ORACLE MAGAZINE

JULY/AUGUST 2019

The Infrastructure for Speed

The highest-performing business apps and processes require the highest-performing infrastructure. Read how Oracle's next-gen cloud can turbocharge your operations.

CLOUD MEANS CONNECTIONS

AUTONOMOUS DATABASE FOR ALL

VOICE IN THE MACHINE

ORACLE®

Connect

with Trusted Partners in Tax Technology

For over 25 years, Oracle and Vertex have worked closely to deliver trusted tax technology to companies across the globe.

Learn More

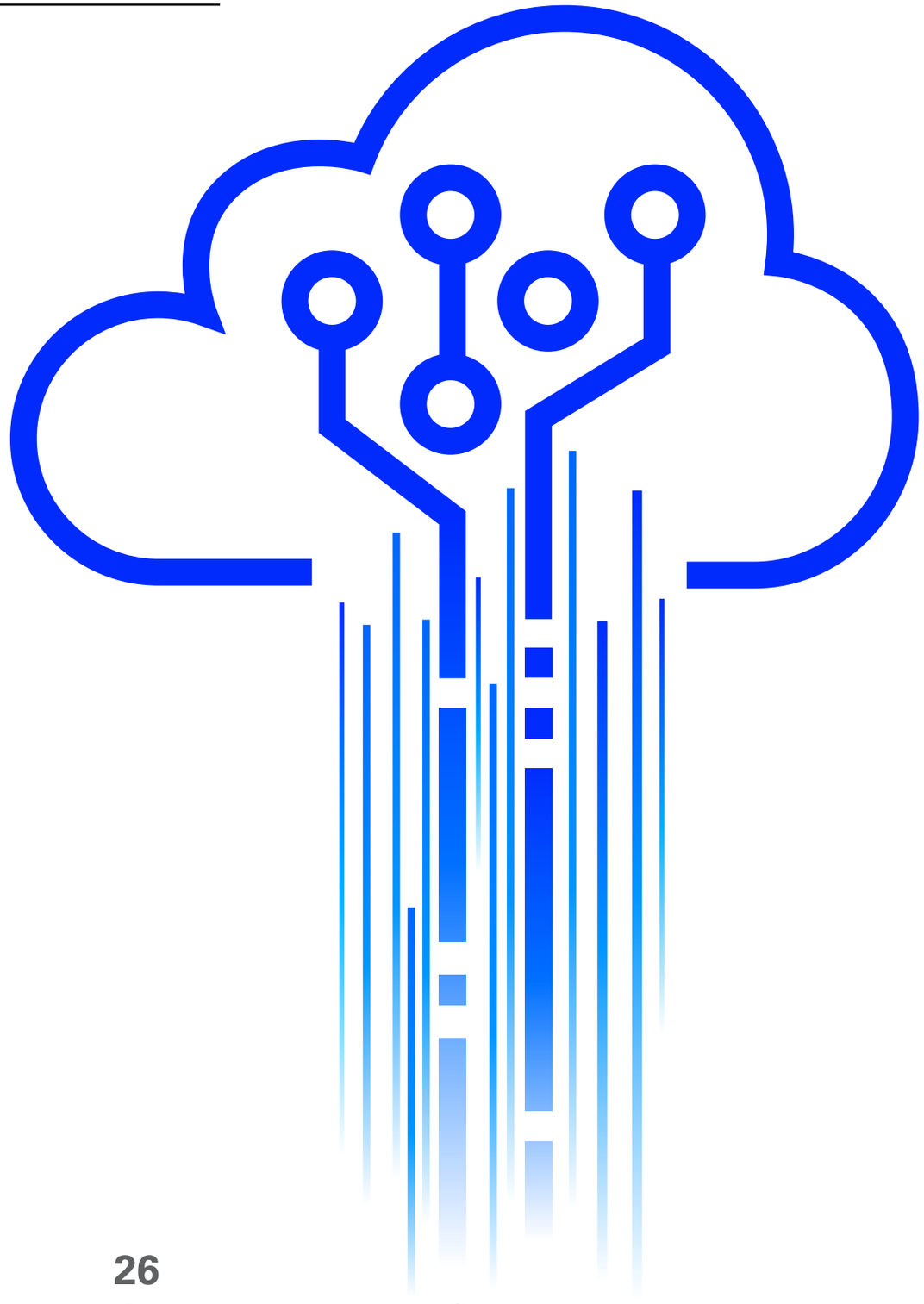
Visit us at Oracle OpenWorld booth #1917

ORACLE Platinum Partner Cloud Standard

VERTEX®

CONTENTS

ORACLE MAGAZINE



FEATURES

UP FRONT

8 Interview

CONTENTS

COMMUNITY

23 Peer-to-Peer

TECHNOLOGY

ORACLE MAGAZINE

EDITORIAL

Editor in Chief Tom Haunert

Editorial Director Chris Murphy

Senior Managing Editor Jan Rogers

Contributing Editor and Writer Blair Campbell

Copy Editors Eva Langfeldt, Karen Perkins

PUBLISHING

Publisher and Audience Development Director Karin Kinnear

ADVERTISING SALES

Tom Cometa +1.510.339.2403

Mailing-List Rentals Contact your sales representative

SUBSCRIPTION INFORMATION

Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICE

Omeda Communications

PRIVACY

Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or email address not be included in this program, contact Customer Service at oracle@omeda.com.

Address the ELEPHANT IN THE ROOM

Bad address data costs you money, customers and insight.

Melissa's 30+ years of domain experience in address management, patented fuzzy matching and multi-sourced reference datasets power the global data quality tools you need to keep addresses clean, correct and current. The result? Trusted information that improves customer communication, fraud prevention, predictive analytics, and the bottom line.

- Global Address Verification
- Digital Identity Verification
- Email & Phone Verification
- Location Intelligence
- Single Customer View

+ JD Edwards® EnterpriseOne

+ Oracle® E-Business Suite

+ Oracle® Peoplesoft®

See the Elephant in Your Business - Name it and Tame it!

melissa

www.Melissa.com | 1-800-MELISSA

Free API Trials, Data Quality Audit & Professional Services.

Tom Haunert

What's Next?

There's a lot of technology out there, and it's time we stopped limiting our coverage of it.

We've published *Oracle Magazine* on multiple digital platforms over the last four years, catering to reader feedback, but we've also missed some things along the way. The good news is that we've got a plan to fix what we missed, and we're confident it's a win for you.

MORE, AND MORE RESPONSIVE

Here's the plan: We've decided to go all-in on our responsive HTML platform and format. This is the last bimonthly issue of *Oracle Magazine*—we'll post future content to our website at blogs.oracle.com/oraclemagazine *weekly*, with coverage that's more timely and even more compelling than the content we're producing today. In the process, we're going to double down on our popular hands-on how-to content while initiating in-depth coverage of Oracle software-as-a-service (ERP, HCM, CX, and so on) products.

You will find all of that frequent, technical, SaaS, and responsive content on our website. Subscribers: Please be sure to open our weekly *Cloud Leader* emails for the latest website and content updates. (Current *Oracle Magazine* subscribers will be subscribed to *Cloud Leader* automatically.) Email us at opubedit_us@oracle.com if you have any questions or want to offer ideas on how we can serve you better.

WHY WE'RE CHANGING

Historically, planning our magazine material on a bimonthly schedule forced us to defer coverage of a lot of technology areas to "the next issue," given issue theme and space constraints. And for a bimonthly technology magazine, already too infrequent in the fast-paced tech world, deferring content by two months has been far from ideal.

Furthermore, when we stopped printing, in 2015, we committed to producing the magazine in a fixed-dimension page format attuned to desktop screens. But we weren't ready for the rise—let's just call it domination—of mobile phones for reading and watching almost every form of digital content. The desktop-oriented *Oracle Magazine* page format simply isn't readable on a mobile device.

Since 2017, *Oracle Magazine* content has been available in responsive HTML at blogs.oracle.com/oraclemagazine, and that responsive content reads well on all screen sizes, from desktop to phone to anything in between.

Tom Haunert,
Editor in Chief

Emails and posts received by *Oracle Magazine* and its staff may be edited for length and clarity and may be published in any medium. We consider any communications we receive publishable.

PHOTOGRAPH BY **BOB ADLER/GETTY IMAGES**

NEXT STEPS

READ *Oracle Magazine*. **SUBSCRIBE** to *Cloud Leader*.

“Businesses don’t need infinite internet paths—they need to look at connections to their applications and services in terms of reliability, flexibility, and security,” says Charlie Baker, senior director of product management and strategy at Oracle.

Cloud Means Connections

What kind of network is the cloud, and what kinds of cloud networking do you need for the best security, performance, and reliability? **BY TOM HAUNERT**

Maybe you understand when a web page takes time to load because the page is hosted thousands of miles away, but how understanding are you when your enterprise cloud applications seem slower than before? As your company combines on-premises applications and databases with cloud-based applications and databases, your networking challenges are changing.

Oracle Magazine sat down with Charlie Baker, senior director of product management and strategy at Oracle, to talk cloud networking challenges, strategies for addressing those challenges, Oracle Cloud Infrastructure FastConnect, networking support for Oracle Autonomous Database, and more.

Oracle Magazine: What are the key networking challenges in cloud computing today?

Baker: It's an interesting question. When I used to train people on different ways to think about connectivity, I would usually start out with a question: "Who owns the internet?" First there would be quiet in the room. Then answers would start coming, answers like "the cable company" or "the telephone company" or "the government."

The internet is a lot of things to a lot of people, but in terms of connectivity, the internet is a relationship of the different networks that get you from point A to point B. And depending on how an operator or a customer or a business accesses information or applications across the internet, there is almost an infinite number of paths to take to get from point A to point B—because that's the purpose of the internet. But businesses don't need infinite internet paths—they need to look at connections to their applications and services in terms of reliability, flexibility, and security. Providing a reliable, flexible, and secure cloud network is the key challenge, and the challenge for cloud vendors is to make it easy to get that up and running.

Depending on how important each of those connection qualities is, businesses will make different choices for how to connect people to where they need to go, how to connect two applications together, how to connect backup systems, and so on to be able to have everyone and every system operate in this environment that we call the cloud-enabled world.

Oracle Magazine: What are the strategies for addressing these cloud networking challenges?

Baker: Cloud customers must choose who and what can access an application. They need to identify the users, offices, and data centers that need cloud network and application access. And the key enterprise workloads that we see in Oracle Cloud Infrastructure typically need to connect from customer data centers to cloud data centers directly—with no one else accessing the data flowing between the locations. They need a reliable, secure connection between their own data centers and the cloud.

"Oracle Cloud Infrastructure FastConnect . . . is like having your data center extended with a wire to Oracle Cloud."

When large enterprises are connecting application components across data centers, they need to look at the latency in the connections between pieces of an application and the effect of that latency on the performance of the application. For example, if I run in a hybrid cloud environment—I'm connecting the cloud to my data centers—what kind of latency can my application tolerate when it's split across data centers? Can I have my database in one data center and my application in another? What does the architecture of the solution look like?

For internet-facing applications, security and reliability drive the requirements. It's important to have an extremely secure model that includes a web application firewall, load balancers in front of your application, and an overall high-availability solution. And to address the next level of security and latency, private and dedicated connectivity is critical, whether it's connecting remote offices via virtual private networks (VPNs) or different data centers.

For the most-secure transactions, cloud applications will typically require a VPN. To connect a customer network or data center directly to a cloud network and reduce latency, however, the solution is often dedicated, direct network connectivity. Oracle Cloud Infrastructure FastConnect, for example, offers an easy and elastic way to create a dedicated, private connection from customer and partner networks to Oracle Cloud with higher bandwidth than internet-based connections. Oracle Cloud Infrastructure FastConnect provides the most reliable and secure way to connect into

"Oracle Cloud Infrastructure is built with the requirements of the Oracle Autonomous Database services at the forefront," says Charlie Baker, senior director of product management and strategy at Oracle.

Oracle Cloud—it's like having your data center extended with a wire to Oracle Cloud. And Oracle works with dozens of Oracle FastConnect connectivity partners that can help deliver to Oracle Cloud customers dedicated connectivity that does not use the internet.

Oracle Magazine: Oracle Autonomous Database runs on Oracle Cloud Infrastructure only. How does Oracle Cloud Infrastructure networking support and work with Oracle Autonomous Database?

Baker: The Oracle Cloud Infrastructure and Oracle Autonomous Database development teams work together closely. There are frequent discussions between the teams on the requirements for the

network, services, features, and how to make everything work seamlessly for the user. We get those things right together.

Oracle Autonomous Database services are the best in the industry, and they take advantage of the network model, security, compute power, and scalability of Oracle Cloud Infrastructure. And Oracle Cloud Infrastructure is built with the requirements of the Oracle Autonomous Database services at the forefront.

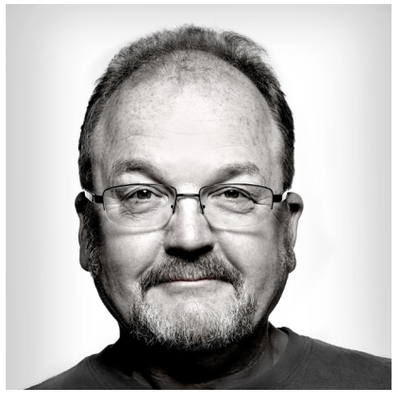
PHOTOGRAPHY BY
JASON GROW/GETTY IMAGES

NEXT STEPS

LEARN more about Oracle Cloud networking.

Oracle Cloud Infrastructure FastConnect.

READ about the Microsoft and Oracle cloud interoperability partnership.



By **Bob Rhubart**

Connecting the Dots

A developer's journey with Oracle Groundbreaker Ambassador Mark Heckler

"A tiny little map dot of a town." That's how Oracle Groundbreaker Ambassador Mark Heckler describes Sumner, Illinois, the town where he grew up. It was there, as a high school student, that he first developed an interest in computers.

For Heckler, now a principal technologist/Spring developer advocate with Pivotal, programming computers was a way to exercise control. "So many times, we operate within boundaries. We're told to do certain things, and we do them, and sometimes we are given latitude. To program a computer, to write a program, actually means you're determining what boundaries exist, if any. So it's an incredibly elevating, liberating experience,

Oracle Groundbreaker Ambassador Mark Heckler programs for control and embraces the business of technology.

because all of a sudden, you're the one who decides if that wall should be there or if you'll just bust down all the walls and go out and make something happen in that particular way that no app currently is fulfilling. So you determine where you go. It's empowering. You define what you want to have done, and the computer does it—that is, in the best of circumstances, of course. There are many failed efforts that may lead up to that, but you dictate the rules."

As a teenager, Heckler engaged in early programming efforts that grew from simple, "Hello, World"–level experiments. "In early years, I started playing with music and some graphics, a little bit of gaming, but nothing terribly sophisticated," he explains.

The local high school offered no computer classes, so Heckler grabbed every bit of available material he could find either online or in books. Lincoln Trail College, in nearby Robinson, Illinois, offered more resources. "I took all the programming courses that I could take," Heckler says. "The courses were listed under the mathematics program, but many of them were programming."

Heckler went on to attend Southern Illinois University in Edwardsville, where he earned a bachelor's degree in information technology and business and an MBA with a focus on finance. "I wanted to understand more of the business side," Heckler says.

"I felt like the MBA would be really useful and helpful to me in understanding

RECOGNITION

The Oracle Groundbreaker Ambassador program recognizes modern experts who blog; write articles; and give presentations on topics such as containers, microservices, SQL, NoSQL, open source technologies, machine learning, and chatbots. Learn more, and follow the Oracle Groundbreaker Ambassadors.

how the rest of the business works and how to convert those higher-level strategies and tech business tactics into actual code," Heckler explains.

For a time, Heckler wrote his code in C++. "I really didn't see anything terribly wrong with that," he says. "But when I saw Java in action and its garbage collector—this thing that meant you didn't have to allocate free memory for everything you did—at first I thought it was a little crazy." But not for long.

Although he found the initial versions of Java to be slow, garbage collection was a major temptation. "So much of your time is spent avoiding or finding and addressing memory leaks," Heckler says. "I felt that if they could get the performance under control, it would be an incredible language and environment in which to run applications. And then, of course, that's exactly what happened."

Now a Java Champion as well as an Oracle Groundbreaker Ambassador, Heckler finds his attention drawn to security as well as the Internet of Things.

"I got my first taste of that several years ago, writing code for an Arduino. Very simple, very limited in what it can do," Heckler says. "But what it did, it did exceedingly well. And then I started looking at 'Hey, can I plug that in here and there?'"

Heckler appreciates the process by which interaction feeds innovation. "One of the things I love about being a developer advocate is that no matter how I might envision how Spring data or Spring security might be used, when I'm out speaking with people, I'm constantly being challenged," he explains. "Each of us has in our mind what the valid use cases are for a piece of software or a component or a library. But when you talk to somebody who's using it differently, you realize, 'Oh, that's still a valid use case.' That expands my mind and expands the valid use cases and 'Wow, I see how that may be a problem using it here.' We need to accommodate that.

That's challenging and interesting."

Heckler now lives in Godfrey, Illinois, a dot on the map a scant 160 miles from Sumner but light-years away from the early attempts at "Hello, World." These days that greeting means so much more.

Oracle Architect Community Manager Bob Rhubart is the host-engineer/producer of the Oracle Groundbreakers Podcast series; produces the 2 Minute Tech Tip video series; and interviews technology experts in DevLIVE videos recorded at Oracle events.

PHOTOGRAPHY BY
BOB ADLER/GETTY IMAGES

NEXT STEPS

READ the Mark Heckler Groundbreaker Ambassador profile.

WATCH Mark Heckler on drones at the Java Hub.

Machine Learning Is the Ultimate Productivity Hack

Heli Helskyaho's tips for teaching yourself ML

By **Alexandra Weber Morales**

You might think of machine learning (ML) as a rarified specialty. Heli Helskyaho argues that you can't afford *not* to understand ML.

"First of all, machine learning *is* productivity, because you have so much data, you can't manually handle it all. Data is the key for making right decisions," the CEO of Miracle Finland Oy, an IT consultancy, says. "Right kind of data, right decisions: That's the equation here. You will not be able to work with data in any way—productive or not—without machine learning."

