zeile 10 nicht nur auf den booleschen Ausdruck referenzieren, sondern parallel auch die technische ID der Regel prüfen. Auf diese Weise werden nur die Regeln angezeigt, für die sowohl die technische ID als auch die zugehörige boolesche Bedingung übereinstimmt.

Auch in dieser Abfrage kann es sein, dass mehrere Regeln für eine Zeile ein positives Resultat liefern. In diesen Fällen hängt es von Ihrem Kontext ab, ob Sie alle Zeilen oder nur die „beste" Zeile anzeigen möchten. Welche Zeile das ist, müssen Sie über ein Sortierkriterium bestimmen und über dieses Kriterium eine Top-N-Analyse (zum Beispiel mittels der `row limiting`-Klausel) durchführen.

Die Kraft solcher Systeme liegt darin, dass die Entscheidungsfindung datengetrieben durchgeführt wird, nicht durch hart kodierte Programmierung. Nehmen wir für unser Beispiel an, dass wir zwei weitere Regeln einführen möchten, die entscheiden, ob eine Neuanlage oder eine Löschung eines Eintrags erfolgte. Es ist trivial, diese Regeln zu formulieren, denn die technische Bedingung lautet für den ersten Fall `best_id is null` und für den zweiten `sap_id is null`. Beide sind einfach mit einer Rationale zu versorgen und in die Tabelle einzutragen, und schon wird die Programmierung, die aus diesen Daten die `select`-Anweisung erzeugt, dies entsprechend berücksichtigen: Die Zeilen der Entscheidungstabelle kontrollieren das Verhalten der select-Anweisung. Gleiches gilt, wenn Sie die Gewichtung der Regeln modifizieren.

Anwendungsbeispiel

Eine Anwendung, die ich in vielen Projekten verwende, basiert maßgeblich auf diesem Ansatz. Die Anwendung ist ein regelbasiertes System, bei dem Entwickler für eine APEX-Anwendungsseite Regeln formulieren können, die festlegen, welches dynamische Verhalten als Antwort auf eine gegebene Situation ausgeführt werden soll.

Ein Beispiel könnte sein: „Wenn der Anwender in einem Dialog zur Neuanlage von Mitarbeitern einen Beruf auswählt, der bonusberechtigt ist, soll das Eingabefeld P1_BONUS angezeigt und zu einem Pflichtfeld gemacht werden." Diese Anforderung ist in APEX nicht trivial umzusetzen, weil die Frage, ob ein Beruf bonusberechtigt ist oder nicht, in aller Regel basierend auf Daten der Datenbank beantwortet werden muss (abgesehen davon, dass es APEX zudem nicht erlaubt, ein Eingabefeld dynamisch zu einem Pflichtfeld zu machen). Daher ist die Bedingung bereits an eine Datenanalyse aus der Datenbank gebunden. Die Anwendung erleichtert diese Dinge durch die Formulierung von Regeln auf der Datenbankseite, die als Antwort ein JavaScript-Script liefert, das das clientseitige dynamische Verhalten kontrolliert.

```
SQL> create or replace view orga_regel_v as
  2  select *
  3    from (values
  4           (1, 'BEST_ID wurde geändert',
  5             'decode(best_id, sap_id, 0, 1) > 0', 1),
  6           (2, 'BEST_PARENT_ID wurde geändert',
  7             'decode(best_parent_id, sap_parent_id, 0, 1) > 0', 2),
  8           (3, 'BEST_NAME wurde geändert',
  9             'decode(best_name, sap_name, 0, 1) > 0', 3)
 10         ) a (or_id, or_rationale, or_condition, or_sort_seq);
View wurde erstellt.
```