Oracle ACE director, EMEA Oracle User Group community ambassador, and author Heli Helskyaho sees machine learning as opportunity and productivity.

ML IS NOT IF-ELSE

Not surprisingly for someone with a background in databases, Helskyaho, the author

“What computers cannot do is understand the world.”

— Heli Helskyaho, Community Ambassador

of *Oracle SQL Developer Data Modeler for Database Design Mastery* (Oracle Press 2015), focuses on big data when thinking about ML. She tells a story of visiting a company and seeing its systems shutting down with no explanation. Helping it discover why this was happening was a simple question of analyzing temperature data with ML, she says, which discovered that on the rare occasion when a certain door was left open, the cold air would kill a particular machine and cause a cascading outage.

ML is not for answering if-else questions, she says. “Machine learning is for something you have no way to predict yourself. ML is, ‘I have no idea what might happen, but whatever happens, I need to be prepared. The world is changing, so I will adapt to that.’”

HOW TO GET STARTED

Understanding the plethora of ML terminology is the first step, Helskyaho says, noting that her popular presentations explain what these words mean and how they relate to each other. “You start out very confused, because people are using different words with the same meaning and that can sometimes mislead you.

“If you are a developer like me, the next step is choosing the programming language you want to start with, because there’s no point to starting with all of them. Use skills you already have, and add something new,” she says. Although Python is probably the most common language for working with ML, “if you’ve been using R, SQL, C++, Java—start with that.”

Next, ask about algorithms, models, and how to prepare the data. “I think 80% of ML work is preparing the data, because the data is usually not the best quality,” Helskyaho says. Using a drag-and-drop tool such as Oracle SQL Developer enables you to build a real-world model and experiment with neural networks, regression, anomaly detection, and classification algorithms.

Starter projects might be trying to

predict tomorrow's temperature or next week's stock prices, or checking whether a person is creditworthy or not.

AVOIDING TIME SINKS

As in any educational journey, you can run into dead ends. In ML, Helskyaho says, the biggest risk is bad data: "You think the data you are using is good quality, but if you just start using it without checking it, you find your predictions are very strange," says Helskyaho. "You start asking, 'What did I do wrong?' You end up realizing the data was rubbish. I see that quite a lot."

It's also important to remember that as ML becomes more commonplace, it doesn't require a scary amount of math and statistics to get started. What is necessary, however, is an understanding of the domain-specific questions that ML can answer.

"What computers cannot do is understand the world," Helskyaho says. "You understand your business, and if the machine learning brings you new knowledge, *you* understand if it's valuable or not. It's a team play, not just one skill set and one person. You need a team to make it really profitable for your company."

Alexandra Weber Morales is Oracle's director of developer content.

PHOTOGRAPHY BY
ORACLE DIGITAL MEDIA PRODUCTION

NEXT STEPS

FOLLOW the Oracle Developers Blog.

Autonomous in the Mix

As autonomous databases become part of large database estates, monitor and manage them with Oracle Enterprise Manager.

By **Jeff Erickson**

At a recent Customer Advisory Board meeting on the Oracle headquarters campus, product managers listened to power users of Oracle Enterprise Manager as they shared their complex use cases and reacted to product roadmaps for Oracle's flagship IT monitoring and management tool.

One thing those product managers heard loud and clear was that customers want to leverage Oracle Autonomous Database as part of their broader database fleets and that they're looking to Oracle Enterprise Manager to make things work seamlessly.

"Autonomous is a good solution for us, because it allows us to have power on demand," says Marco Bettini, of Italy's Bridge Consulting, which manages IT for several of the world's most renowned Italian fashion brands and retailers. When those customers offer promotions, "we need to scale up quickly and offer them performance when they need it," says Bettini. "When they're not, autonomous allows us to scale down and save them money."

Another potential benefit of Oracle Autonomous Database, says Bettini, a Bridge Consulting founding partner, is that the data experts on staff can work less on managing database instances

and spend more time working to improve their clients' applications.

Bridge Consulting chose Oracle Autonomous Data Warehouse as its first foray into autonomous databases to bring analytics workloads online quickly. Bridge plans to follow with Oracle Autonomous Transaction Processing for testing and development and eventually for mission-critical workloads, says Bettini. "We can start creating autonomous databases and aligning them with our on-premises ones, using a tool [to replicate and integrate transactional data] like Oracle GoldenGate," he says.

What the consultants at Bridge really want is *complete* integration between Oracle Enterprise Manager and the autonomous databases, adds Simone Traversari, a solution architect and engineered systems specialist at the company. "We use Oracle Enterprise Manager to monitor the customer's architecture and report on the system performance and uptime and available space and query speed." With autonomous databases, "we want to be able to see what's going on. We want to be able to see inside the resource manager and report to our clients" with the same level of detail they are used to getting from Oracle Enterprise Manager.

Oracle product managers at the Customer Advisory Board meeting were happy to report that the recent Oracle Enterprise Manager release, Oracle Enterprise Manager 13.3 PG, is designed to give users such as Traversari a dashboard for his database fleet, including on-premises, cloud, and autonomous databases. Bridge Consulting and other customers got more good news on advanced management capabilities for Oracle Database 19c, Oracle Exadata X7/X8, and Zero Data Loss Recovery Appliance.

Bettini and Traversari are looking forward to the new release to help meet expectations: "Our customers don't see our work, so they don't care

about automatic indexing or patching. They just expect that everything works fine," Bettini says. "We know how to give that to them with Oracle Enterprise Manager." Happily, that will continue as Bridge Consulting brings more autonomous databases into the mix.

Jeff Erickson is editor at large for Oracle Content Central.

NEXT STEPS

READ "The Dashboard for the Modern Database Fleet."

WATCH the "Introducing the Dashboard for the Modern Database Fleet" webcast.

GET Oracle Enterprise Manager app on Oracle Cloud Marketplace.

BY BLAIR CAMPBELL

Problem Solvers

Peers hack family reunions, decide databases aren't evil, and use cloud as an accelerator.

Niels de Bruijn

Ratingen, Germany

ORACLE ACE Director

Company/URL: Managing Technology AG

Job title: Business unit manager

Length of time using Oracle products: 18 years

What interesting challenge have you solved with Oracle solutions? My wife's family has a reunion every three years, and it always gave me a headache to try to understand the relationships of the relatives. So I obtained a 200-page Word document containing all information about the family in an unstructured way. Of course, this was a perfect fit for Oracle Application Express [Oracle APEX], so I built a web app in four hours, using the hosted Oracle APEX environment on apex.oracle.com, and started to manually enter the data of relatives who lived up to 350 years ago. The result is a nice graphical representation—you can test it out yourself here.

How are you using social media in your work these days? The social apps I use most are Twitter, Slack, and WhatsApp. The hashtag #orclapex on Twitter is especially great, as many Oracle APEX developers are actively using it in their tweets.

What's your go-to Oracle reference book? Jürgen Sieben's great book about Oracle APEX, *Oracle APEX: Das umfassende Handbuch für Entwickler* [Rheinwerk Computing, 2017]—but it's available only in German. In addition to books, I always go through all the slides from Oracle APEX conferences to keep myself up to date. It also helps that all content from the German Oracle user group [DOAG] can be downloaded freely.

Marco Mischke
Dresden, Germany

ORACLE ACE

Company/URL: Robotron
Job title: Group lead, database projects
Oracle credentials: Oracle Performance Tuning Expert (Oracle Database 11g), Oracle Real Application Clusters Certified Expert (Oracle Database 11g, Oracle Database 12c)
Length of time using Oracle products: 20 years

How did you get started in IT? As a teenager, I started machine-programming my old Commodore 64 and learned how computers work at a low level. That led to my study of IT at the Professional Academy in Dresden, and after finishing my studies, I got a job as a DBA at Infineon Technologies AG. My colleagues and I were responsible for roughly 100 production databases running 24/7. Up to that point, I thought databases were evil—I blamed them for performance issues that I was having during the software development stage and later when going live. But I quickly learned that it was my fault for not implementing things properly. I often look back on that experience when I talk to young developers—I can understand them and help them figure out what might be the best approach to solving their problems.

What technology has most changed your life? Mobile computing. It makes work much easier when you're not bound to an office, but it can be very addictive. You must really force yourself not to use your mobile device constantly for social connection. When I was a child growing up here in Dresden and there were no computers at all, "social connection" meant meeting up in the wild, and there was no stress if you weren't at home and didn't answer a phone call.

You've taken Oracle University [OU] classes in the past. What led you to do this? I needed a foundation upon which to build my knowledge of Oracle Database. The most useful OU class was Oracle Database Administration I + II. Without it, I wouldn't be where I am today.

Francesco Tisiot
Verona, Italy

ORACLE ACE Director

Company/URL: Rittman Mead
Job title: Business intelligence technology lead
Length of time using Oracle products: 11 years

What advice do you have about getting into web development? Be passionate about learning new things. This is a fast-changing world: The paradigms you know today will be different in six months or a year. Studying and experimenting with how various components can work together is what adds some spice to your day-to-day job. And while you learn, try to share what you're learning. You'll help other people in the same situation, and you'll start creating your public profile.

How are you using cloud computing in your work these days? Cloud is pervasive now in everybody's life. It provides services that used to take weeks or months in seconds or minutes. From a personal point of view, I use cloud as an enabler, allowing me to quickly study new features, integration options, and tools. From a professional/client point of view, it's both an accelerator and a focus-keeper. It gives the ability to create proofs of concept or full projects on timelines that were unimaginable before and, at the same time, removes almost all the pain related to "keeping systems alive," which was burning team resources and focus.

What's the next big thing driving change in your industry? I would say it's artificial intelligence embedded in analytics. AI and machine learning used to be niche features that only data scientists could handle properly. Analytical platforms such as Oracle Analytics Cloud are now embedding AI in the traditional workflow, democratizing data science, and thus enabling business analysts to gain data-scientist-level insights into the data with the same visual toolset they've been using on a daily basis.

Autobahn in the Cloud

How Gen 2 Oracle Cloud Infrastructure delivers for your business

BY ALAN ZEICHICK

Fast transactions mean happy customers. Fast transactions mean happy employees. And fast transactions mean more business agility and scalability. Oracle has worked hard to ensure that every part of its second-generation Oracle Cloud Infrastructure is geared toward performance. Part of that effort means using the latest technology within the servers, memory, storage, and network fabric.

Equally important is the networking used to connect resources within the Oracle Gen 2 cloud and to provide the fastest-possible

highways into and out of the cloud as well. That means high-speed links from on-premises data centers to applications running in Oracle Cloud—and recently to services running in Microsoft's Azure cloud.

Together, these advanced technologies provide rapid ingress and egress of data, drive applications and transactions, and help deliver success for enterprise customers and independent software vendors (ISVs) that use Oracle's business applications or deploy custom applications on the platform.

Fast Interconnects Pave the Superhighway

The high-performance servers and internal networks within the Gen 2 Oracle cloud ensure that cloud applications run quickly, delivering user experiences, analytics, AI, and other benefits directly to businesses.

Many Oracle Cloud users, however, run their applications across clouds from different vendors and in a hybrid environment, where part of the workload resides in Oracle Cloud Infrastructure and other parts are in the customer's own on-premises data center. In such scenarios, it's essential to provide a fast, reliable interconnection between Oracle Cloud Infrastructure and the customer's data center. *Reliable*, in this case, means more than ensuring that the link doesn't go down. It also means that the performance of the interconnect is predictable.

The interconnect is critical to performance, because some queries and transactions may require dozens or even hundreds of information transfers, some in parallel but others sequential, one after another. A slow interconnect, or one with unpredictable delays or throughput, might bog down one-off or routine tasks and cause a terrible user experience. In some cases, job processing might even time out and fail, requiring the process to start over.

"If I run in a hybrid cloud environment and I'm connecting the cloud to my data centers, what kind of latency can my application tolerate to be split across those things?" asks Charlie Baker, Oracle's senior director of product management for network services in Oracle Cloud Infrastructure. "Can I have my database in one place and my application in another?"

The answer, says Baker, is yes, and the reason is Oracle Cloud Infrastructure FastConnect,

> **You can choose how to run an application in the cloud and how to connect it with your on-premises apps—or not. We provide that flexibility and isolation between what you do in the cloud and what you do on premises.**
>
> —**Charlie Baker**, Senior Director, Oracle

which offers fast, predictable connectivity across on-premises and cloud networks. Using Cloud Infrastructure FastConnect is an alternative to using the unpredictable public internet, where data routing is based on how the internet is doing at that moment. Instead, Cloud Infrastructure FastConnect uses dedicated, pre-established paths to and from Oracle Cloud.

"It's like having your data center extended with a wire to the cloud," says Baker, adding that Cloud Infrastructure FastConnect was built specifically to take advantage of the network model, scalability, and compute capacity of Oracle Cloud Infrastructure. For example, customers can use Cloud Infrastructure FastConnect for disaster recovery and database backup and to bridge their data centers' applications and on-premises databases to Oracle Autonomous Data Warehouse, Oracle Autonomous Transaction Processing, and Oracle Autonomous Database Dedicated services running on Gen 2 Oracle Cloud Infrastructure—with better performance, improved reliability, and easier administration.

"Oracle Cloud Infrastructure is built with the requirements of the Oracle Autonomous Database services at the forefront," adds Baker.

Embrace Your Network Differences

Business and cloud networks are not all the same, and Cloud Infrastructure FastConnect enables administrators and network architects to choose how to deploy their virtual networks, including how to connect, route, and scale, says

Baker. "You can choose how to run an application in the cloud and how to connect it with your on-premises apps—or not," he says. "We provide that flexibility and isolation between what you do in the cloud and what you do on premises."

How does Oracle pull this off? It's a team effort: The Cloud Infrastructure FastConnect service is supported by dozens of top-shelf connectivity partners around the world.

What about customers that use multiple clouds, such as Oracle Cloud Infrastructure and Microsoft Azure? The two companies recently announced a direct connection using a fast interconnect. This connection is initially between Oracle's Ashburn Gen 2 cloud region and Microsoft's Azure US East cloud facility, both physically located in Ashburn, Virginia. The companies have plans to expand to additional regions.

Fast Gen 2 Servers Have Plenty of Horsepower

Cloud Infrastructure FastConnect is far from being Oracle Cloud Infrastructure's only performance driver—the entire architecture is built for performance.

Oracle Cloud Infrastructure completely separates the servers running customer workloads—code, data, and resources—from the servers Oracle uses to administer those servers. This approach has two key benefits:

1. **Security.** Customers deploy their workloads on compute and storage hosts on which no Oracle software of any kind runs. Oracle's administrative software can't see the customer's code—and customer code can't see Oracle's administrative software. This isolation is an important key to effective security.
2. **Reduction of overhead.** Essentially, 100% of the customer's cloud server's processor, memory, storage, storage bandwidth, and

> **"If I run in a hybrid cloud environment and I'm connecting the cloud to my data centers, what kind of latency can my application tolerate to be split across those things?"**
>
> —**Charlie Baker**, Senior Director, Oracle

network bandwidth is available for the customer's applications.

Workload isolation also helps to drive service reliability. Oracle's service-level agreement (SLA) specifies between 99% and 99.99% uptime for Oracle Cloud Infrastructure, depending on the services, so everyone can depend on Oracle Cloud Infrastructure for business-critical applications.

Alan Zeichick is director of strategic communications for Oracle, where he provides insights into and analysis on cloud computing and other advanced technologies. Follow him @zeichick.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

LEARN more about Oracle Cloud networking.

Oracle Cloud Infrastructure FastConnect.

READ about the Microsoft and Oracle cloud interoperability partnership.

Autonomous Database for All

A new dedicated service and the latest developer tools let more go autonomous.

BY JEFF ERICKSON

Even for people eager to use Oracle Autonomous Database's self-driving, self-repairing, and self-securing capabilities, the question always comes down to this: Is it right for my workload? So Oracle has expanded the types of autonomous database services it offers to cover more use cases and at the same time provide more developer tool choices for building apps using the cloud database.

A new deployment choice—Oracle Autonomous Database Dedicated—

means more high-volume, mission-critical workloads can take advantage of autonomous technology, because those workloads get their own isolated servers on Oracle Cloud. Plus, Oracle has added the most popular database developer tools to its autonomous services, including Oracle REST Data Services (REST Data Services), Oracle SQL Developer, and Oracle Application Express (Oracle APEX).

The autonomous database is part of Oracle Cloud Infrastructure's "compelling array of advanced technology features," says Oracle Executive Chairman and CTO Larry Ellison. The self-driving database "automatically encrypts all your data, backs itself up, tunes itself, upgrades itself, and patches itself when a security threat is detected," he says. "It does all of this autonomously—while running—without the need for any human intervention and without the need for any downtime. No other cloud infrastructure provides anything close to these autonomous features."

Oracle Autonomous Database is also a boon for developers, who get a new level of freedom to deploy a powerful Oracle database instance at the moment their projects call for it, says Maria Colgan, a master product manager at Oracle. Now "they've got the most performant and functionally rich database platform at their fingertips for a fraction of the time and cost," Colgan says.

What more do Oracle Autonomous Database Dedicated and the new developer tools mean for developers and mission-critical workloads in the cloud? *Oracle Magazine* caught up with a couple of experts to find out more.

Oracle Autonomous Database Dedicated "offers another deployment choice for autonomous databases" alongside the serverless options of Oracle Autonomous Transaction Processing and Oracle Autonomous Data Warehouse, says Sachin Sathaye, senior director, Oracle Cloud. Oracle Autonomous Database Dedicated offers a customizable private database instance running on dedicated Oracle Exadata (Exadata) infrastructure in Oracle Cloud. It's designed to address the well-known concerns of enterprise customers about workload isolation, performance, security, and control over operational policies when moving to the cloud, says Sathaye.

Oracle Autonomous Database Dedicated currently is hosted on a quarter rack of Exadata servers, with larger server allocations coming in

> **There's the comfort of the isolation, of being on your own dedicated infrastructure. You get a higher level of control over database provisioning, software updates, and availability.**
>
> —**Maria Colgan**, Master Product Manager, Oracle

future releases. "This is your completely dedicated hardware; it's just for you," says Sathaye. That means you keep the benefits of autonomous in terms of its self-driving, self-repairing, self-securing capabilities but gain new controls and visibility for your database, including being able to manage your dedicated autonomous database instances with Oracle Enterprise Manager.

"There's the comfort of the isolation, of being on your own dedicated infrastructure," says Colgan. "You get a higher level of control over database provisioning, software updates, and availability. Inside that Oracle Autonomous Database Dedicated Exadata rack, you can create separate clusters, separate container databases, and finally individual databases for separate groups."

The dedicated server means customers can also set the timing of software updates to better fit their business cycles. "Oracle is still going to patch and maintain that database for you, but you get to control when it happens," says Colgan.

Coding Options for Autonomous

"Developers spinning up a self-driving, self-tuning database for projects is one of the top use cases for autonomous databases, because they get all of the database setup, maintenance, and even scaling without bugging the IT team," says Manish Kapur, director, Oracle Cloud. Now, with a collection of popular developer tools available for all flavors of Oracle Autonomous Database, that trend is about to accelerate.

For example, Oracle APEX is a wildly popular low-code development tool in the Oracle Database community, and it comes with every on-premises Oracle Database and Oracle

Database service in Oracle Cloud. Oracle APEX comes with Oracle Autonomous Database and lets you quickly build and deploy scalable, secure, data-centric applications. It can be used to import spreadsheets and develop a single source of truth for your data in minutes, or it can be used to build your next mission-critical data management applications with compelling data visualizations. Since a developer can easily spin up an Oracle Autonomous Database instance without any other IT support, "in a few minutes you're up and running and building an app" using Oracle APEX, Kapur says.

Also available with Oracle Autonomous Database, REST Data Services supports developers building and deploying RESTful services. "REST Data Services gives you a way to build modern REST-enabled interfaces for relational data in Oracle Database," he says.

Another popular tool, Oracle SQL Developer Web is a web interface for working with Oracle Database services in Oracle Cloud. Now available with Oracle Autonomous Database, Oracle SQL Developer Web lets developers easily run queries, create tables, and generate schema diagrams.

"Regardless of the flavor of Oracle Autonomous Database you're using, these are all great options for both cloud native and low-code developers," Kapur says.

Jeff Erickson is editor at large for Oracle Content Central.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

LEARN more about Oracle Autonomous Database Dedicated.

TRY Oracle Autonomous Database.

READ more about Oracle APEX on Oracle Autonomous Database.

Oracle SQL Developer Web for Oracle Autonomous Database.

Oracle REST Data Services for Oracle Autonomous Database.

Chris White (left), Director of Customer Experience, and Chris Ashworth, CIO, at Hermes

VOICE IN THE MACHINE

Hermes creates a digital assistant for better customer experience and a more strategic workforce. **BY DAVID BAUM**

Last fall the fast-growing package delivery company Hermes launched a digital assistant named Holly to help customers get answers to their questions about their packages. Just seven months later, this chatbot is handling 38% of all chat volume and Hermes has reduced the support interactions that require a human by 30%. Holly helps customers track their shipments, change delivery orders, update account preferences, and handle many other tasks.

"A software bot can manage routine interactions quickly and effectively," says Chris White, director of customer experience at Hermes. "Holly automates a huge volume of transactions with our customers and with our network of couriers."

Founded in Germany in the 1970s, Hermes is one of Europe's leading parcel delivery services. The company delivers approximately 340 million parcels over the course of a year, both for consumers and as a partner of 90% of the UK's top retailers.

Hermes created Holly to help the company contend with a familiar problem: rapid growth. With year-over-year growth exceeding 20%, Hermes' customer service staff was working overtime to field a mounting volume of inquiries via three main contact channels: telephony, email, and chat.

"On any given day, we have roughly a million parcels running around in our network—and about twice that many during the busy holiday period," White says. "As our physical parcel volume continued to grow, we started to outgrow the elements of our services infrastructure."

New Girl on the Job

To create Holly, Hermes used Oracle Digital Assistant, a platform and set of tools for building AI-powered assistants. Developers use this cloud-based software environment to create natural conversational interfaces, through text or speech, to enterprise systems. It enables them to build and train digital assistants without the need for specialized AI skills. Developers don't have to worry about creating algorithms to understand natural-language processing or classify inputs.

"We got Holly up and running fairly quickly," White says. "Most of our time was spent establishing the flow or 'journey' that consumers follow during conversational interactions. We

Hermes' digital assistant, Holly, does more than help customers. "The agents feel less rushed, because Holly is dealing with a good chunk of the caseload for them," says Chris White, director of customer experience at Hermes.

wanted to make sure that these exchanges were as straightforward as possible."

White says the important thing with this type of project is to carefully model the workflows. "If you put nonsense in, then you're going to get nonsense back from the bot," he continues. "We put a lot of thought into creating a decision tree. The branches represent the ways a conversation can go, based on the supplied information and context. Within these conversations, there isn't much that a live agent would do differently from the bot."

Moving Toward an AI Future

Although artificial general intelligence remains a difficult nut to crack, chatbots programmed

> **We've taken away the mundane activity and freed up the human agents to handle the tricky cases—especially those instances that require experience and sensitivity.**
>
> — **Chris White**, Director of Customer Experience, Hermes

to respond to questions within a circumscribed domain have become extremely accurate and consistent.

"The more you increase your mastery of an ever-smaller domain, the easier it is to instantiate that knowledge in a computer program," explains Byron Reese, a futurist, entrepreneur, and author of *The Fourth Age: Smart Robots, Conscious Computers, and the Future of Humanity* (Atria Books, 2018). "If you want an AI to play chess or detect email spam, then you just teach it that simple thing. You don't ask it to generalize, contextualize, be creative, or anything else. That is what we call narrow AI."

Intelligent bots such as Holly use AI to improve the conversational experience—and boost efficiency—in the call center. "Using a bot that can operate 24 hours a day eliminates friction, by allowing us to accommodate a steady exchange of data," says Chris Ashworth, CIO at Hermes. "We welcome Holly for its efficient, speedy, and consistent performance, but it also injects some self-deprecating humor into what could be cold transactions."

But make no mistake: These transactions drive the business, and there is nothing funny about the information systems that give Hermes an edge. Just as Uber and Lyft transformed the taxi industry with an app that connects drivers with riders, Hermes created a logistics platform that connects parcel senders, a network of couriers, and parcel recipients, and its macro delivery infrastructure facilitates parcel delivery through its network of self-employed couriers.

“We are feeding 15,000 small business owners the information they need in order to efficiently complete our deliveries,” Ashworth says.

Hermes decided to use Oracle Digital Assistant partly because it is easy to integrate with other databases and cloud services. Finished bots can be exposed through standard chat and voice channels, a custom mobile app, or a link on a website.

A delivery business ships parcels and information. “We are feeding 15,000 small business owners the information they need in order to efficiently complete our deliveries,” says Chris Ashworth, CIO, at Hermes.

Hermes integrated its Holly chatbot with Oracle Service Cloud and is establishing an interface to Facebook Messenger. All of Hermes’ data is stored in Oracle Database, which will soon be migrated to Oracle Cloud.

“We have certain automated transactions that write back to Oracle Service Cloud,” White says. “The bot handles them directly.”

New Opportunities for Employees

There was an initial concern in Hermes’ customer service department that Holly would eliminate some jobs. In fact, the total number of jobs hasn’t changed, but customer support workers are engaged in tasks that are more interesting and challenging. “We’ve taken away the mundane activity and freed up the human agents to handle the tricky cases—especially those instances that require experience and sensitivity,” White notes. “The one thing we are not asking Holly to do right

now is deal with the elements that require an emotional response."

As Holly takes over more and more of the routine chat volume, Hermes' customer support team can devote more time to the company's email and voice channels, through which the more complex support cases tend to come. "The agents feel less rushed, because Holly is dealing with a good chunk of the caseload for them," White says.

The effects of Holly on Hermes employees collectively are broader still. Thanks in part to Holly, Ashworth observes, Hermes has already seen improved customer satisfaction (CSAT) scores on overall outcomes, and it expects to see reduced employee churn.

Holly has become quite capable in handling the simple tasks that have been entrusted to it. According to Ashworth, Holly occasionally receives thank you notes from appreciative customers. "I look forward to getting to the point where Holly can inform consumers if something is amiss, to mitigate problems before they even contact us," he says.

David Baum is a freelance writer focused on the intersection of business, technology, and culture.

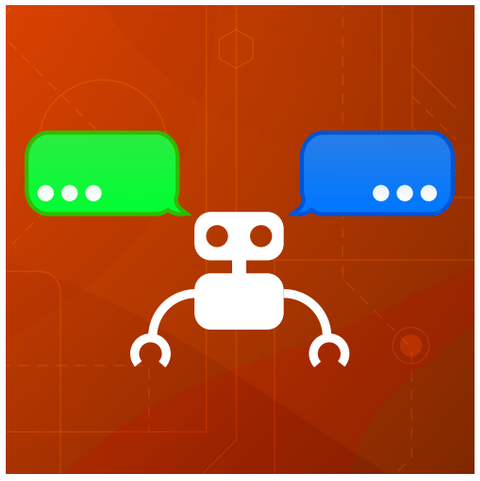
PHOTOGRAPHY BY
IAN GAVAN/GETTY IMAGES

NEXT STEPS

LEARN more about Oracle Digital Assistant.

TRY Oracle Digital Assistant.

BY FRANK NIMPHIUS



ORACLE DIGITAL ASSISTANT

Creating Chatbots That Are Less Robotic

Generate alternating bot responses with Oracle Digital Assistant.

"At the third stroke, the time from BT will be [hour] [minute] and [second] seconds." Almost forgotten, but still in use, is the speaking clock. A speaking clock is a service provided by telephone businesses and other organizations that announces the correct time in intervals of 10 seconds.

If, from time to time, it could say, "My goodness, time goes by really fast today. At the third stroke, the time from BT will be [hour] [minute] and [second] seconds," the clock would not only be more engaging but also feel more natural, because humans use different phrases when saying things repeatedly.

Unlike a speaking clock that is designed for a short consumer interaction and that does not need to be engaging, chatbots are supposed to have longer interactions with users while assisting them to complete a task on the conversational channel.

Chatbots are not humans and should not pretend they are, but conversations between bots and humans must be designed to be vivid, natural, and appealing to humans. Alternating bot responses is one technique to apply to your chatbots to improve user experience.

Following the hands-on instructions in this article, you will implement alternating bot responses for a skill in Oracle Digital Assistant. For this you will be using resource bundles referenced from the dialog flow and from items in a composite bag entity.

GETTING READY

Below are the prerequisites for following along with the hands-on steps in this article:

- You need a trial or a paid instance of Oracle Digital Assistant. You can sign up for a free trial at cloud.oracle.com.
- You need to download and extract the resources for this hands-on exercise to your computer.

Follow these initial hands-on steps to start the service and import and test the bot.

1. Start Oracle Digital Assistant in a browser by typing `https://<your cloud URL>/botsui/` into the **URL** field.
2. Click the hamburger icon (≡) at the upper left.
3. Choose **Development** and then **Skills**.
4. Close the menu by clicking the hamburger icon (≡).
5. Click the **Import Skill** button at the upper right.
6. Navigate to the downloaded and extracted resources for this article and navigate to the `starter` folder.

7. Select the `OraMagTravel(1.0).zip` starter bot file and click **Open**.
8. Click the **OraMagTravel** tile in the **Skills** dashboard to open the skill bot.
   Note: If you don't see the imported bot because other bots fill your screen, type `Ora` into the **Filter** field above the **+ New Skill** tile.
   What you just did: By importing the `OraMagTravel(1.0).zip` file, you installed the travel bot sample skill. The sample bot does not use natural language processing (NLP) or machine learning (ML), so there is no need to train the skill.

EXPERIENCE THE BOREDOM OF BOTS THAT ARE ROBOTIC

Before you implement alternating bot messages, first experience the bot from a user perspective.

9. Run the sample bot in the embedded conversation tester by clicking the **Skill Tester** icon (▶) in the left menu.
10. Type `Hi` into the **Message** field and press the Enter key.
11. Notice the prompt that is displayed for selecting the destination: "Where would you like to go to?"
12. Type `Miami` into the **Message** field and press the Enter key again.
13. Miami is not a valid answer. Type `Lisbon` into the **Message** field and press the Enter key.
14. Again, this is not a valid choice. However, notice that the prompt for the destination dialog box always displays, "Where would you like to go to?"
15. Select **Paris** from the list of destinations.
16. When prompted for the departure date, type `August 20 2019`.
17. When prompted to provide a return date, type `never`.
18. Of course, "never" is not a valid date. So the return date prompt appears again. And, as before, it displays, "Please provide a return date."

19. Click the **Reset** button (Reset) at the top right.
20. Close the skill tester by clicking the **Close** icon (X), located next to the **Reset** button.

What happened: The travel bot validated your input against a custom entity for destinations and the `DATE` system entity. When you provided an invalid user input, the dialog box asked for a valid entry using the same message displayed before. In a human conversation, this would not be the case. Also in a human conversation, the response of the person you talk to would not be the same each time you say, "Hi."

Note: A previous Oracle Magazine article explained how to handle input validation errors gracefully. The focus of this article is on alternating prompts.

BEFORE YOU START

Using references to resource bundles is common practice in software development for displaying labels, prompts, and similar messages. Resource bundles not only make it easy for you to maintain strings but they also allow you to build translatable user interfaces.

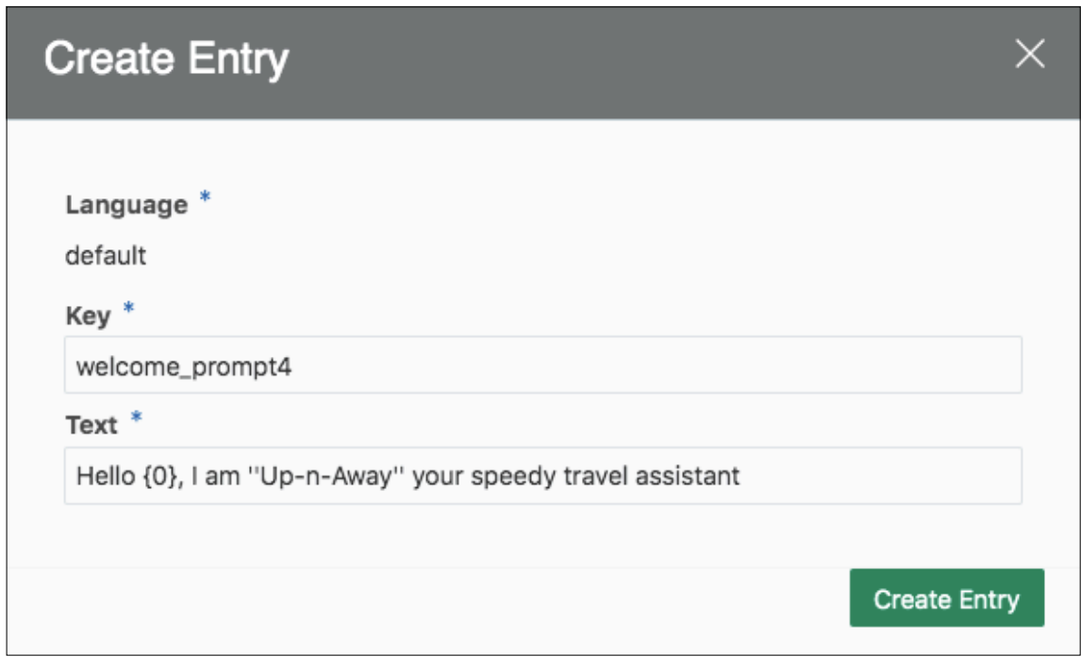
The starter skill that you imported includes a resource bundle created for English. Before you can start building alternating bot responses, take a look at the strings and their keys.

21. Click the **Resource Bundles** icon ( ) in the menu on the left.
22. In the upper right corner, select **Language** from the **View By** list. This lists all keys and their string values in a tabular layout.
23. Note that several keys have the same name and differ only by a consecutive number.
24. Also note that the message for each of those keys is related but different.
25. Create a new key by clicking the **+ Key** button (+ Key).

26. In the **Key** field, type `welcome_prompt4`.

27. In the **Text** field, type `Hello {0}, I am ''Up-n-Away'' your speedy travel assistant` (as shown in **Figure 1**). The "{0}" reference is a parameter that allows bot designers to inject data into the resource bundle message.
Note: The string "Up-n-Away" is enclosed by two single quotation marks at its beginning and its end. This will be displayed as 'Up-n-Away' at runtime.

Figure 1: Creating a new resource string key and message



28. Click the **Create Entry** button.

29. Notice that not all `welcome_prompt` keys use parameters in their messages.

30. Close the **Resource Bundles** panel by selecting the **Flows** icon ( ) in the left menu.

What you've seen: In this section, you've seen part of the solution for implementing alternating bot responses in Oracle Digital Assistant. For each message, label, or prompt to be displayed in a conversation, you create multiple resource bundle entries. The resource bundle keys defined for a dialog flow state differ by a consecutive number. The other part of the solution is the generation of a random number that determines the prompt, label, or message to display at runtime.

IMPLEMENTING YOUR FIRST ALTERNATING BOT MESSAGE

In this section, you are going to implement an alternating bot message for the welcome message. Ensure that you have the dialog flow builder open by selecting the Flows icon (), as instructed in step 30.

31. Place the cursor into line 14.

32. Ensure that the cursor is vertically aligned with the `returnDate` variable (in line 13). If needed, use blank spaces to align the cursor.

33. Create a resource bundle variable `rb` of type `resourcebundle`. To do this, type `rb: "resourcebundle"`.

34. Go to line 20 and change the value of the `text` property from

```
"Hi ${profile.firstName}, I am 'Up-n-Away', your travel
booking assistant. Let's get your booking rolling."
```

to

```
${rb('welcome_prompt${(.now?long %
4)+1}','${profile.firstName}')}
```

Note: You can copy the expression from the `welcome_prompt.txt` file located in the `strings` folder of the extracted resources for this article.