Listing 2: Regeln des »Expertensystems« als View

```
SQL> select coalesce(sap_id, best_id) best_id,
  2         sap_parent_id best_parent_id, sap_name best_name,
  3         case when best_id is null then 'I'
  4              when sap_id is null then 'D'
  5              else 'U' end best_dml_action,
  6         or_rationale best_rationale
  7    from bestand
  8    full join sap
  9      on best_id = sap_id
 10    join orga_regel_v
 11      on (or_id = 1 and decode(best_id, sap_id, 0, 1) > 0)
 12      or (or_id = 2 and decode(best_parent_id, sap_parent_id, 0, 1) > 0)
 13      or (or_id = 3 and decode(best_name, sap_name, 0, 1) > 0);

BEST_ID BEST_PARENT_ID BEST_NAME         B BEST_RATIONALE
------- -------------- ----------------- - ------------------------------
    234            345 Unterabteilung B2 U BEST_PARENT_ID wurde geändert
    234            345 Unterabteilung B2 U BEST_NAME wurde geändert
    567            345 Unterabteilung B3 U BEST_NAME wurde geändert
    123                                  D BEST_ID wurde geändert
    123                                  D BEST_NAME wurde geändert
    678            345 Unterabteilung B4 I BEST_ID wurde geändert
    678            345 Unterabteilung B4 I BEST_PARENT_ID wurde geändert
    678            345 Unterabteilung B4 I BEST_NAME wurde geändert
```

Listing 3: Keimzelle eines Expertensystems

Die Daten, die in dieser Anwendung analysiert werden müssen, nenne ich „Page-State“ in Anlehnung an den „Session-State“. Im konkreten Beispiel sind im Page-State die aktuell eingefügten Werte der Anwendungsseite sowie einige Informationen zum Ereignis, wie zum Beispiel zum Seitenelement, das die Anfrage ausgelöst hat, enthalten. Die Daten des Page-State werden in einer Zeile einer SQL-Abfrage zur Verfügung gestellt, die Spaltenbezeichner entsprechen den Seitenelement-IDs. Eine Tabelle enthält für diese Anwendungsseite Regeln, die die Entwickler formuliert haben. Sie verwenden dabei eine Syntax, die einer `where`-Klausel entspricht und auch Funktionsaufrufe enthalten kann.

Im Beispiel könnte die Regelbedingung lauten:

```
my_pkg.ist_bonusberechtigter_
beruf(P1_JOB_ID) = 'Y'.
```

Die Entscheidungstabelle liefert die am besten passende Regel-ID für eine gegebene Anwendungssituation zurück und führt die Dinge aus, die mit dieser Regel als Reaktion verknüpft wurden.

Das Expertensystem ist hier erforderlich, um die Regel zu identifizieren, deren Aktivitäten nun ausgeführt werden sollen. Wäre diese Tabelle nicht mit der Erweiterung dieses Artikels ausgestattet, wäre die Verwendung in diesem Zusammenhang nicht möglich.

Betrachten wir die Flexibilität dieses Ansatzes:

1. Die Daten, die im Page-State hinterlegt werden können, sind durch die APEX-Anwendungsseite definiert und werden so letztlich vom Anwendungsentwickler bereitgestellt.

2. Die Regelbedingungen entsprechen einfachen Ausschnitten aus `where`-Klauseln und sind daher in einer Syntax zu schreiben, die der Entwickler kennt, zumal er auf die Werte des Page-States zugreifen kann wie auf eine Spalte einer Tabelle. Auch diese Regeln kommen durch die Anwendungsentwickler.
3. An die Regeln kann der Entwickler eine Reihe von Aktivitäten hängen, die wiederum steuern, was auf der Anwendungsseite geschieht, ähnlich den dynamischen Aktionen, die deklarativ in APEX formuliert werden können. Diese Regeln können client- und/oder datenbankseitige Aktionen ausführen.

Das gesamte System kommt ohne Änderung am Expertensystem aus, es wird lediglich durch die Anwendungsentwickler auf Basis der Anwendungsdaten konfiguriert, ohne dass hierfür ein spezielles Wissen erforderlich wäre. Aus den Daten, die die Entwickler bereitstellen, berechnet ein Codegenerator die konkrete Entscheidungstabelle, die nun – je nach Inhalt des Page-States – die passende Antwort generiert.