35. Click the **Validate** button in the upper right to ensure that you provided the correct syntax.
36. Run the sample bot in the embedded conversation tester by clicking the **Skill Tester** icon (▶) in the left menu.
37. Type `Hi` into the **Message** field and press the Enter key.
38. Notice the welcome message.
39. Click the **Reset** button (Reset) in the upper right.
40. Again, type `Hi` into the **Message** field and press the Enter key. This should display a different welcome message.
41. Close the skill tester by clicking the **Close** icon (X), located next to the **Reset** button.

Note: This example uses four possible messages to greet users. There is a possibility that two consecutive runs will display the same message, but the likelihood that alternative messages will be displayed increases with the number of message keys added for a prompt or bot response.

How it works: For the welcome message, there are four keys defined in the message bundle. An Apache FreeMarker expression is used to generate a number between 1 and 4. There is no dynamic way to determine the number of keys defined for a message, so the number of available welcome messages needs to be provided using `% 4`. To create a random number, the expression uses the current date in milliseconds, which it accesses through the `.now` reference. The generated number is then appended to the `welcome_prompt` string to build a valid key contained in the message bundle.

Note: Some of the `welcome_prompt` keys use a parameter to add the user name to the message. For this you passed `${profile.firstname}` as a second argument in

the expression. If a message does not support parameters, this second argument is ignored.

ADDING ALTERNATING MESSAGES TO ALL BOT RESPONSES

In this section, you are going to replace all other string messages with resource bundle references. For simplicity, the starter skill has three resource bundle keys defined for each message.

42. Go to line 28 and change the value of the `prompt` property from

```
"Where would you like to go to?"
```

to

```
${rb('destination_prompt${(.now?long % 3)+1}')}
```

Note: You can copy the expression from the `destination_prompt.txt` file located in the `strings` folder of the extracted resources for this article.

43. Go to line 45 and change the value of the `prompt` property from

```
"Please provide a start date"
```

to

```
"${rb('departureDate_prompt${(.now?long % 3)+1}')}"
```

Note: You can copy the expression from the `departureDate_prompt.txt` file located in the `strings` folder of the extracted resources for this article.

44. Go to line 61 and change the value of the `text` property to the string below. You can copy the expression from the `confirmation_prompt.txt` file located in the `strings` folder of the extracted resources for this article.

```
${rb('confirmation_prompt${(.now?long % 3)+1}',
'${destination.value}','${startDate.value.date?
long?number_to_date?string("dd-MMM-yyyy")}',
'${returnDate.value.date?long?number_to_date?
string("dd-MMM-yyyy")}')}
```

Note that this time, the `text` property value ***must not be enclosed*** within double quotation marks (see **Figure 2**).

Figure 2: Prompt expression not enclosed within double quotation marks.

```
printConfirmation:
  component: "System.Output"
  properties:
    text: ${rb('confirmation_prompt${(.now?long % 3)+1}','${destination.value}',
          '${startDate.value.date?long?number_to_date?string("dd-MMM-yyyy")}',
          '${returnDate.value.date?long?number_to_date?string("dd-MMM-yyyy")}')}
  transitions:
    return: "done"
```

Note: Figure 2 shows the expression formatted in multiple lines for better viewing. When you copy and paste the expression from `confirmation_prompt.txt`, it will be in a single line, which is perfectly OK.

45. To test your configuration, click the **Skill Tester** icon (▶) in the left menu.
46. Type `Hi` into the **Message** field and press the Enter key.
47. Select **Paris**.
48. When prompted for a departure date, type `August 20 2019`.
49. When prompted for a return date, type `August 31 2019`.
50. You should see a confirmation message.
    Click the **Reset** button (Reset) in the upper right and repeat steps 46 to 49. You should see different prompts and a different confirmation message.
    Note: As mentioned earlier, this example uses three resource bundle entries for each dialog flow state to display prompts and messages. The likelihood that alternative messages will be displayed increases with the number of message options added for a prompt or bot response.
51. Close the tester by clicking the **Close** icon (X), located next to the **Reset** button.

Note: In step 44, the Apache FreeMarker expression is added as a value to the `text` property without surrounding quotes. The reason for this is that double quotation marks are used in the formatting of the dates: *`?string("dd-MMM-yyyy")`*. Adding surrounding double quotation marks to the expression as a whole would have led to an invalid BotML document.

ON YOUR OWN

Did you notice there is no alternating message defined yet for the prompt of the `setEndDate` dialog flow state in line 53? Well, how about a challenge? Try building the expression yourself considering the following information:

- The resource bundle key base name is `returnDate_prompt`.
- There are three key entries defined in the resource bundle.
- None of the bundle messages contains a parameter.

- The solution is similar to the departure date prompt you added in line 45.

Note: To verify your solution, or to skip this challenge on a lazy day, you can find the correct expression in the `returnDate_prompt.txt` file located in the `strings` folder of the extracted resources for this article.

With this addition, you have completed the first part of this hands-on exercise. All skill bot prompts and text responses should now use alternating messages and appear less robotic. You can find a completed solution—`OraMagTravel_Completed(1.0).zip`—in the `completed` folder of the extracted downloads for this article.

ABOUT COMPOSITE BAG ENTITIES

Composite bag entities are like business domain objects and encapsulate multiple entities and variables into a single object. A composite bag entity, for example, could represent an order, an account, a product, and more.

Besides being business oriented, the benefit of composite bag entities is that they resolve themselves at runtime by generating the dialog boxes needed for a user to provide required entity information.

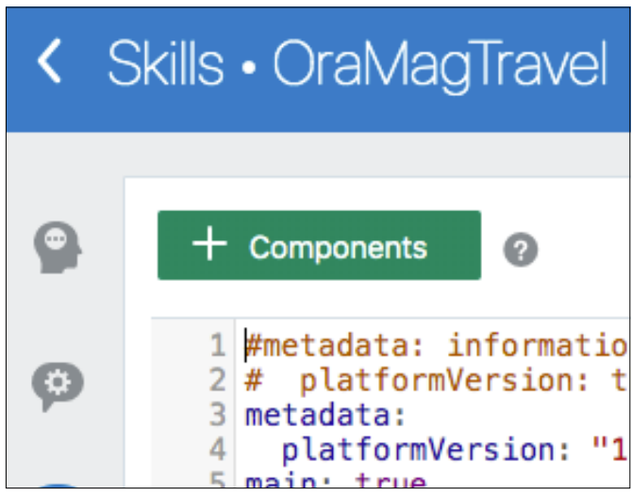
Because composite bag entities generate the bot conversation flow at runtime, you don't build dialog flow states, which has an impact on how you define prompts. In the next sections, you will learn how to define prompts for composite bag entities and how to define them so they display alternating responses.

GETTING READY

To start working with composite bag entities, first import the composite bag skill.

52. If you are in the **OraMagTravel** bot, click **<** (left arrow) in the header (see **Figure 3**) to exit skill editing.

Figure 3: Left arrow icon in the skill builder header to exit skill editing



53. Back in the **Skills** dashboard, click the **Import Skill** button at the upper right.

54. Navigate to the downloaded and extracted resources for this article and navigate to the `starter` folder.

55. Select the `OraMagTravelCompositeBag(1.0).zip` starter bot file, and click **Open**.

56. Click the **OraMagTravelCompositeBag** tile in the **Skills** dashboard to open the skill bot.

Note: If you don't see the imported bot because other bots fill your screen, type `Ora` into the **Filter** field above the **+ New Skill** tile.

USING ALTERNATING MESSAGES WITH COMPOSITE BAG ENTITIES

As mentioned earlier, prompts ideally reference resource bundles, so they are easy to maintain and support translations. In this section, you are defining prompts for composite bag entity items and adding the composite bag entity to the dialog flow using the `System.ResolveEntities` component.

57. Click the **Entities** icon ( ) in the left menu.
58. Select the **TravelBooking** entity, which is a composite bag entity.
59. Under **Configuration** -> **Bag Items**, select the **StartDate** item.
60. Click the **Edit** icon ( ) on the right to edit the bag item.
61. In the **Edit Bag Item** page, scroll to the **Prompts** section.
62. Click the **+ Prompt** button (+ Prompt).
63. Type `${rb.departureDate_prompt1}` into the **Prompt** field.
64. Press the Tab key twice to navigate out of the edit fields. Navigating out of the edit fields ensures the changes will be saved.
65. Repeat steps 62 to 64 to create prompts for `${rb.departureDate_prompt2}` and `${rb.departureDate_prompt3}`.
    The **Prompts** section should now look like the image in **Figure 4**.
66. When all three prompts have been created, click the **Close** button at the top right.
    Note: The sequence numbers that are automatically added to the prompts you create define the order in which the prompts are displayed. You could

Figure 4: Definition of three departure date prompts

remove the sequence numbers if you want to display the prompts in a purely random order.

67. Repeat steps 59 to 66 for the following bag items and the six prompts listed below:

Bag Item	Prompts
TravelDestination	
	`${rb.destination_prompt1}`
	`${rb.destination_prompt2}`
	`${rb.destination_prompt3}`
EndDate	
	`${rb.returnDate_prompt1}`
	`${rb.returnDate_prompt2}`
	`${rb.returnDate_prompt3}`

68. Select the **Flows** icon () in the left menu to switch to the dialog flow builder.
69. Notice that the variable `rb` of type `resourcebundle` has been defined already for you (in line 11). This variable is mandatory for the skill to use resource bundle references, not only in dialog flows but also in composite bag entities.
70. Also notice the `travelBooking` variable (in line 10), which, as a type, is set to the name of the composite bag entity (`TravelBooking`).
71. To render the composite bag entity in the context of a bot/user interaction, you need to add a `System.ResolveEntities` component state to the dialog flow. To do this, first click the **+ Components** button at the top of the editor.
72. In the opened dialog box, select the **User Interface** category.

73. Scroll through the list of component templates to find the **Resolve entities** entry.
74. Select **Resolve entities**.
75. Select **Welcome** from the **Insert After** list. This will add the new component state after the existing welcome state in the dialog flow.
76. Enable the **Remove Comments** toggle. This removes comments from the generated component state.
77. Click the **Apply** button.
78. In the dialog flow builder, change the `resolveEntities` state name to `selectDestination`.
79. Set the `variable` property value to `"travelBooking"`.

```
variable: "travelBooking"
```

80. Delete the following properties:

```
nlpResultVariable:
maxPrompts:
cancelPolicy:
transitionAfterMatch:
autoNumberPostbackActions:
headerText:
footerText:
showMoreLabel:
translate:
```

81. Edit the transitions section so the component state looks as shown below:

```
selectDestination:
  component: "System.ResolveEntities"
  properties:
    variable: "travelBooking"
    transitions:
      next: "printConfirmation"
```

Note that the size of each indentation is two blank spaces.

What you just did: In this section, you created prompts for the three bag items defined in the **TravelBooking** composite bag entity. For each item, you defined three prompts that reference resource bundle strings. The resource bundle strings are the same as were used in the first part of this hands-on exercise. To resolve the composite bag entity, you used a component template for the `System.ResolveEntities` component. To resolve the component to the composite bag entity, you referenced the `travelBooking` variable, which is of type `TravelBooking` from the component's `variable` property.

Now test the user experience with the composite bag entity.

82. Open the conversation tester by clicking the **Skill Tester** icon (▶) in the left menu.

83. Type `Hi` into the **Message** field and press the Enter key on your keyboard.

84. Notice the prompt requesting the departure date information, which is read from the resource bundle.

85. To see a different prompt, type `Hello` into the **Message** field and press the Enter key.

86. Next, type `August 20 2019` into the **Message** field and press the Enter key.

87. Again, to see different messages, type `Lisbon` when prompted for a destination. Because Lisbon is an invalid value, a prompt appears, this time with a different message.
88. When prompted again to provide a destination, select **Paris**.
89. When prompted for a return date, type `August 30 2019` into the **Message** field and press the Enter key.
90. Notice the confirmation message.
91. Close the skill tester by clicking the **Close** icon (X), located next to the **Reset** button.
92. In the dialog flow editor, review the `printConfirmation` state and notice the variable reference to the composite bag entity being used to pass parameter values to the resource bundle.
    Note: By default, the composite bag items use sequences to determine when to show which message. Because of the sequences, the prompts are displayed the same when rerunning the whole conversation. To see alternating prompts, you either need to cause a field to be rerendered, as you did by providing invalid values for a prompt, or you need to remove the sequence numbers.

Both approaches, showing alternating messages in a sequence and showing them in random order, have their use cases. Sequences are good for "escalating" user input failure, because the initial prompt in the sequence can be short while subsequent prompts can provide more information about what the user is expected to provide.

Note: You can find a completed solution—`OraMagTravelCompositeBag_Completed(1.0).zip`—in the `completed` folder of the extracted downloads for this article.

CONCLUSION

In this article, you learned about the two approaches in Oracle Digital Assistant to implementing alternating bot response messages for chatbots to make them appear less robotic. In real bot implementations, you will use both approaches—random messages defined in the dialog flow and messages defined on composite bag entity items—because there are no restrictions on how and when to use composite bag entities within a bot conversation.

Frank Nimphius is a master principal product manager in the Oracle Digital Assistant Product Management team.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

READ more about the speaking clock.

TRY Oracle Digital Assistant.

DOWNLOAD the bot for this article.

ORACLE DATABASE

Files Up and Files Down

Use node-oracledb to support uploading and downloading files from your REST APIs.

Storing images, documents, and other files in Oracle Database is nothing new. But it wasn't until support for binding and fetching buffers was added to node-oracledb that a simple solution for this emerged in the Node.js world. In this article, I'll show you how to use Node.js to upload files from a client into the database—and then download them back out.

WHY STORE FILES IN THE DATABASE?

You might be wondering why you should use the database to store files (unstructured data) when you could just use the file system? This is a great question. Here are a few important reasons:

To better relate unstructured data to structured data. For example, you can associate images of receipts with detail lines in an expense reporting application.

To simplify backup/recovery. It often makes more sense to reuse your existing database backup/recovery strategies than to invent new ones for the file system.

To leverage advanced database security features. From end-to-end encryption to advanced access controls, Oracle Database has a lot to offer when it comes to security.

To index files and make them searchable. Many document types can be made searchable with "Google-like" search results via Oracle Text.

As you can see, there are some good use cases for storing files in the database, so let's get started!

SETUP AND API REQUIREMENTS

In this article, I assume you're running everything from within the Database App Development VM with Node.js and Git installed. (See this post for instructions on setting up this type of environment.) You may use any other setup you wish, but you'll need to adapt the instructions that follow accordingly.

I'll pick up the code where I left off in the series on building a REST API with Node.js. To get access to the code for that series on GitHub, run the following lines of code from a terminal:

```
cd ~
git clone
cd oracle-db-examples/javascript/rest-api/part-5-manual-pagination-sorting-
and-filtering/hr_app
```

This will clone the oracle-db-examples repo in GitHub and then change directories to the last article of that REST API series. This will be your starting point for building out the API for uploading and downloading files with Node.js.

Run the following lines of code in a tool such as Oracle SQL Developer to create the table in the HR schema that will be used to store uploaded files:

```
create table jsao_files (
  id           number generated always as identity not null,
  file_name    varchar2(255) not null,
  content_type varchar2(255) not null,
  blob_data    blob,
  constraint jsao_files_pk primary key (id)
);
```

With the table created, you're now ready to start building out the API. To demonstrate a few capabilities beyond simply uploading and downloading files, I'll define a couple business rules in the API. For uploads, the maximum size of the file should be 50 MB. For downloads, browsers should be able to download the file with the name saved when the file was uploaded. Consider adding more business rules once you have the API working correctly.

ROUTING LOGIC

The first change you will make to the Node.js API is to add another route and related handlers to the router. Open the `services/router.js` file, and add the following line below the existing variable declarations (line 4):

```
const files = require('../controllers/files.js');
```

Currently the `files.js` controller file doesn't yet exist. You'll create that file in a

subsequent step, but keep in mind that your application will not start until that file is created.

Next add the following lines below the existing call to `router.route`.

```
router.route('/files/:id?')
  .get(files.get)
  .post(files.post);
```

The new route being defined is `/files`, with an optional `id` parameter in the path. HTTP `get` and `post` requests will be routed to the respective `get` and `post` functions from the `files` controller module.

CONTROLLER LOGIC FOR UPLOADS

Create a new file in the `controllers` directory named `files.js`. Open the file, and add the following lines of code (and don't forget to save your changes afterward):

```
const files = require('../db_apis/files.js');

async function post(req, res, next) {
  try {
    const maxFileSize = 1024 * 1024 * 50; // 50MB; OCI limit is 1 GB unless streaming
    let contentBuffer = [];
    let totalBytesInBuffer = 0;
    let contentType = req.headers['content-type'] || 'application/octet';
    let fileName = req.headers['x-file-name'];
```

```
if (fileName === '') {
  res.status(400).json({error: `The file name must be
                          passed to the via x-file-name header`});
  return;
}

req.on('data', chunk => {
  contentBuffer.push(chunk);
  totalBytesInBuffer += chunk.length;

  if (totalBytesInBuffer > maxFileSize) {
    req.pause();
    res.header('Connection', 'close');
    res.status(413).json({error: `The file size exceeded the
                          limit of ${maxFileSize} bytes`});

    req.connection.destroy();
  }
});

req.on('end', async function() {
  contentBuffer = Buffer.concat(contentBuffer, totalBytesInBuffer);

  try {
    const fileId = await files.create(fileName, contentType, contentBuffer);
```

```
            res.status(201).json({fileId: fileId});
          } catch (err) {
            console.log(err);

            res.header('Connection', 'close');
            res.status(500).json({error: 'Oops, something broke!'});

            req.connection.destroy();
          }
        });
      } catch (err) {
        next(err);
      }
    }

    module.exports.post = post;
```

The controller logic above starts by declaring and initializing some variables. Note that `contentBuffer` is an array in which data will be buffered during a file upload. Also, the code expects the file's name and content type to be passed in via HTTP headers, because the HTTP body is reserved for the file content.