Es ist sogar ein rekursiver Ansatz möglich: Ändert eine Regel, die als Antwort auf eine Anwendungssituation ausgeführt wird, den Page-State (indem es den Wert eines Eingabefeldes neu berechnet), reicht es, die „Regelmaschine“ erneut anzuwerfen: Da nun ein anderer Page-State als Grundlage verwendet wird, wird das System eine weitere Antwort generieren. Alle Aufgaben, die sich aus den Antworten für die clientseitige Reaktion ergeben, werden zusammengefasst und an die APEX-Anwendungsseite zurückgeschickt.

Ein Seiteneffekt dieses Verfahrens hat mich besonders gefreut: Eine Analyse der Bedingungen gegen die Metadaten der APEX-Anwendungsseite ermittelt sehr schnell, für welche Anwendungselemente Regeln hinterlegt wurden. Da sich eine Antwort auf eine Anfrage nur dann ergeben kann, wenn der Wert eines dieser Seitenelemente verändert wurde, kann aus dieser Information abgeleitet werden, welche Inhalte in den Page-State gehören und auf welche Änderungen der Anwendungsseite das System reagieren muss. Als Teil der Speicherung des Regelsatzes ergibt sich so auch ein Initialisierungscode, der beim Laden der Anwendungsseite alle Seitenelemente mit einem JavaScript-Observer versorgt, sodass das System bei Änderungen an diesen relevanten Seitenelementen automatisiert informiert wird. Der Kreis schließt sich: Es sind die Daten – sowohl die Metadaten der APEX-Anwendungsseite als auch die Daten, die durch den Entwickler in die Entscheidungstabelle eingefügt wurden –, die nicht nur die korrekte Reaktion auf eine Anwendungssituation berechnen, sondern zudem auch die Administration des Systems und die Interaktion mit der Anwendungsseite automatisiert bereitstellen können. Die Daten kontrollieren den gesamten Fluss, ohne weitere Programmierung und ohne, dass der Anwendungsentwickler auf Seite der APEX-Anwendung irgendeine Konfiguration dieser Regelmaschine vornehmen muss.

Und mit dieser Einsicht sind wir am Kern dieser Artikelserie: Es ist die Kraft der Daten, die hier programmiert, das Expertensystem flankiert lediglich die Entscheidungen, die sich durch die Daten ergeben: Wir programmieren mit Daten.

Jürgen Sieben
j.sieben@condes.de

Oracle Datenbanken Monthly News

Auf dem deutschsprachigen Oracle-Blog ist die Januar-Ausgabe der News-Serie erschienen.

DOAG Online

Es ist wieder so weit: die neue Ausgabe ist online! Das sechsköpfige Redaktionsteam von Oracle Deutschland hat wieder Neuigkeiten rund um die Oracle-Datenbank für On-Premises und Cloud-Installation zusammengestellt.

Alles wird wieder in einem Video präsentiert.

In der aktuellen Ausgabe wird wieder ein zusätzliches Quick Link Posting (in Englisch) zur Verfügung gestellt, um einen schnellen Zugriff auf die zugehörigen Beiträge zu gewährleisten.

https://www.doag.org/de/home/news/oracle-datenbanken-monthly-news-54/

KI und/oder Datenschutz – Gesellschaft, Technik und ethische Herausforderungen

Sandra Leist, Werk3

Künstliche Intelligenz ist längst Teil des Alltags, jedoch oft unbemerkt, aber mit spürbaren Auswirkungen. Sie sortiert Informationen, priorisiert Entscheidungen und beeinflusst Prozesse, die Menschen direkt betreffen. Genau darin liegt ihre Stärke und ihr Risiko. Dieser Artikel beleuchtet, wie KI-Systeme technisch funktionieren, welche Rolle Datenqualität und Nachvollziehbarkeit spielen und warum Datenschutz und Ethik zentrale Voraussetzungen für vertrauenswürdige KI sind. Anhand realitätsnaher Praxisbeispiele und vor dem Hintergrund des europäischen Rechtsrahmens zeige ich Ihnen, wie Organisationen KI verantwortungsvoll, resilient und zukunftsfähig einsetzen können.