An event listener is added to the request's `data` event. This event will be triggered one or more times as data is transferred from the client to the server. The callback function takes the data passed in (a `Buffer` by default) and pushes it into the `contentBuffer` array. The rest of the logic in the function is used to track the number of bytes in the buffer. If the maximum file size (`maxFileSize`) of 50 MB is exceeded,

an error message will be sent to the client and the connection will be destroyed.

Next an event listener is added to the `end` event of the request. This event will fire after all the `data` events, when there is no more data to read. The callback function takes the `contentBuffer` array and converts it into a scalar `Buffer` object. The buffer, filename, and content type are then passed to the `files` database API to insert the file into the table. If successful, a "201 Created" status code and file ID are sent back to the client in the response.

With the controller logic in place, you can now add the related database logic.

DATABASE LOGIC FOR UPLOADS

Create a new file named `files.js` in the `db_apis` directory. Open the file, and add the following lines of code:

```
const oracledb = require('oracledb');
const database = require('../services/database.js');

const createSql =
 `insert into jsao_files (
    file_name,
    content_type,
    blob_data
  ) values (
    :file_name,
    :content_type,
    :content_buffer
  ) returning id into :id`;
```

```
async function create(fileName, contentType, contentBuffer) {
  const binds = {
    file_name: fileName,
    content_type: contentType,
    content_buffer: contentBuffer,
    id: {
      type: oracledb.NUMBER,
      dir: oracledb.BIND_OUT
    }
  };

  result = await database.simpleExecute(createSql, binds);

  return result.outBinds.id[0];
}

module.exports.create = create;
```

This module starts by bringing in node-oracledb (to access some constants) and the custom database module (for the `simpleExecute` method). The SQL `insert` statement assigned to `createSql` is very simple—and that's the beauty of buffering!

In the `create` function, all you need to do is create a `binds` object and execute the SQL. Note that the `content_buffer` bind variable is just a simple "in bind" that takes the entire file contents as a buffer. The `id` bind variable is defined as an "out bind," and its value is returned to the controller logic after the insert is complete.

CONTROLLER LOGIC FOR DOWNLOADS

Now that the logic for uploading files is in place, it's time to add the logic for downloading files. Return to the `controllers/files.js` file, and append the following lines of code to the bottom of the file:

```
async function get(req, res, next) {
  try {
    const id = parseInt(req.params.id, 10);

    if (isNaN(id)) {
      res.status(400).json({error: `The id of the file must be provided`});
      return;
}

const rows = await files.get(id);

if (rows.length === 1) {
  res.status(200);

  res.set({
    'Cache-Control': 'no-cache',
    'Content-Type': rows[0].content_type,
    'Content-Length': rows[0].file_length,
    'Content-Disposition': 'attachment; filename=' + rows[0].file_name
  });
```

```
      res.send(rows[0].blob_data);
    } else {
      res.status(404).end();
    }
  } catch (err) {
    next(err);
  }
}
module.exports.get = get;
```

The get function starts by parsing the ID of the file to be retrieved. If it's not parsed correctly, an error message will be returned to the client; otherwise the ID is passed to the get function of the `files` database API.

Next the length of the array returned from the get call is checked. If the array has exactly one row, a "200 OK" status message will be sent to the client. Various headers are added to the response to satisfy the requirement of working with browsers. Finally, the actual BLOB fetched from the database is sent to the client.

DATABASE LOGIC FOR DOWNLOADS

The last part of the code needed for the API to work is the get function in the `files` database API. Open the `db_apis/files.js` file again, and add the following lines of code to the bottom of the file:

```
const getSql =
 `select file_name "file_name",
    dbms_lob.getlength(blob_data) "file_length",
```

```
      content_type "content_type",
      blob_data "blob_data"
    from jsao_files
    where id = :id`

async function get(id) {
  const binds = {
    id: id
  };
  const opts = {
    fetchInfo: {
      blob_data: {
        type: oracledb.BUFFER
      }
    }
  };
  const result = await database.simpleExecute(getSql, binds, opts);
  return result.rows;
}
module.exports.get = get;
```

As was the case with the `insert` statement, the SQL query assigned to `getSql` is very simple. All that was necessary to get it to work in the `get` function was to define the `binds` and `opts` objects. The only bind variable was an "in bind" for the ID. As to the options, I'm using the `fetchInfo` option to change the return type for the `blob_data` column to be a buffer (a stream object is returned for BLOBs by default).

TESTING THE API

With all the work completed, it's time to try the new additions to the API! Run the following commands in a terminal:

```
cd ~/oracle-db-examples/javascript/rest-api/part-5-manual-pagination-sorting-and-
filtering/hr_app
npm install
node .
```

That should start the Node.js application.

If you get an "ORA-01017: invalid username/password;" error message, it indicates that the environment variables referenced in the `config/database.js` file have not been set or are incorrectly set. Run the following lines of code in a terminal, and then try to start the app again:

```
echo "export HR_USER=hr" >> ~/.bashrc
echo "export HR_PASSWORD=oracle" >> ~/.bashrc
echo "export HR_CONNECTIONSTRING=0.0.0.0/orcl" >> ~/.bashrc
source ~/.bashrc
```

To test an upload, open a new terminal window (while the API is running in the previous terminal) and run the following lines of code:

```
cd ~
curl -X "POST" "http://localhost:3000/api/files" \
```

```
-H 'x-file-name: runTimeClickHere.png' \
-H 'Content-Type: image/png' \
--data-binary '@runTimeClickHere.png'
```

The `curl` command issues a `POST` request to the new API endpoint and sends an image file that was in the Oracle user's home directory as the body of the request. This simulates the type of HTTP request a browser would issue when using `XMLHttpRequest.send()` with a file reference. You should receive a response such as `{"fileId":1}`, which indicates that the file was successfully inserted and its ID value is 1.

Next you can test-download the file by putting the previously returned ID value at the end of the URL path. Run the following command in a terminal:

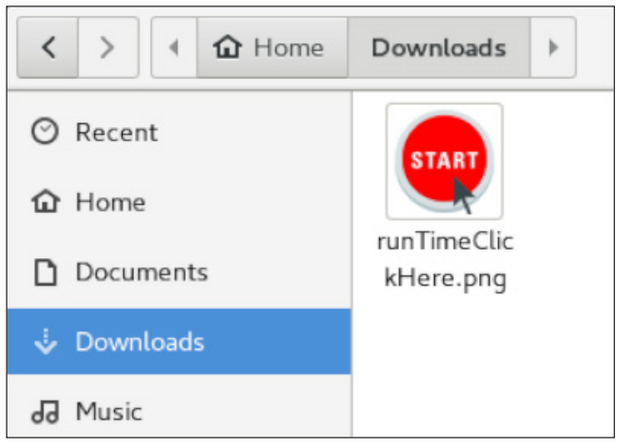
```
curl "http://localhost:3000/api/files/1" --output my-file.png
```

That `curl` command will take the contents of the HTTP response body and write them to a file named `my-file.png`. The new file is a binary copy of the original file.

Although `curl` doesn't pay attention to the response headers, by default, you can confirm that they are working by opening the Firefox web browser and navigating to `http://localhost:3000/api/files/1`. When prompted, select **Save File** and then click **OK**. Finally, use the file explorer to navigate to your downloads directory, where you'll find a file that has the same name as the original (as shown in **Figure 1**). If you can see the image, then both uploads and downloads are working correctly.

Hopefully you'll agree that the node-oracledb driver's ability to bind and fetch buffers can make quick work of adding file upload and download capabilities to your

Figure 1: Checking the downloads directory



Node.js APIs. If you'd like to learn more about uploading and downloading files with Node.js and Oracle Database, including how to work with streams and use a sample client app, visit this series at the JavaScript and Oracle blog.

Dan McGhan is the Oracle developer advocate for JavaScript and Oracle Database. He enjoys sharing what he's learned about these technologies and helping others be successful with them.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

LEARN more about JavaScript and Oracle.

TRY Oracle Cloud.

ORACLE AUTONOMOUS TRANSACTION PROCESSING

Developers: Create Autonomous Databases

Create an Oracle Autonomous Transaction Processing instance with the Python SDK for Oracle Cloud Infrastructure.

You've been working with Oracle Autonomous Transaction Processing, using the web dashboard, and now you're ready to automate the process. This article demonstrates how to create a new Oracle Autonomous Transaction Processing instance in your Python application, using the Python SDK for Oracle Cloud Infrastructure. (You can refer to the documentation for more information about the SDK.)

PREREQUISITES

Before you create your new instance, confirm that

- You are able to create an Oracle Autonomous Transaction Processing instance, using the Oracle Cloud Autonomous Database web dashboard.

- You have successfully installed and configured the command-line interface for Oracle Cloud Infrastructure.
- You have installed Python and have a basic understanding of Python programming. Python 2.7+ and 3.5+ are supported.

I will be using Python 3 for the article examples, and I will be running them on a Linux system. You may need to alter some system commands if you're using a different operating system.

INSTALLATION

To start the SDK installation, first create and change into a directory for your project.

```
$ mkdir ~/ociPythonSdkExample
$ cd ~/ociPythonSdkExample
```

Python setup. To isolate your Python environments, it's a good idea to use the virtualenv Python virtual environment. Follow the steps for your operating system if you're new to virtualenv.

On my Linux system, I use the following commands to install virtualenv (`virtualenv`) and create a new environment (`env`).

```
$ python3 -m pip install --user virtualenv
$ python3 -m venv env
```

This will create a new directory called `env`.

Now I activate the new environment and check that it is active.

```
$ source env/bin/activate
$ which python
```

For your vitualenv setup, you should see something similar to this result when you activate the environment and check it:

```
/home/bcarter/ociPythonSdkExample/env/bin/python
(env) [bcarter ociPythonSdkExample]$
```

Now that you're safely working in the virtualenv Python virtual environment, install the Python SDK for Oracle Cloud Infrastructure.

```
$ pip install oci
```

Verify that it is installed and configured correctly.

```
$ python3
Type "help", "copyright", "credits" or "license" for more information.
>>> import oci
>>> # Load the configuration file
... config = oci.config.from_file("~/.oci/config", "DEFAULT")
>>> # Create a service client
... identity = oci.identity.IdentityClient(config)
>>> # Get the current user
... user = identity.get_user(config["user"]).data
>>> print(user)
```

```
{
 "capabilities": {
   "can_use_api_keys": true,
   "can_use_auth_tokens": true,
   "can_use_console_password": true,
   "can_use_customer_secret_keys": true,
   "can_use_smtp_credentials": true
 },
 "compartment_id": "ocid1.tenancy.oc1..aBigLongGuidString",
 "defined_tags": {},
 "description": "me",
 "email": "blaine.carter@oracle.com",
 "external_identifier": null,
 "freeform_tags": {},
 "id": "ocid1.user.oc1..aBigLongGuidString ",
 "identity_provider_id": null,
 "inactive_status": null,
 "is_mfa_activated": false,
 "lifecycle_state": "ACTIVE",
 "name": "BlaineCarter",
 "time_created": "2019-03-21T18:58:06.069000+00:00"
}
>>> quit()
```

If you receive an error message when you're verifying the installation and configuration of Python SDK for Oracle Cloud Infrastructure, make sure the command-line

interface for Oracle Cloud Infrastructure is also installed and configured correctly.

Oracle Cloud Infrastructure compartment. It's a good idea to work inside an Oracle Cloud Infrastructure compartment so you don't accidentally affect the cloud objects of other users.

If you'd like to use a specific compartment, you'll first need to get its Oracle Cloud ID (OCID).

To get the OCID in your Oracle Cloud Infrastructure dashboard,

1. Expand the menu.
2. Click **Identity/Compartments**.
3. Find your compartment in the list.
4. Click the OCID value.
5. Click **Copy** in the pop-up.

Now create an environment variable with this OCID value:

```
$ export OCI_COMPARTMENT='ocid1.compartment.oc1..aBigLongGuidString'
```

The example code will look for this value or default to the root compartment if you don't set it.

PYTHON CODE

The `oci_atp.py` utility module will include the methods that will create a new Oracle Autonomous Transaction Processing instance. The module will include three methods, and each method will accept an oci.database.DatabaseClient object.

create_atp(db_client, atp_details). The `create_atp` method in `oci_atp.py` also accepts an oci.database.models.CreateAutonomousDatabaseDetails object. The `atp_details` object defines the properties of the new Oracle Autonomous

Transaction Processing instance. The properties I will be passing into the `create_atp` method are

- `atp_details.admin_password`
- `atp_details.compartment_id`
- `atp_details.db_name`
- `atp_details.display_name`
- `atp_details.cpu_core_count`
- `atp_details.data_storage_size_in_tbs`
- `atp_details.license_model`

The `create_atp` method calls create_autonomous_database, passing in the details object. It stores the response in the `atp_response` variable. The response object contains detailed information about the newly provisioned Oracle Autonomous Transaction Processing instance, but I'm going to use only the `atp_id` value. The `atp_id` is the OCID for the new Oracle Autonomous Transaction Processing instance.

```
def create_atp(db_client, atp_details):
   # Provision a new ATP Database
   atp_response = db_client.create_autonomous_database(
       create_autonomous_database_details=atp_details)

   # Get the OCID for the new ATP Database, print and return the ID
   atp_id = atp_response.data.id
   print("Created Automated Transaction Processing Database: {}".format(atp_ID))

   return atp_id
```

delete_atp(db_client, atp_id). The `delete_atp` method in `oci_atp.py` accepts the Oracle Autonomous Transaction Processing OCID, which is passed into delete_autonomous_database. This method will *terminate* the Oracle Autonomous Transaction Processing instance. Be careful with this command—there is no undo.

```
def delete_atp(db_client, atp_id):
   # TERMINATE the automated transaction processing database
   db_client.delete_autonomous_database(atp_id)
   print("TERMINATED Automated Transaction Processing Database: {}".format(atp_id))
```

get_wallet(db_client, atp_id, password, fileName). In a previous article ("Getting Started with Autonomous"), I explained that you will need an Oracle wallet to access your new Oracle Autonomous Transaction Processing instance and I walked through the steps to manually download it through the Oracle Cloud dashboard.

Instead of using the dashboard to get the wallet this time, I'll use the `get_wallet` method in `oci_atp.py` to generate and download a wallet.

The `get_wallet` method accepts the `database_client` object, the OCID of the Oracle Autonomous Transaction Processing instance, a password for the .zip file, and the path/filename to use when creating the .zip file. After generating and downloading the new wallet, the `get_wallet` method will create a .zip file, using the response data.

```
def get_wallet(db_client, atp_id, password, fileName):

   # Create a wallet details object
   atp_wallet = oci.database.models.GenerateAutonomousDatabaseWalletDetails()
```

```
# Set the password
atp_wallet.password = password

# Generate the wallet and store the response object
atp_wallet_response = db_client.generate_autonomous_database_wallet(
    autonomous_database_id = atp_id,
    generate_autonomous_database_wallet_details = atp_wallet,
)

# Create the new .zip file using the response data
with open(fileName, "wb") as f:
    for data in atp_wallet_response.data:
        f.write(data)

   print("Wallet Downloaded \x1b[31m***Keep this fileName secure. It can be used to
access your Database!***\x1b[0m")
```

The complete oci_atp.py module. Here are the methods just described—`create_atp`, `delete_atp`, and `get_wallet`—in the complete `oci_atp.py` utility module.

```
import oci

def create_atp(db_client, atp_details):
   # Provision a new ATP Database
   atp_response = db_client.create_autonomous_database(
       create_autonomous_database_details=atp_details)
```

```
    # Get the OCID for the new ATP Database, print and return the ID
    atp_id = atp_response.data.id
    print("Created Automated Transaction Processing Database: {}".format(atp_id))

    return atp_id

def delete_atp(db_client, atp_id):
    # TERMINATE the automated transaction processing database
    db_client.delete_autonomous_database(atp_id)
    print("TERMINATED automated transaction processing database: {}".format(atp_id))

def get_wallet(db_client, atp_id, password, fileName):
# Create a wallet details object
atp_wallet = oci.database.models.GenerateAutonomousDatabaseWalletDetails()

# Set the password
atp_wallet.password = password

# Generate the wallet and store the response object
atp_wallet_response = db_client.generate_autonomous_database_wallet(
    autonomous_database_id = atp_id,
    generate_autonomous_database_wallet_details = atp_wallet,
)

# Create the new .zip file using the atp_wallet_response data
with open(fileName, "wb") as f:
```

```
        for data in atp_wallet_response.data:
            f.write(data)

    print("Wallet Downloaded \x1b[31m***Keep this fileName secure. It can be used to
access your Database!***\x1b[0m")
```

USING THE OCI_ATP.PY MODULE

To access the `oci_atp.py` module, you first need to create an oci.database.DatabaseClient, `db_client`.

The `db_client` is created with the configuration details you included when you set up the command-line interface for Oracle Cloud Infrastructure.

You can load the default configuration.

```
config = oci.config.from_file()
```

Or, if you want to use a different file or configuration, you can load the details, as I did when I verified the SDK above.

```
config = oci.config.from_file("~/.oci/config", "DEFAULT")
```

You can now create the database client, using the `config` object.

```
# Initialize the client
db_client = oci.database.DatabaseClient(config)
```

Next you need to create and populate the `atp_details` object you will use to

define the new Oracle Autonomous Transaction Processing instance.

```
   # Create the CreateAutonomousDatabaseDetails object
   atp_details = oci.database.models.CreateAutonomousDatabaseDetails()

   # Populate the details used to create the ATP Database
   atp_details.admin_password = "Welcome1!SDK" # ****(Remember to change this to a good
password)*****
   atp_details.compartment_id = os.getenv("OCI_COMPARTMENT") or config["tenancy"] # Use
a specific compartment or default to the Root compartment
   atp_details.db_name = "PySdkAtpTemp"
   atp_details.display_name = "Python SDK ATP Example"
   atp_details.cpu_core_count = 1
   atp_details.data_storage_size_in_tbs = 1
   atp_details.license_model = atp_details.LICENSE_MODEL_BRING_YOUR_OWN_LICENSE
```

Finally, provision the new Oracle Autonomous Transaction Processing instance, using the `create_atp` method.

```
   # Provision the ATP Database
   atp_id = oci_atp.create_atp(db_client, atp_details)
```

It will take a few minutes for the database to be fully provisioned and available.

Rather than manually watching the status on the Oracle Cloud dashboard, I use the `wait_until` method. This method will watch the new Oracle Autonomous Transaction Processing instance until the `lifecycle_state` value is `AVAILABLE`. By

default, the method will check the status at an interval of `max_interval_seconds=30`, but I've changed it to 60 seconds. Also by default, the `wait_until` method will wait for only 12,000 seconds, but I've changed it to wait for three hours.

```
# Wait for the new ATP Database to become available
get_atp_response = oci.wait_until(
    db_client,
    db_client.get_autonomous_database(atp_id),
    'lifecycle_state',
    'AVAILABLE',
    max_interval_seconds=60,
    max_wait_seconds=21600,
    wait_callback = wait_callback
)
```

The use of `wait_callback` is optional, but I like to have the visual it provides, so I know that the instance creation is still working. I use this method to print the status if it changes or a "." if not.

```
atp_status = "NONE"

def wait_callback(attempts, results):
   global atp_status
   if atp_status != results.data.lifecycle_state:
       atp_status = results.data.lifecycle_state
       print(atp_status, end='', flush=True)
```

```
    else:
        print('.', end='', flush=True)

print("AVAILABLE")
```

The output should look like this:

```
PROVISIONING.......AVAILABLE
```

Once the database is available, you can download the wallet. The file will be a .zip file, so name it correctly. You can add a path to the `fileName` value if you'd like to store the wallet in another location.

```
    # Wallet values
    walletPassword = "Wallet1!SDK" # CHANGE THE PASSWORD! - ****(Remember to change this
to a good password)*****
    fileName = "{}_wallet.zip".format(atp_details.db_name)

    # Download the wallet
    oci_atp.get_wallet(db_client, atp_id, walletPassword, fileName)
```

The code for atp_example.py. The following `atp_example.py` example module calls the methods in the `oci_atp.py` utility module to create a new Oracle Autonomous Transaction Processing instance and download the .zip file.

```
import oci, os
```

```
import oci_atp

config = oci.config.from_file()

atp_status = "NONE"

def wait_callback(attempts, results):
   global atp_status
   if atp_status != results.data.lifecycle_state:
       atp_status = results.data.lifecycle_state
       print(atp_status, end='', flush=True)
   else:
       print('.', end='', flush=True)

if __name__ == "__main__":
   # Initialize the client
   db_client = oci.database.DatabaseClient(config)

   # Create the CreateAutonomousDatabaseDetails object
   atp_details = oci.database.models.CreateAutonomousDatabaseDetails()

   # Populate the details used to create the ATP Database
   atp_details.admin_password = "Welcome1!SDK" # ****(Remember to change this to a good
password)*****
   atp_details.compartment_id = os.getenv("OCI_COMPARTMENT") or config["tenancy"] # Use
a specific compartment or default to the Root compartment
```

```
atp_details.db_name = "PySdkAtpTemp"
atp_details.display_name = "Python SDK ATP Example"
atp_details.cpu_core_count = 1
atp_details.data_storage_size_in_tbs = 1
atp_details.license_model = atp_details.LICENSE_MODEL_BRING_YOUR_OWN_LICENSE

# Provision the ATP Database
atp_id = oci_atp.create_atp(db_client, atp_details)

# Wait for the new ATP Database to become available
get_atp_response = oci.wait_until(
    db_client,
    db_client.get_autonomous_database(atp_id),
    'lifecycle_state',
    'AVAILABLE',
    max_interval_seconds=60,
    max_wait_seconds=21600,
    wait_callback = wait_callback
)

print("AVAILABLE")

# Wallet values
walletPassword = "Wallet1!SDK" # CHANGE THE PASSWORD! - ****(Remember to change this
to a good password)*****
fileName = "{}_wallet.zip".format(atp_details.db_name)
```

```
# Download the wallet
oci_atp.get_wallet(db_client, atp_id, walletPassword, fileName)
```

Test. Delete. More. After creating your new Oracle Autonomous Transaction Processing instance, you can test your connection by following the steps in the "Getting Started with Autonomous" article mentioned above.

I did not include a call to the `delete_atp` method, but it could be called like this:

```
oci_atp.delete_atp(db_client, "ocid1.autonomousdatabase.oc1.phx. aBigLongGuidString")
```

With the Python SDK for Oracle Cloud Infrastructure, you can create, configure, and delete almost all of your Oracle Cloud objects, and this article should get you started on using the SDK for those operations with Oracle Autonomous Transaction Processing. You can find some other examples on GitHub. If you run into problems with these examples, please reach out to me and I will do my best to help.

Blaine Carter is the Oracle developer advocate for open source. He applies his exploratory eye and tinkering inclinations to the intersection of open source software and Oracle Database.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

LEARN more about Oracle Autonomous Transaction Processing.

TRY Oracle Autonomous Transaction Processing.

ORACLE DATABASE

SODA and PL/SQL, Part 2

Use the SODA API for PL/SQL to read and write to SODA documents in Oracle Database.

In my first article on SODA (Simple Oracle Document Access) and PL/SQL, I showed you how to perform SODA operations in Oracle Database 18c and higher to create SODA collections and documents in your schema.

In this article, I explore how to use methods of a new SODA operations type to find documents so that you can perform read and write operations on them.

A NEW OPERATIONS TYPE

Oracle Database provides the `SODA_OPERATION_T` type (added in Oracle Database 18.3) to specify all read and write operations, except for inserts (which I covered in my first article on SODA and PL/SQL). It contains methods for finding documents in a collection, replacing those documents, and even removing a document from a collection.

The usual approach to take advantage of these operations is to call the `FIND` method of `SODA_COLLECTION_T`. `FIND` returns an instance of the operation type of the collection. You then use methods of `SODA_OPERATION_T` to perform the desired actions.

The following are `SODA_OPERATION_T` methods that perform read operations:

- `get_cursor` returns a pointer to a cursor that enables you to fetch all specified documents (through the API, not SQL).
- `get_one` gets a single document.
- `count` returns the count of documents that match your criteria.

And the following are `SODA_OPERATION_T` methods that perform write operations:

- `replace_one` replaces the contents of a document.
- `replace_one_and_get` replaces the contents and returns the key to the new document.
- `remove` removes the specified document.

These methods are *terminal* methods: They are the last method you invoke in a chain of invocations to perform the read or write operation.

Every method call before that in the chain is a *nonterminal* method, which helps you get to precisely the document(s) you want to read or write. You can think of the nonterminal methods as all contributing in their way to a "`WHERE` clause" applied to the search through the collection. These are the nonterminal methods:

- `key` specifies the key for the document you want to operate on.
- `keys` provides a comma-delimited list of keys.
- `Filter` uses JSON query-by-example syntax to specify documents of interest, an ordering of the documents returned, and more.
- `version` specifies the document version for your find operation.

- `skip` skips *N* documents and is used for pagination.
- `limit` limits the retrieval to *N* documents and is also used for pagination.

LET'S GET STARTED

To perform operations on a document or a collection of documents, I first need to create that collection. Here's the data I'll be working with in this article:

```
DECLARE
  l_collection   soda_collection_t;
  l_status       PLS_INTEGER;
BEGIN
  l_collection := dbms_soda.create_collection ('FriendsWithDocs');

l_status :=
  l_collection.insert_one (
    soda_document_t (
      b_content   => UTL_RAW.cast_to_raw (
                       '{"friend_type":1,
                         "friend_name":"Lakshmi"},
                         "favorites": [{"song" : "Somewhere over the Rainbow"},
                                       {"spice" : "tamarind"},
                                       {"flavor" : "cherry"} ]}')));

l_status :=
  l_collection.insert_one (
    soda_document_t (
```

```
        b_content   => UTL_RAW.cast_to_raw (
                         '{"friend_type":2,
                           "friend_name":"Sally"},
                           "favorites": [{"color" : "blue"},
                                         {"flavor" : "chocolate"},
                                         {"flower" : "rose"} ]}')));

l_status :=
  l_collection.insert_one (
    soda_document_t (
      b_content   => UTL_RAW.cast_to_raw (
                       '{"friend_type":2,
                       "friend_name":"Jorge",
                       "favorites": [{"color" : "green"},
                                     {"flavor" : "chocolate"},
                                     {"tree" : "oak"} ]}')));
END;
```

The data comes from three friends (`friend_name`), of two different friend types (`friend_type`)—I will leave it up to your imagination to choose a couple of types of friends—and it includes a variety of favorites (`favorites`) in a nested JSON array.

Note: I use `cast_to_raw` to convert my string to a BLOB. This is OK for demonstrations and small JSON documents but could cause issues with larger documents. A more robust (but also more complicated) solution, suitable for production, would be to use `DBMS_LOB` subprograms.

OPERATION CHAINING

Method chaining is a common practice in object-oriented programming but less familiar in the procedural world (and PL/SQL is a procedural language). When coding your read and write operations on documents, you will usually chain together several of the nonterminal methods listed previously.

First, here's a block of code that does *not* use method chaining:

```
DECLARE
  l_collection   soda_collection_t;
  l_operation    soda_operation_t;
  l_cursor       soda_cursor_t;
  l_status       BOOLEAN;
BEGIN
  l_collection := dbms_soda.open_collection ('FriendsWithDocs');

  l_operation := l_collection.find ();

  l_cursor := l_operation.get_cursor ();

  l_status := l_cursor.close;
END;
```

I start the block by opening my collection. Then I call the `FIND` method to return an instance of the `SODA_OPERATION_T` type for *all* the documents in the collection. (I will show you later in the article how to filter that `FIND` if you don't want them all). I then

call the `get_cursor` method to return a cursor I can use to iterate through the documents. Finally, because it is the right thing to do, I close the cursor.

Here's that same functionality, now using the chained method approach.

```
DECLARE
  l_cursor   soda_cursor_t;
  l_status   BOOLEAN;
BEGIN
  l_cursor :=
    dbms_soda.open_collection ('FriendsWithDocs').find ().get_cursor ();

  l_status := l_cursor.close;
END;
```

There is no need to declare the collection or operation instances. I simply "chain" the method calls together, using dot notation. So: fewer declarations, fewer lines of code. Each call to `open_collection` involves a SQL query against the underlying tables. So if you plan to perform multiple find operations on the same collection, you should open it in one step, returning the collection instance, and then use that instance for multiple chained operations.

When you first start working with SODA elements, you might want to break these nonterminal methods out so you can more easily debug the steps in your chain. Once you are familiar with SODA, however, you will find the chained approach preferable.

ITERATING THROUGH ALL DOCUMENTS

The SODA API offers a cursor to make it easy for you to iterate through all documents returned by the `FIND` and `FILTER` methods.

```
DECLARE
  l_document      soda_document_t;
  l_cursor        soda_cursor_t;
  l_content_shown BOOLEAN := FALSE;
  l_status        BOOLEAN;
BEGIN
  l_cursor :=
    dbms_soda.open_collection ('FriendsWithDocs').find ().get_cursor ();

-- Loop through the cursor
WHILE l_cursor.has_next
LOOP
  l_document := l_cursor.NEXT;

  IF l_document IS NOT NULL
  THEN
    DBMS_OUTPUT.put_line ('Document key: ' || l_document.get_key);

    IF NOT l_content_shown
    THEN
      /* Just show content once to reduce output volume */
```

```
      l_content_shown := TRUE;
      DBMS_OUTPUT.put_line (
        'Content: ' || json_query (l_document.get_blob, '$' PRETTY));
    END IF;

    DBMS_OUTPUT.put_line('Creation timestamp: ' || l_document.get_created_on);
    DBMS_OUTPUT.put_line('Last modified timestamp: ' || l_document.get_last_modified);
    DBMS_OUTPUT.put_line('Version: ' || l_document.get_version);
  END IF;
END LOOP;

l_status := l_cursor.close;
END;
```

This cursor block finds all the documents in a collection, returns a cursor to those documents, and then uses these cursor methods to iterate:

- `has_next` returns `TRUE` if there is more to "fetch."
- `next` gets the next document.

Note that for "basic" find operations in which order is not specified and no pagination methods are called, no ordering is applied to the documents. When calls to `skip()` or `limit()` are involved, the order of the documents is based on the internal key for that document, a universal unique identifier (UUID). I'll explore how to override this sorting later in this article.

Here's the output from running the above cursor block:

```
Document key: 29D6E98467EC4FA7BF18572986FF8925
Content: {
  "friend_type" : 1,
  "friend_name" : "Lakshmi",
  "favorites" :
  [
    {
      "song" : "Somewhere over the Rainbow"
    },
    {
      "spice" : "tamarind"
    },
    {
      "flavor" : "cherry"
    }
  ]
}
Creation timestamp: 2019-05-03T18:39:46.787333Z
Last modified timestamp: 2019-05-03T18:39:46.787333Z
Version: 698637C8DC1A843E0078D77DD1680D17A4E8B9A8108B87081BE44902AD9484FD
Document key: 6CFE30D607284F15BF0F59B1FD952842
Creation timestamp: 2019-05-03T18:39:46.787884Z
Last modified timestamp: 2019-05-03T18:39:46.787884Z
Version: D5B1E2370AB9A6C15FDCC0DB53E38D1D3B5306E57B039040E22BF7619A5166B1
Document key: BE2CA098F5554F5DBFF9454A8F02DEA4
```

```
Creation timestamp: 2019-05-03T18:39:46.788234Z
Last modified timestamp: 2019-05-03T18:39:46.788234Z
Version: 63A7781DBA6117B6CBACECF5CB9959054D1FF6D0852317701D9A10F9A43DABA4
```

FINDING A SPECIFIC DOCUMENT

Sometimes you'll want to iterate through multiple documents, and sometimes you'll want to get just one document. You can get that document by referencing a specific key.

Usually those keys are generated internally and passed back and forth between the database and the application. For purposes of the demonstration below, I create a collection and use the `metadata` parameter to tell PL/SQL that I will provide the keys myself.

That way I know the key I can use to retrieve the document.

```
DECLARE
  l_document       soda_document_t;
  l_collection     soda_collection_t;
  l_metadata       VARCHAR2 (4000)
    := '{"keyColumn" : { "assignmentMethod" : "CLIENT" }}';
  l_status         PLS_INTEGER;
BEGIN
  l_collection := dbms_soda.create_collection ('MyOwnKeys', l_metadata);

  DBMS_OUTPUT.put_line (
      'Collection specification: '
```

```
 || json_query (l_collection.get_metadata, '$' PRETTY));

l_status :=
  l_collection.insert_one (
    soda_document_t ('FriendLakshmi',
      b_content   => UTL_RAW.cast_to_raw (
                           '{"friend_type":1,"friend_name":"Lakshmi"}')));

l_document :=
  l_collection
    .find ()
    .key ('FriendLakshmi')
    .get_one;

  DBMS_OUTPUT.put_line ('Key: ' || l_document.get_key);
  DBMS_OUTPUT.put_line ('Content: ');
  DBMS_OUTPUT.put_line (
    l_document.get_key|| '+'||json_query (l_document.get_blob, '$' PRETTY));
END;
```

The output below shows first the information about the collection, verifying that the `assignmentMethod` for the key is `CLIENT`. Following that is the information for that document.

```
Collection specification: {
  "schemaName" : "HR",
```

```
"tableName" : "MyOwnKeys",
"keyColumn" :
{
  "name" : "KEY",
  "sqlType" : "VARCHAR2",
  "maxLength" : 255,
  "assignmentMethod" : "CLIENT"
},
"contentColumn" :
{
  "name" : "JSON_DOCUMENT",
  "sqlType" : "BLOB",
    "compress" : "NONE",
    "cache" : false,
    "encrypt" : "NONE",
    "validation" : "STANDARD"
  },
  "readOnly" : false
}
Key: FriendLakshmi
Content:
FriendLakshmi+{
  "friend_type" : 1,
  "friend_name" : "Lakshmi"
}
```

FINDING DOCUMENTS THAT MATCH FILTERS

I've shown you how to get all the documents in a collection and a single document by key, but you have much more flexibility than that. You can also use JSON query-by-example syntax to filter your document retrieval.