KI als gesellschaftliche Infrastruktur

Künstliche Intelligenz hat sich in vielen Organisationen von einem Innovationsprojekt zu einer tragenden Infrastruktur entwickelt. Sie unterstützt Entscheidungen, priorisiert Anfragen, bewertet Risiken oder prognostiziert Entwicklungen. In Unternehmen, Verwaltungen und im Gesundheitswesen werden KI-Systeme eingesetzt, um Bearbeitungszeiten zu verkürzen, Ressourcen effizienter zu verteilen oder Handlungsempfehlungen auszusprechen.

Damit verlässt KI den rein technischen Raum und wirkt unmittelbar auf Menschen, Organisationen und gesellschaftliche Strukturen. Während klassische Software deterministisch arbeitet, beruhen KI-Systeme auf Wahrscheinlichkeiten. Sie treffen keine Entscheidungen im rechtlichen Sinne, beeinflussen diese jedoch maßgeblich, zum Beispiel wenn ein System Vorschläge priorisiert oder Risiken klassifiziert *[1]*.

Ein praktisches Beispiel zeigt sich im Kundenservice: KI-gestützte Ticket-Systeme entscheiden, welche Anfragen zuerst bearbeitet werden. Wird ein Anliegen systematisch niedriger priorisiert, kann dies faktisch zu einer Benachteiligung führen, obwohl formal keine Entscheidung „gegen" eine Person getroffen wurde.

Datenschutz und Ethik sind in solchen Szenarien keine Zusatzanforderungen, sondern zentrale Voraussetzungen für Nachvollziehbarkeit und Akzeptanz.

Technische Grundlagen: Wie KI-Systeme Entscheidungen vorbereiten

KI-Systeme unterscheiden sich grundlegend von klassischer Softwareentwicklung. Statt fest definierter Regeln lernen Modelle aus historischen Daten. Üblicherweise lassen sich drei Phasen unterscheiden: Datensammlung und -aufbereitung, Training des Modells sowie die Anwendung auf neue Daten.

Bereits in der Datenauswahl werden zentrale Weichen gestellt. In der Praxis zeigt sich, dass Trainingsdaten häufig aus operativen Systemen stammen, die ursprünglich nicht für KI-Zwecke konzipiert wurden. Fehlende Dokumentation, uneinheitliche Datenformate oder implizite Annahmen erschweren die Nachvollziehbarkeit erheblich *[2]*.

Ein typisches Beispiel aus der Praxis ist der Einsatz von KI zur Prognose von Zahlungsausfällen. Werden historische Forderungsdaten genutzt, ohne Kontextfaktoren zu berücksichtigen, stuft das Modell bestimmte Kundengruppen systematisch als risikoreicher ein. Technisch korrekt, fachlich jedoch problematisch.

Für Entwickler bedeutet dies: Nachvollziehbarkeit entsteht nicht allein durch erklärbare Modelle, sondern durch saubere Datenpipelines, Versionierung und strukturierte Protokollierung *[3]*.

Datenqualität als Fundament und als Risiko

Daten gelten als zentraler Erfolgsfaktor für KI-Systeme. Gleichzeitig sind sie eine der größten Fehlerquellen. Daten spiegeln vergangene Entscheidungen, organisatorische Routinen und gesellschaftliche Ungleichheiten wider.

Ein klassisches Praxisbeispiel ist der Einsatz von KI-Systemen zur Bewerbervorauswahl. Werden historische Einstellungsentscheidungen als Trainingsgrundlage verwendet, lernt das Modell nicht Eignung, sondern vergangene Präferenzen. Dies kann dazu führen, dass bestimmte Profile systematisch benachteiligt werden, ohne dass dies im operativen Betrieb sofort auffällt *[4]*.

Ein weiteres Beispiel aus der Verwaltungspraxis sind KI-gestützte Prognosen zur Fallwahrscheinlichkeit bei Sozialleistungen. Solche Systeme können dazu führen, dass bestimmte Anträge häufiger überprüft oder verzögert bearbeitet werden. Diskriminierung entsteht hier nicht durch Absicht, sondern durch Datenlage und Modelllogik.

Datenschutzrechtliche Einordnung im europäischen Kontext

Die Datenschutz-Grundverordnung bildet weiterhin den zentralen Rechtsrahmen für den Einsatz von KI in Europa. Sie stellt Anforderungen an Transparenz, Zweckbindung, Datenminimierung und Fairness und begrenzt automatisierte Entscheidungen mit rechtlicher oder ähnlich erheblicher Wirkung *[5]*.

Ergänzend verfolgt die EU-KI-Verordnung einen risikobasierten Ansatz. Hochrisiko-KI-Systeme, etwa im Beschäftigungs-, Gesundheits- oder Verwaltungsumfeld, unterliegen erweiterten Pflichten zur Dokumentation, Überwachung und menschlichen Kontrolle *[6]*.

Mit dem Cyber Resilience Act rückt zudem die Sicherheit digitaler Produkte in den Fokus *(siehe Abbildung 1)*. Für KI-Systeme bedeutet dies, dass Datenschutz, IT-Sicherheit und Resilienz gemeinsam betrachtet werden müssen. Manipulierbare oder unzureichend abgesicherte Systeme gefährden nicht nur Daten, sondern auch die Verlässlichkeit automatisierter Entscheidungen *[7]*.

Ethik in der Praxis: Der Anti-Bias-Ansatz

Ethik im KI-Kontext darf nicht auf Leitbilder oder Code-of-Conduct-Dokumente reduziert werden. Der Anti-Bias-Ansatz verfolgt einen praxisnahen Zugang und setzt früh im Lebenszyklus eines KI-Systems an.

Konzeptionell bedeutet Anti-Bias:

- bewusste Zieldefinition
- kritische Auswahl und Prüfung von Datenquellen
- regelmäßige Überprüfung von Ergebnissen und Nebenwirkungen

In der Praxis zeigt sich dies beispielsweise bei KI-Systemen zur Entscheidungsunterstützung im Gesundheitswesen. Modelle, die auf unvollständigen oder einseitigen Datensätzen trainiert wurden, erkennen Risiken bestimmter Bevölkerungsgruppen schlechter. Ohne gezielte Gegenmaßnahmen entsteht eine strukturelle Benachteiligung *[8]*.

Ein wirksamer Anti-Bias-Ansatz umfasst dokumentierte Prüfprozesse, Testläufe mit alternativen Datensätzen und klar definierte menschliche Eingriffsmöglichkeiten *(siehe Abbildung 2)*.

Governance statt Vertrauen auf Zuruf

Viele Organisationen verlassen sich beim Einsatz von KI auf implizites Vertrauen in

Abbildung 1: Cyber Firewall (Quelle: Pixabay)

Abbildung 2: Menschliche Eingriffsmöglichkeiten (Quelle: Pixabay)

Technologie oder Anbieter. In der Praxis reicht dies nicht aus. Spätestens bei Datenschutzvorfällen, Sicherheitsproblemen oder behördlichen Prüfungen zeigt sich der Bedarf an formalisierter Governance.

KI-Governance bedeutet:

- klare Zuständigkeiten zwischen IT, Fachbereich und Management
- dokumentierte Entscheidungs- und Freigabeprozesse
- Integration in bestehende Datenschutz- und Sicherheitsstrukturen

Praxisnah umgesetzt bedeutet dies beispielsweise, KI-Systeme in bestehende Datenschutz-Folgenabschätzungen, Risikomanagementprozesse und Sicherheitskonzepte einzubinden *[9]*.