You do this by adding the `filter` method to the chained invocation, right after `find`, as in

```
dbms_soda.open_collection ('FriendsWithDocs').find ().filter (l_filter).get_cursor ();
```

In the block below, I set the filter to find only those friends of type "2" (whatever that might be!).

```
DECLARE
  l_document   soda_document_t;
  l_cursor     soda_cursor_t;
  l_filter     VARCHAR2 (128) := '{"friend_type" : "2"}';
  l_status     BOOLEAN;
BEGIN
  l_cursor :=  dbms_soda.open_collection ('FriendsWithDocs').find ().filter (l_filter)
.get_cursor ();

  WHILE l_cursor.has_next
  LOOP
    l_document := l_cursor.NEXT;

    IF l_document IS NOT NULL
```

```
     THEN
       DBMS_OUTPUT.put_line ('Document key: ' || l_document.get_key);
       DBMS_OUTPUT.put_line (
         'Content: ' || json_query (l_document.get_blob, '$' PRETTY));
   END IF;
   END LOOP;

   l_status := l_cursor.close;
END;

Document key: 6CFE30D607284F15BF0F59B1FD952842
Content: {
"friend_type" : 2,
"friend_name" : "Sally",
"favorites" :
[
  {
    "color" : "blue"
  },
  {
    "flavor" : "chocolate"
  },
  {
    "flower" : "rose"
  }
]
```

```
}
Document key: BE2CA098F5554F5DBFF9454A8F02DEA4
Content: {
  "friend_type" : 2,
  "friend_name" : "Jorge",
  "favorites" :
  [
    {
      "color" : "green"
    },
    {
      "flavor" : "chocolate"
    },
    {
      "tree" : "oak"
    }
  ]
}
```

I can also use the query-by-example syntax to search for documents whose nested array elements match a certain criterion. The code below finds and displays all friends whose favorite flavor is chocolate:

```
DECLARE
  l_document   soda_document_t;
  l_cursor     soda_cursor_t;
```

```
  l_filter     VARCHAR2 (128) := '{"favorites.flavor" : "chocolate"}';
  l_status     BOOLEAN;
BEGIN
  l_cursor :=  dbms_soda.open_collection ('FriendsWithDocs').find ()
.filter (l_filter).get_cursor ();

  WHILE l_cursor.has_next
  LOOP
    l_document := l_cursor.NEXT;

    IF l_document IS NOT NULL
    THEN
      DBMS_OUTPUT.put_line ('Document key: ' || l_document.get_key);
      DBMS_OUTPUT.put_line (
         'Content: ' || json_query (l_document.get_blob, '$' PRETTY));
    END IF;
  END LOOP;

  l_status := l_cursor.close;
END;

Document key: 6CFE30D607284F15BF0F59B1FD952842
Content: {
  "friend_type" : 2,
  "friend_name" : "Sally",
```

```
                    "favorites" :
                    [
                      {
                        "color" : "blue"
                      },
                      {
                        "flavor" : "chocolate"
                      },
                      {
                        "flower" : "rose"
                      }
                    ]
                  }
                  Document key: BE2CA098F5554F5DBFF9454A8F02DEA4
                  Content: {
                    "friend_type" : 2,
                    "friend_name" : "Jorge",
                    "favorites" :
                    [
                      {
                        "color" : "green"
                      },
                      {
                        "flavor" : "chocolate"
                      },
```

```
    {
      "tree" : "oak"
    }
  ]
}
```

I can also find all the documents whose keys match the provided list. I demonstrate this feature below by again specifying a document in which I can provide the keys myself. I do this with the `metadata` parameter and an `assignmentMethod` set to `CLIENT`.

```
DECLARE
  l_collection   soda_collection_t;
  l_metadata     VARCHAR2 (4000)
     := '{"keyColumn" : { "assignmentMethod": "CLIENT" }}';
  l_keys         soda_key_list_t;
  l_cursor       soda_cursor_t;
  l_status       BOOLEAN;
  l_document     soda_document_t;
BEGIN
  l_collection := dbms_soda.create_collection ('ColorKeys', l_metadata);

  l_status :=
    l_collection.insert_one (
      soda_document_t (
```

```
      'Red',
     b_content   => UTL_RAW.cast_to_raw ('{"thing":"blood"}'))) = 1;
l_status :=
  l_collection.insert_one (
    soda_document_t (
      'Green',
      b_content   => UTL_RAW.cast_to_raw ('{"thing":"grass"}'))) = 1;

l_keys := soda_key_list_t ('Green');

l_cursor := l_collection.find ().keys (l_keys).get_cursor ();

  WHILE l_cursor.has_next
  LOOP
    l_document := l_cursor.NEXT;

    IF l_document IS NOT NULL
    THEN
      DBMS_OUTPUT.put_line ('Key: ' || l_document.get_key);
      DBMS_OUTPUT.put_line (
         json_query (l_document.get_blob, '$' PRETTY));
    END IF;
  END LOOP;

  l_status := l_cursor.close;
END;
```

And I see this output:

```
Key: Green
{
  "thing" : "grass"
}
```

REPLACING DOCUMENTS

Suppose I want to change a document in my collection. Currently I will need to replace it with another. So I first create or receive the document; find the document, using its key; and use `replace_one` or `replace_one_and_get`.

In the block below, I use the same `ColorKeys` collection created in the previous section and replace the `Red` document with a `Blue` document.

```
DECLARE
  l_collection   SODA_COLLECTION_T;
  l_document     SODA_DOCUMENT_T;
  l_new_document SODA_DOCUMENT_T;
BEGIN
  l_collection := DBMS_SODA.open_collection('ColorKeys');
  l_document := SODA_DOCUMENT_T(
                b_content => utl_raw.cast_to_raw('{"Blue" : "Bluebird"}'));
  l_new_document := l_collection.find().key('Red').replace_one_and_get(l_document);

  IF l_new_document IS NOT NULL
  THEN
```

```
    DBMS_OUTPUT.put_line('Document:');
    DBMS_OUTPUT.put_line('Key: ' || l_new_document.get_key);
  END IF;

  COMMIT;
END;
```

If I do not need the result document (the document returned by and_get methods), I can call the `replace_one` method. That replaces the specified document and returns a status value: 1 if the document was replaced, 0 otherwise.

REMOVING DOCUMENTS

My previous article on SODA showed you how to remove a *single* document. You can now use a `remove` method in `SODA_OPERATION_T` to remove one or *many* documents, once you have found, via key or filter, the document or documents you wish to remove. Here is an example of single-document removal:

```
DECLARE
  l_collection  SODA_COLLECTION_T;
  l_status NUMBER;
BEGIN
  l_collection := DBMS_SODA.open_collection('ColorKeys');

  l_status:= l_collection.find().key('Green').remove;

  IF  l_status = 1
```

```
  THEN
     DBMS_OUTPUT.put_line('Document removed');
  END IF;
END;
```

And in the block below, I use query-by-example to remove all documents from `FriendsWithDocs` whose `friend_type` is equal to 2.

```
DECLARE
  l_collection  SODA_COLLECTION_T;
  l_status NUMBER;
BEGIN
  l_collection := DBMS_SODA.open_collection('FriendsWithDocs');

  l_status:= l_collection.find().filter('{"friend_type" : "2"}').remove();

  IF  l_status = 1
  THEN
     DBMS_OUTPUT.put_line(Friend type 2 documents removed');
  END IF;
END;
```

PAGINATION FEATURES

Suppose your collection has 1,000,000 documents and you want to work with batches of 100 documents at a time. `SODA_COLLECTION_T` offers the `skip()` and

`limit()` nonterminal methods to help you get the job done.

In the following block, I create a collection and stuff it full of 100 very simple documents. I then use `find()`, `skip()`, and `limit()` to skip the first 50 documents and get just the next 5.

```
DECLARE
  l_collection   soda_collection_t;
  l_document     soda_document_t;
  l_cursor       soda_cursor_t;
  l_status       BOOLEAN;
BEGIN
  l_collection := dbms_soda.create_collection ('SillyCollection');

FOR indx IN 1 .. 100
LOOP
  l_status :=
    l_collection.insert_one (
      soda_document_t (
        b_content   => UTL_RAW.cast_to_raw (
                          '{"myIndex":'
                       || indx
                       || ',"myValue":"'
                       || CASE MOD (indx, 2)
                            WHEN 0 THEN 'Even'
                          ELSE 'Odd'
```

```
                                    END
                                || '"}'))) = 1;
END LOOP;

l_cursor := l_collection.find ().skip (50).LIMIT (5).get_cursor;

WHILE l_cursor.has_next
LOOP

  l_document := l_cursor.NEXT;

  IF l_document IS NOT NULL
  THEN
    DBMS_OUTPUT.put_line (
      'Content: ' || json_query (l_document.get_blob, '$' PRETTY));
  END IF;
END LOOP;

  l_status := l_cursor.close;
END;
```

Here's my output:

```
Content: { "myIndex" : 89, "myValue" : "Odd" }
Content: { "myIndex" : 48, "myValue" : "Even" }
```

```
Content: { "myIndex" : 50, "myValue" : "Even" }
Content: { "myIndex" : 2, "myValue" : "Even" }
Content: { "myIndex" : 60, "myValue" : "Even" }
```

But wait! Why don't I see 51 through 55? That's because results are ordered only by the document key values (which are by default generated automatically) unless `skip()` or `limit()` appears in method calls.

If you need to change the ordering, use the `filter` method and specify an `orderby`. In the block below, I order by the `myIndex` value. I specify that the `datatype` is a `number` to ensure the correct ordering (it will otherwise be treated as a string).

```
DECLARE
  l_collection   soda_collection_t;
  l_document     soda_document_t;
  l_cursor       soda_cursor_t;
  l_status       BOOLEAN;
BEGIN
  l_collection := dbms_soda.open_collection ('SillyCollection');

  l_cursor := l_collection. find(). filter('{"$orderby" : [ { "path" : "myIndex",
"datatype" : "number", "order" : "asc"}]}').
    skip(50).limit(5).get_cursor();

  WHILE l_cursor.has_next
  LOOP
    l_document := l_cursor.NEXT;
```

```
    IF l_document IS NOT NULL
     THEN
       DBMS_OUTPUT.put_line (
         'Content: ' || json_query (l_document.get_blob, '$' PRETTY));
    END IF;
  END LOOP;


  l_status := l_cursor.close;
END;
```

And I now see the following output:

```
Content: { "myIndex" : 51, "myValue" : "Odd" }
Content: { "myIndex" : 52, "myValue" : "Even" }
Content: { "myIndex" : 53, "myValue" : "Odd" }
Content: { "myIndex" : 54, "myValue" : "Even" }
Content: { "myIndex" : 55, "myValue" : "Odd" }
```

One thing to keep in mind when you are doing pagination and sorting: The process can be slow if you have lots of documents and you paginate through all of them. Whenever possible, try to restrict the set of documents in advance. Do this with a more complex query-by-example specification, such as

```
qbe = '{"$query" : {"salary" : {"$between" : [10000, 20000]}},
"$orderby" : {"someField" : 1}}';
```

POWERFUL DOCUMENT TOOLS

SODA for PL/SQL will probably never be a "mainstream" feature set for Oracle Database developers. The majority of our work is likely to remain centered on traditional, extremely flexible, proven relational database objects, such as tables and views.

But as more and more UI developers, used to working with JavaScript and document databases, learn that they can also get the job done with Oracle Database, demand for these features and the skills to use them properly will increase.

Steven Feuerstein is a developer advocate for Oracle, specializing in PL/SQL. Feuerstein's books, including Oracle PL/SQL Programming*; videos; and more than 1,500 quizzes at the Oracle Dev Gym (devgym.oracle.com) provide in-depth resources for Oracle Database developers.*

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

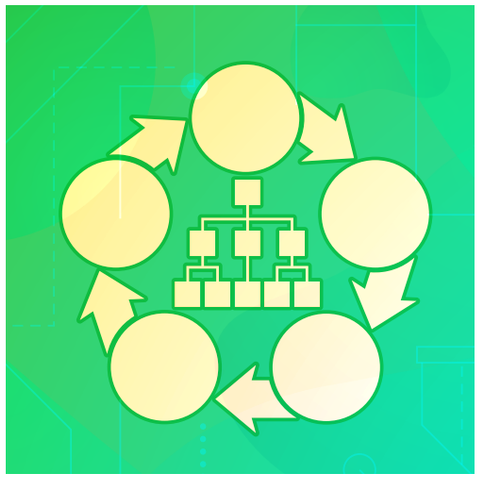
READ "SODA and PL/SQL."

LEARN more about SODA for PL/SQL.

READ an introduction to SODA.

WORK with the LiveSQL script for this article.

BY ARUP NANDA



ORACLE DATABASE

Autonomous Indexing

The new Automatic Indexing feature in Oracle Database 19c detects the need for indexes, creates them, and drops them automatically—without DBA intervention.

John, the chief architect of Acme Travels, walks into the office of the CTO, Tanya, and he can almost smell the frostiness in the air. Acme, a travel products company, has been operating for more than 20 years, and during that time, the company's data model has evolved—as any 20-year-old company's data model would. The developers create new tables, alter the existing tables, write new SQL statements, and modify the existing SQL constantly. To improve performance, they create indexes on the columns proactively, but more often they add indexes later—after a performance issue has been detected.

The DBA manager, Betty, points out that adding indexes to address performance issues is a double-edged sword. It may improve performance for some queries, but it will definitely slow down `INSERT` statements and may negatively impact

`UPDATE` and `DELETE` statements, often enough to negate any perceived performance gains elsewhere. Therefore Betty's team doesn't create indexes willy-nilly but only after a very careful analysis of the impact on all queries, sometimes with the use of Oracle Database's SQL Performance Analyzer feature, and even then, those indexes are almost always created *after* the SQL statements have been issued against the database. When the SQL statements that use the new indexes cause performance issues, DBAs are blamed, and DBAs, in turn, blame the developers for writing the poorly performing queries in the first place. Similarly, as a result of changing queries, indexes created earlier are sometimes no longer needed. The DBAs check for usage of indexes and drop them if they're unused, which is also a time-consuming—not to mention risky—exercise for the DBAs.

Debbie, the development manager, expects the DBAs to perform this analysis 24 hours a day, seven days a week, along with everything else they are doing.

"You've got to be kidding to suggest something that preposterous, especially with our lean staff!" says an enraged Betty.

"Well, *we* certainly can't do it!" responds Debbie. "We follow the Agile methodology for development. We make small changes, perhaps 100 times a week. There is no way for us to slow down and check the impact of each change line by line. The analysis and changes *must* come from your team."

The tension inside the office is so thick you could cut it with a knife. Obviously, Tanya doesn't like it and wants to end it with a solution acceptable to everyone and also beneficial to Acme. She does agree that with Agile development practices and the continuous integration/continuous deployment (CI/CD) philosophy, it's important to maintain the velocity of development. So she also understands that these indexing issues must be dealt with *after* development, not before. But at the same

time, she understands that expecting the DBAs to do the time-consuming analysis and make the risky changes 24/7 is impractical unless she quadruples the DBA staff, which she can't do.

She wonders out loud if the teams should slow down development considerably and include a thorough analysis before deploying applications. No way, Debbie responds, because that would be catastrophic for application delivery. And this time Betty agrees with her, saying that it would not solve the problem entirely, because there will still be indexes to be analyzed—just more in a bunch rather than spread out through the releases. So, although the original problem would still exist, Acme would lose the benefit of the CI/CD approach. It's a lose-lose for everyone.

"Well, so much for solutions," sighs Tanya. "That's why I asked John to be here," she announces. "I'm hoping he knows if there's a way to get us out of this bind."

All eyes turn to John. "Yes, there is," smiles John, trying to warm the chill.

PRIOR SOLUTIONS

John, who until recently was the lead DBA at Acme, reminds everyone how indexes are being managed now. Most of them are created as a part of the performance issues captured and solutions recommended by the advisors that come with Oracle Database, such as Index Advisor, Partition Advisor, and In-Memory Advisor. These advisors alert the DBA to the potential issues and advise if a new index would actually help. However, the responsibility for determining the overall impact and implementing the indexes—both fairly onerous tasks—still lies with the DBAs. For example, the DBAs must determine

- If the recommended index will actually help, based on multiple metrics such as data blocks retrieved, elapsed time savings, and so on

- If the recommended index will affect the data manipulation language (DML) SQL statements negatively
- If the indexes created earlier, once useful, are still useful
- If a fresh analysis of the need for new indexes is required, due to data and structural changes

The last part is particularly tricky, John points out. The table and columns, and even the SQL statements, may not have changed, but if the data pattern has changed, the once-useful indexes may not be useful anymore while also being a drag on the performance of DML statements. Similarly, some indexes not needed earlier could actually help after data pattern changes. And because data pattern changes are almost impossible to track down, the battle for the ideal indexes is usually a losing one.

AUTOINDEXING COMES TO THE RESCUE

This is where a new feature in Oracle Database 19c, Automatic Indexing, comes to the rescue, continues John. The feature acts like a DBA inside the database that evaluates the need for new indexes and the need for the existing indexes, creates new ones if they're needed, and drops them when they are no longer needed. Both single-column and concatenated indexes are considered by the feature, so it covers a majority of the indexes typically used in a database. But the best part, John reminds everyone, is that it's software: It works 24 hours a day, seven days a week, and does not get tired, does not take sick leave, and does not ask for raises. It will do everything Tanya is asking for.

Everyone is listening.

SETUP

Connected to Oracle Database 19c, John starts his demo by executing this SQL statement to enable Automatic Indexing.

```
begin
  dbms_auto_index.configure ('AUTO_INDEX_MODE', 'IMPLEMENT');
end;
/
```

Betty is a bit queasy about running a statement like this, one that implements automatic indexing in the database without her knowing it. "That's understandable," John agrees. John runs another SQL statement to prepare the database for the feature, but he specifies that SQL statements *not* use the new autocreated indexes.

```
begin
   dbms_auto_index.configure ('AUTO_INDEX_MODE','REPORT ONLY');
end;
/
```

This sets in motion the process of identifying the possible indexes. Automatic Indexing, John explains, captures the SQL statements, identifies those from the list that may be helped by indexing, and then creates those indexes—automatically. These autocreated indexes are named with the prefix `SYS_AI`, to differentiate them from the manually created indexes. A new column, named `AUTO` in the `DBA_INDEXES` view, shows a `YES` value for these indexes.

"Wait a minute," interrupts Betty. "It simply creates an index without checking in a representational test system whether the index will actually help or hurt something else? That sounds like an incredibly stupid idea, especially for a machine."

"Not quite," assures John. Creating an index on a whim is not smart. Therefore, the Automatic Indexing feature creates the index as invisible—it is not known to the database optimizer. He explains the concept of an invisible index to the audience with a simple demonstration.

He creates an index, but with the `invisible` keyword.

```
SQL> create index ix_region_id_01 on regions (region_name) invisible;

Index created.
```

Then he checks the plan of a very simple SQL statement against that table.

```
SQL> explain plan for
  2  select * from regions where region_name = 'Europe';

Explained.

SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

Plan hash value: 3077898360
```

```
-------------------------------------------------------------------------------
|Id | Operation          | Name    | Rows| Bytes | Cost (%CPU) | Time       |
-------------------------------------------------------------------------------
| 0 | SELECT STATEMENT   |.        |  1  | 14    |    3   (0)  | 00:00:01   |
|*1 |  TABLE ACCESS FULL| REGIONS |  1  | 14    |    3   (0)  | 00:00:01   |
-------------------------------------------------------------------------------

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

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------

   1 - filter("REGION_NAME"='Europe')

13 rows selected.
```

John calls the attention of the audience to the output and the fact that the SQL statement did not use the index just created where it could have. The reason is simple: The index is marked as invisible to the optimizer.