Mein Ausblick

Künstliche Intelligenz ist längst Teil gesellschaftlicher Realität. Datenschutz, Ethik und Resilienz sind keine Gegenspieler der Innovation, sondern ihre Voraussetzung. Wer KI-Systeme verantwortungsvoll gestalten will, muss technische Nachvollziehbarkeit, rechtliche Anforderungen und gesellschaftliche Wirkung gemeinsam denken.

Der europäische Rechtsrahmen liefert hierfür klare Leitplanken. Entscheidend bleibt jedoch die praktische Umsetzung – dort, wo sich zeigt, ob KI-Vertrauen schafft oder verspielt.

Quellen

[1] vgl. Floridi, L.; Cowls, J.; Beltrametti, M. et al. (2018): AI4People – An Ethical Framework for a Good AI Society. Springer, Cham.

[2] vgl. Russell, S.; Norvig, P. (2021): Artificial Intelligence – A Modern Approach. Pearson, London.

[3] vgl. ISO/IEC (2023): Artificial Intelligence – Risk Management Framework. ISO, Genf.

[4] vgl. O'Neil, C. (2016): Weapons of Math Destruction. Crown Publishing, New York.

[5] vgl. Europäische Union (2016): Datenschutz-Grundverordnung. Amtsblatt der Europäischen Union, Brüssel.

[6] vgl. Europäische Kommission (2024): Verordnung über künstliche Intelligenz (AI Act). EU, Brüssel.

[7] vgl. Europäische Kommission (2023): Cyber Resilience Act. EU, Brüssel.

[8] vgl. High-Level Expert Group on Artificial Intelligence (2020): Ethics Guidelines for Trustworthy AI. Europäische Kommission, Brüssel.

[9] vgl. European Data Protection Board (2024): Guidelines on Automated Decision-Making and Profiling. EDPB, Brüssel.

Über die Autorin

Ich bin externe Datenschutzbeauftragte und CEO der Werk3 für Datenschutz GmbH. Ich berate Unternehmen, Organisationen und Einrichtungen an der Schnittstelle von Technik, Recht und Ethik – mit Schwerpunkt auf Datenschutz, KI-Governance und Informationssicherheit. Ich verbinde regulatorische Anforderungen mit technischer Praxis und organisationaler Realität. Ziel meiner Arbeit ist es, digitale Innovation verantwortungsvoll, nachvollziehbar und nachhaltig zu gestalten.

Sandra Leist
mail@datenschutz-werk.de

2026

DOAG

Konferenz + Ausstellung

Die ORACLE Anwenderkonferenz

Nürnberg | 17. – 20. Nov.

#DOAG2026

anwenderkonferenz.doag.org

BEST OF DOAG ONLINE

Eine Auswahl der besten DOAG News von Oktober 2025 bis Januar 2026

DOAG VOICES – Warum IT ohne Kultur nicht funktioniert

Was macht gute IT-Arbeit wirklich aus? Diese Folge von DOAG VOICES – in der Edition People mit Dave König und seinem Gast Dr. Jürgen Baier – zeigt, warum Motivation, Haltung und Kultur entscheidender sind als jedes Tool.

DOAG Datenbank Kolumne: Proxy Authentication – ein Lösungsansatz für einige Securitythemen

Eines der größten Securitythemen ist die Benutzeridentifizierung, speziell wenn es um den Bereich Datenbank-Administration beziehungsweise "gewachsene" Applikationen geht.

DOAG VOICES – Sicherheit neu gedacht: Cloud, KI und die Zukunft europäischer IT

Souveräne Cloudlösungen gewinnen rasant an Bedeutung. Im Gespräch mit Martin Mangold zeigt sich, wie europäische Security-Ansätze heute global mithalten – und warum Architekturentscheidungen darüber entscheiden, ob Cloud wirklich souverän ist.

DOAG VOICES – KI in der Softwareentwicklung: Zwischen Code, Kultur und Community

In dieser Ausgabe von DOAG VOICES in der Edition FutureAI trifft Host Dave König auf Tobias Struckmeier, Principal Engineer bei adesso. Gemeinsam beleuchten sie, wie KI den Entwickleralltag verändert – vom Coding über Testing bis zur Lernkultur im Unternehmen.

DOAG Datenbank Kolumne: Oracle-Datenbank-Lizenzierungsmetriken Enterprise Edition

Dies ist der zweite Artikel einer Mini-Serie zum Thema Oracle-Datenbank-Lizenzierungsmetriken für Standard- und Enterprise-Edition.

DOAG VOICES – Gesetze mit Sinn: Warum KI-Regulierung neu gedacht werden muss

Künstliche Intelligenz fordert nicht nur Entwickler heraus, sondern auch Gesetzgeber. Wie viel Regulierung ist sinnvoll – und wann wird sie zur Innovationsbremse? Dr. Benjamin Linnik und Baltasar Cevc geben Antworten in der neuen Folge des Podcasts DOAG VOICES.

Wir begrüßen unsere neuen Mitglieder

Natürliche Mitglieder:

- Stefan Radau
- Jonas Probst
- Martin Hanke
- Eric Amann
- Susanne Fertmann
- Jens Kaschuba
- Sebastian Wallek
- Jonas Janz
- Michael Mühlbeyer
- Jessica Steger
- Oliver Schumann

Korporative Mitglieder:

- .riess engineering europe gmbh, Repräsentant Wolfgang Behr
- INFORMATICS Healthcare GmbH, Repräsentant Peter Altreiter
- OG Consultancy Services GmbH, Repräsentant Markus Schmelzle
- Blackwood Digital GmbH, Repräsentant Claus Engel
- Krankenhaus Barmherzige Brüder Regensburg, Repräsentantin Antje Schoppa

Termine

März 03

10. - 12.03.2026
JavaLand 2026
Zwei ereignisreiche Konferenztage mit anschließendem Schulungstag rund um das Java-Ökosystem
Rust, Europa-Park

12. - 13.03.2026
DevLand 2026
Digitale Erlebniswelt für Software-Architekten, KI-Experten, Cloud-Strategien und Young Professionals, die gemeinsam die Zukunft moderner Softwarelandschaften entwerfen
Rust, Europa-Park

Mai 05

18. - 19.05.2026
DOAG 2026 Datenbank mit Cloud Infrastructure
Konferenz rund um die Oracle-Datenbank und Cloud Infrastructure
Soltau, Heide Park

18. - 20.05.2026
APEX connect 2026
Konferenz mit zahlreichen Vorträgen und Workshops zu den Themen APEX, PL/SQL, JavaScript und Solutions
Soltau, Heide Park

20. - 22.05.2026
CloudLand 2026 -
Das Cloud Native Festival
Community-Veranstaltung mit den Themen Cloud-native Software Engineering, Architecture, DevOps, Public Cloud, Organization, Culture, AI, ML, Security und vielen mehr
Soltau, Heide Park

September 09

22.09.2026
Java Forum Nord 2026
Die Community Konferenz der Java User Groups aus dem Norden
Hannover

Impressum

Red Stack Magazin wird gemeinsam herausgegeben von DOAG 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 die Community-Publikation für angewandte Informations- und Kommunikationstechnologie (ITK) im Raum Deutschland, Österreich und Schweiz. Es setzt bewusst auf Technologieoffenheit mit Blick auf Anwendung und IT-Innovationen.

Es bildet die Interessensschwerpunkte der Anwenderinnen und Anwender ab – von Cybersicherheit bis Datenschutz, von Datenbank und Development über Data Analytics bis Digitalisierung, von Cloud und Infrastruktur über Künstliche Intelligenz bis Open Source und Soft Skills – vermittelt praktisches Wissen und fördert den Know-how-Transfer und die Netzwerkbildung zwischen den Leserinnen und Lesern.

Die Inhalte des Red Stack Magazin werden von ausschließlich ehrenamtlichen Autorinnen und Autoren eingereicht und von der Redaktion aufbereitet.

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. DOAG e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. DOAG e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Björn Bröhl

Redaktion:
Sitz: DOAG Dienstleistungen GmbH
(Anschrift s.o.)
ViSdP: Fried Saacke
Redaktionsleitung Red Stack Magazin:
Martin Meyer
Kontakt: redaktion@doag.org

Autorinnen und Autoren dieser Ausgabe
(in alphabetischer Reihenfolge):
Maria Colgan, Amin Farvardin,
Markus Flechtner, Dagmar Förster,
Daniel Horwedel, Sandra Leist,
Ilgar Machmudov, Martin Meyer,
Thomas Petrik, Christian Pfundtner,
Lena Rudenko, Philipp Salvisberg,
Marcus Schröder, Jürgen Sieben,
Andreas Ströbel, Christian Trieb,
Wolfgang Wilhelm, Rainer Willems

Titel, Gestaltung und Satz:
Diana Tkach
DOAG Dienstleistungen GmbH
(Anschrift s.o.)

Fotonachweis:
Titel: © KI Bildgenerator | *www.freepik.com*
S. 08: © sman_5 | *www.pixabay.com*
S. 12: © geralt | *www.pixabay.com*
S. 18: © garten-gg | *www.pixabay.com*
S. 24: © awadpalestine | *www.pixabay.com*
S. 31: @ lillolillolillo | *www.pixabay.com*
S. 38: © StockSnap | *www.pixabay.com*
S. 43: © Twarezak | *www.pixabay.com*
S. 52: © Chris18769 | *www.pixabay.com*
S. 55: © cocoparisienne | *www.pixabay.com*
S. 61: © geralt | *www.pixabay.com*
S. 62: © stevepb | *www.pixabay.com*
S. 67: © freepik | *www.freepik.com*
S. 74: © Alexas_Fotos | *www.pixabay.com*
S. 80: © geralt | *www.pixabay.com*
S. 84: © geralt | *www.pixabay.com*

Anzeigen:
sponsoring@doag.org

Mediadaten und Preise:
www.doag.org/go/mediadaten

Druck:
WIRmachenDRUCK GmbH,
www.wir-machen-druck.de

Inserentenverzeichnis

SOUG Day Bern

11. März 2026

ORACLE-WISSEN, DAS WIRKT.
Von den innersten Mechaniken der Oracle-Datenbank über moderne Security- und Multicloud-Architekturen bis hin zu APEX, Autonomous DB und den neuesten On-Premises-Innovationen:

Der SOUG-Day in Bern bringt geballte Expertise, echte Praxisbeispiele und strategische Einblicke auf eine Bühne. Ob Datenbank-Expert*in, Architekt*in, Entwickler*in oder Entscheider*in – dieser Tag liefert Orientierung, Know-how und Austausch auf Augenhöhe. U.a. mit Stefan Oehrli, Hans Viehmann, Flora Barriele, Martin Berger, Daniel Overby Hansen, Philipp Salvisberg und vielen mehr.

Ein Tag. Eine Community. Maximale Relevanz.
☞ Jetzt Platz sichern und Teil des SOUG-Days in Bern werden. Anmeldung unter **soug.ch**

SOUG

Many thanks to **«Die Mobiliar»** for hosting us.

PROGRAMM ONLINE

12. + 13. MÄRZ 2026

»DEV LAND«

WWW.DEVLAND.EU

#DEVLAND

THE NEXT GENERATION OF DEVELOPMENT

EUROPA-PARK IN RUST

ARCHITECTURE AVENUE
AI VALLEY
CLOUD CLIFFS
DATA DOCKS
CAREER COAST

nur 95 € (ZZGL. MWST.)

STARTER PASS