He then alters the index to make it visible to the optimizer and checks the execution plan of the same SQL statement.

```
SQL> alter index ix_region_id_01 visible;

Index altered.

SQL> explain plan for select * from regions where region_name = 'Europe';

Explained.
```

```
SQL> select * from table(dbms_xplan.display());

Plan hash value: 3897602228

_____________________________________________________________________________________________
|Id| Operation.                    | Name           | Rows| Bytes| Cost (%CPU)| Time     |
_____________________________________________________________________________________________
| 0| SELECT STATEMENT              |                |    1|    14|     2 (0)  | 00:00:01|
| 1|  TABLE ACCESS BY INDEX ROW ..| REGIONS        |    1|    14|     2 (0 ) | 00:00:01|
|*2|   INDEX RANGE SCAN            | IX_REGION_ID_01|    1|      |     1 (0)  | 00:00:01|
_____________________________________________________________________________________________

Predicate Information (identified by operation id):
_____________________________________________________

   2 - access("REGION_NAME"='Europe')
```

Pointing to the output that shows that the `IX_REGION_ID_01` index is used this time, John highlights that this happens because the optimizer now recognizes the presence of this index. Automatic Indexing autocreates indexes as invisible first and then tests the index impact against SQL statements. If the impact is positive—if the SQL statements perform better with an index—then the index is made visible, as John shows; otherwise, that index is marked unusable. To demonstrate that concept, John marks the index unusable and checks the execution plan once again:

```
SQL> alter index IX_REGION_ID_01 unusable;
```

```
Index altered.

SQL> explain plan for select * from regions where region_name = 'Europe';

Explained.

SQL> select * from table(dbms_xplan.display());

Plan hash value: 3077898360

-------------------------------------------------------------------------------
|Id | Operation         | Name    | Rows  | Bytes | Cost(%CPU)| Time     |
-------------------------------------------------------------------------------
| 0 | SELECT STATEMENT  |         |     1 |    14 |    3   (0)| 00:00:01 |
|*1 |  TABLE ACCESS FULL| REGIONS |     1 |    14 |    3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Pointing to the output, John shows that the index is not used now, even though it's visible. He confirms that by using the following SQL:

```
SQL> select VISIBILITY, STATUS from user_indexes where index_name = 'IX_REGION_ID_01';

VISIBILITY  STATUS
----------  --------

VISIBLE     UNUSABLE
```

The Automatic Indexing feature, John explains, checks the SQL statements and evaluates if they can perform better via new indexes. It then creates the indexes as invisible. After that it checks the execution plans of all the SQL statements using that new index. If all the statements show improvement (or no impact), it will make the index visible. If all the SQL statements show degraded performance, the index will remain invisible.

Debbie seems unconvinced. A more likely scenario, she opines, is that some SQL statements will show improvements and some will show degradation due to this autocreated index. What will the fate of the index be then?

John agrees that scenario is most likely, and he assures her that the feature handles such a situation quite well. In this case, the index will be made visible, but Automatic Indexing will create a SQL plan baseline to prevent a SQL statement that regressed in performance from using the index. The other SQL statements—statements that did not regress in performance—will continue to use the index. Debbie nods in satisfaction.

ADVISOR JOBS

"But what component of Oracle Database actually performs this autoindexing," Tanya asks. "Starting with Oracle Database 11g, the database automatically executes multiple tasks," John answers. For example, Oracle's advisors run as automatic tasks and the `DBA_ADVISOR_TASKS` view shows information about these tasks. John pulls up the view data:

```
select *
from dba_advisor_tasks
where owner='SYS'
```

```
order by task_id;

TASK_ID TASK_NAME                      ADVISOR_NAME
      2 SYS_AUTO_SPM_EVOLVE_TASK       SPM Evolve Advisor
      3 SYS_AI_SPM_EVOLVE_TASK         SPM Evolve Advisor
      4 SYS_AI_VERIFY_TASK             SQL Performance Analyzer
      5 SYS_AUTO_INDEX_TASK            SQL Access Advisor
      6 AUTO_STATS_ADVISOR_TASK        Statistics Advisor
      7 INDIVIDUAL_STATS_ADVISOR_TASK  Statistics Advisor
```

He points out the `SYS_AUTO_INDEX_TASK` task and two tasks with `_AI_` in their name. These are the tasks behind the Automatic Indexing feature.

ADMINISTRATION

Betty's interest is piqued now. "How do the DBAs administer this feature?" she asks. John reminds her that the DBAs actually do not need to implement anything; the system detects the need to create indexes automatically and creates them, and it even drops them when they are not needed. So the DBAs' involvement is limited to

- Configuring the properties and parameters
- Getting reports on automatic indexing usage

To set the default tablespace of the autocreated indexes to `USER_AI`, John uses the following:

```
dbms_auto_index.configure ('AUTO_INDEX_DEFAULT_TABLESPACE', 'USER_AI');
```

But autocreated indexes may overwhelm the space in that tablespace. To prevent

that, John executes the following to ensure that only 50% of the tablespace is used:

```
dbms_auto_index.configure ('AUTO_INDEX_SPACE_BUDGET', '50');
```

Sometimes third-party application vendors don't allow index creation or deletion outside of their control. "Will this feature break those applications?" asks one of the DBAs. "Not at all," assures John. To exclude a specific schema, John uses

```
dbms_auto_index.configure ('AUTO_INDEX_EXCLUDE_SCHEMA', 'SCOTT');
```

"But I am not comfortable that all these important activities would be performed by the system under the covers without our knowing about it," says Betty. Others chime in as well. The actions are hardly opaque, John assures them. All the current and historical actions are available in several data dictionary views. He points to the following types of information and views containing them to his audience:

To get . . .	Use this view
The history of Automatic Indexing task executions	`DBA_AUTO_INDEX_EXECUTIONS`
Statistics related to automatic indexes	`DBA_AUTO_INDEX_STATISTICS`
Actions performed on automatic indexes	`DBA_AUTO_INDEX_IND_ACTIONS`
Actions performed on SQL to verify automatic indexes	`DBA_AUTO_INDEX_SQL_ACTIONS`
The history of configuration settings related to automatic indexes	`DBA_AUTO_INDEX_CONFIG`

But the easiest way to monitor Automatic Indexing is to use the built-in report feature. The `DBMS_AUTO_INDEX.REPORT_ACTIVITY` function returns a CLOB containing all the relevant details. John pulls the data for activities between the 20th and 21st of April.

```
declare
  report clob := null;
begin
  report := DBMS_AUTO_INDEX.REPORT_ACTIVITY (
              activity_start => TO_TIMESTAMP('2019-04-20', 'YYYY-MM-DD'),
              activity_end   => TO_TIMESTAMP('2019-04-21', 'YYYY-MM-DD'),
              type           => 'TEXT',
              section        => 'SUMMARY',
              level          => 'BASIC');
end;
```

This produces a report containing all the needed information. Here is part of the report:

```
_______________________________________________________________________________

GENERAL INFORMATION

_______________________________________________________________________________

Activity start              : 20-APR-2019 00:00:00
Activity end                : 20-APR-2019 00:00:00
Executions completed.       : 27
```

```
Executions interrupted      : 2
Executions with fatal error : 0
_____________________________________________________________________________

SUMMARY (AUTO INDEXES)
_____________________________________________________________________________

Index candidates                                : 98
Indexes created (visible / invisible)           : 21/0
Space used (visible / invisible).               : 312.23 MB (312.23 MB / 0 MB)
Indexes dropped                                 : 2
SQL statements verified                         : 312
SQL statements improved (improvement factor)    : 115 (3x)
SQL statements disallowed from auto indexes     : 34
Overall improvement factor                      : 3x
_____________________________________________________________________________

SUMMARY (MANUAL INDEXES)
_____________________________________________________________________________

Unused indexes (visible / invisible) : 9 (6 / 3)
Space used (visible / invisible)     : 281 MB (183 MB / 98 MB)
Unusable indexes                     : 0

INDEX DETAILS
_____________________________________________________________________________

1. The following indexes were created:*: invisible
_____________________________________________________________________________
```

```
_____________________________________________________________________________

|Owner |Table       |Index                |Key         |Type   |Properties|
_____________________________________________________________________________

|HR    |DEPARTMENTS |SYS_AI_0urcv8chmxu20 |LOCATION_ID |B-TREE |NONE      |
|HR    |LOCATIONS   |SYS_AI_1hgrs7xdghs31 |CITY        |B-TREE |NONE      |
... report truncated to save space ...
_____________________________________________________________________________

VERIFICATION DETAILS
_____________________________________________________________________________

1. The performance of the following statements improved:
_____________________________________________________________________________

Schema Name         : HR
SQL ID              : 3dfa28psdfe73
SQL Text            : select * from regions where region_name = 'Europe';
Improvement Factor  : 2x

PLANS SECTION
_____________________________________________________________________________

Original
_________________________________

Plan hash value: 3077898360
_____________________________________________________________________________

| Id | Operation          | Name    | Rows| Bytes | Cost (%CPU) | Time    |
_____________________________________________________________________________

|  0 | SELECT STATEMENT   |         |    1|    14 |.     3   (0)| 00:00:01|
| *1 |  TABLE ACCESS FULL | REGIONS |    1|    14 |      3   (0)| 00:00:01|
_____________________________________________________________________________
```

```
With Auto Indexes
_____________________________

Plan hash value: 3897602228
_____________________________________________________________________________________
|Id | Operation             | Name                 | Rows| Bytes| Cost (%CPU)| Time    |
_____________________________________________________________________________________
| 0 | SELECT STATEMENT      |                      |    1|    13|     2(0)   | 00:00:01|
| 1 |  TABLE ACCESS BY IN...| REGIONS              |    1|    13|     2(0)   | 00:00:01|
|*2 |   INDEX RANGE SCAN    | SYS_AI_8hjkdfss93kdf|    1|      |     1(0).  | 00:00:01|
_____________________________________________________________________________________

Predicate Information (identified by operation id):

   2 - access("REGION_NAME"='Europe')
... report truncated to save space ...
```

The report, John explains, provides vital information on the activities of the Automatic Indexing feature, such as when it ran, how many SQL statements it considered, how many automatic indexes were created, how much space they consumed, and so on. In the subsequent sections, it also shows the indexes automatically created on columns. The last part of the report shows the SQL statements and the execution plans before and after the autocreated indexes, to confirm the improvements. This is the report both Debbie and Betty can examine to see the effectiveness of the Automatic Indexing feature at both a high and detail level.

SAFETY MECHANISMS

Debbie is still somewhat skeptical. During the next evaluation of SQL statements by this feature, she explains, an existing SQL statement will likely inspire the automatic creation of an index that has *already been created* but not used. And Automatic Indexing will once again be forced to evaluate that index and again discard it as not useful. Isn't this a vicious cycle of wasted resources, she asks, whereas a human would've avoided it?

Absolutely, agrees John; this would be a vicious cycle. To prevent it, he explains, the feature marks those SQL statements, and subsequent executions exclude these SQL statements from consideration and avoid getting into vicious cycles of repeated evaluation, creation, and invisibility of indexes.

Debbie is relieved but has another doubt. Sometimes the system simply won't know the negative impact of an autocreated index as well as a human performance tuner. In this case, the SQL statement using the automatic index will simply be detrimental. Is there a way to suppress the use of automatic indexes for a query proactively, she wants to know.

"Of course there is," replies John. If Debbie wants the optimizer not to use autocreated indexes for a specific SQL statement, she can simply include a hint:

```
select /*+ NO_USE_AUTO_INDEXES */ from regions where region_name = 'Europe';
```

Likewise, the `USE_AUTO_INDEXES` hint acts just the opposite way: It forces the SQL statement to use the autocreated indexes, if they are available.

"But that's not all," John continues. "Automatic Indexing includes several additional safety features to protect the database from damage." He lists a handful of key ones:

- The system ignores autocreated indexes for any SQL statements run for the first time. This prevents evaluation of one-off SQL that is never issued again and wouldn't have been able to benefit from the indexes anyway.
- DBAs can disable the autoindex job for specific periods of time, so as not to affect normal processing.
- Using the Resource Manager feature of Oracle Database, DBAs can limit the job to a limited number of CPUs, to reduce any negative effect of the autoindex jobs.
- If the autoindex job is not completed by a certain time, the next run will be skipped. This prevents proliferation of runaway jobs.
- The indexes created automatically are deleted after a specific number of days, which defaults to 373. But Betty can set the retention period of the unused autoindexes to, say, 100 days:

```
DBMS_AUTO_INDEX.CONFIGURE ('AUTO_INDEX_RETENTION_FOR_AUTO', '100')
```

However, the unused manually created indexes are never deleted by the automatic indexing process. They can be deleted automatically, if needed, but never by default. Betty can set another property, `AUTO_INDEX_RETENTION_FOR_MANUAL`, to specify after how many days the unused manual indexes can be dropped.

This seems to allay everyone's concern that something could go terribly out of control. In conclusion, John points out the big difference between the advisors available earlier and the Automatic Indexing feature in Oracle Database 19c. The advisors identify the need for and suggest possible indexes, but the onus of deciding whether those indexes will help or not lies with the DBAs. Automatic Indexing takes that responsibility away; it implements needed indexes automatically and constantly

checks for their usefulness. DBA tasks are limited to setting the configuration parameters such as the default tablespace or the number of days to retain the unused indexes and getting reports on Automatic Indexing activities. Everyone is impressed with this new feature, thanks John, and leaves the room a little warmer and with their usual smile back on.

Arup Nanda has been an Oracle DBA since 1993, handling all aspects of database administration, from performance tuning to security and disaster recovery. He was Oracle Magazine*'s DBA of the Year in 2003 and received an Oracle Excellence Award for Technologist of the Year in 2012.*

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

LEARN more about Automatic Indexing.

Oracle Database 19c.

TRY Oracle Autonomous Database.

BY CONNOR McDONALD

ORACLE DATABASE

A Higher-Level Perspective on SQL Tuning, Part 3

Tune poorly executing SQL statements.

In my previous article on SQL tuning, I discussed the mechanisms for identifying SQL statements that are executing poorly or, more accurately, SQL statements that have a high cost in terms of either CPU or I/O. But this identification for those SQL statements also defines "high cost" as having a negative impact on the *delivery of core business functions*. As I continue to stress in this series, the delivery of business functionality must be the driving force for tuning efforts. Although the focus of this article is how to proceed once a problematic SQL statement has been identified, I am assuming that the appropriate prerequisite tasks of liaising with the users, mapping a critical business function to this SQL statement, and validating that the SQL statement is syntactically and functionally correct to meet that business function have all been completed.

THE EXECUTION PLAN

By way of example, I will assume that the SQL statement in **Listing 1** has been identified as problematic. It provides a report to management of total employee salaries per department, based on a range of criteria from each employee's job history.

Listing 1: SQL statement that requires analysis and improvement

```
SQL> select e.department_id, sum(salary)
  2  from   employees e,
  3         job_history j
  4  where  e.employee_id = j.employee_id
  5  and    extract(year from e.hire_date) > 1985
  6  and    j.end_date > j.start_date + 1
  7  and    j.start_date >= e.hire_date
  8  group by e.department_id;

DEPARTMENT_ID  SUM(SALARY)
_____________  ___________
           50         7900
           90        17000
           30        11000
           20        13000
           80        17200
```

Let's also assume that the task of validating that the SQL meets the business requirement correctly has been completed.

Now let's look at a critical piece of information: the execution plan the optimizer derives to execute the SQL statement. Modern tools make it easy to run an `EXPLAIN PLAN` command on the SQL statement to derive the execution plan. **Listing 2** shows an example of this in Oracle's SQL*Plus, using the `SET AUTOTRACE` facility to defer running the actual SQL statement and only execute `EXPLAIN PLAN` on it.

Listing 2: Using AUTOTRACE TRACEONLY EXPLAIN in Oracle SQL*Plus

```
SQL> set autotrace traceonly explain
SQL> select e.department_id, sum(salary)
  2  from   employees e,
  3         job_history j
  4  where  e.employee_id = j.employee_id
  5  and    extract(year from e.hire_date) > 1985
  6  and    j.end_date > j.start_date + 1
  7  and    j.start_date >= e.hire_date
  8  group by e.department_id;

Execution Plan
----------------------------------------------------------

SQL_ID: 789950bkkyhcu
Plan hash value: 2697813438
```

```
-------------------------------------------------------------------------------------------
| Id | Operation                    | Name            | Rows| Bytes| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|  0|SELECT STATEMENT              |                 |    1|    39|     5  (20)| 00:00:01 |
|  1|HASH GROUP BY                 |                 |    1|    39|     5  (20)| 00:00:01 |
|  2|NESTED LOOPS                  |                 |    1|    39|     4   (0)| 00:00:01 |
|  3|NESTED LOOPS                  |                 |    5|    39|     4   (0)| 00:00:01 |
|* 4|TABLE ACCESS FULL             |EMPLOYEES        |    5|    95|     3   (0)| 00:00:01 |
|* 5|INDEX RANGE SCAN              |JHIST_EMPLOYEE_IX|    1|      |     0   (0)| 00:00:01 |
|* 6|TABLE ACCESS BY INDEX ROWID|JOB_HISTORY         |    1|    20|     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - filter(EXTRACT(YEAR FROM INTERNAL_FUNCTION("E"."HIRE_DATE"))>1985)
   5 - access("E"."EMPLOYEE_ID"="J"."EMPLOYEE_ID")
   6 - filter("J"."START_DATE">="E"."HIRE_DATE" AND
```

Unfortunately, using `EXPLAIN PLAN` for tuning is the single most common mistake made by database developers.

Even in the earliest releases of the Oracle Database cost-based optimizer, there was the potential to *not* get to the true plan for the SQL statement execution, because

- The developer is potentially running an `EXPLAIN PLAN` command in an optimizer environment not identical to the application runtime environment. (The

optimizer modes, National Language Support [NLS] settings, optimizer parameter settings, and so on may be different.)

- The data types of any bind variables can influence the execution plan. (See **"The Importance of Data Types"** sidebar for an example.)
- The values within bind variables can be examined at execution time and influence optimizer decisions, but this examination is *not done* during an `EXPLAIN PLAN` command.

As the optimizer becomes more powerful with each release of Oracle Database, many more scenarios where `EXPLAIN PLAN` does not get the true execution plan are possible, such as when previous executions of the same SQL statement are "remembered" by the database and help determine the best execution plan for future executions. A full treatment of these enhancements is beyond the scope of this SQL tuning series, but the core message here is

Do not rely on the `EXPLAIN PLAN` command result to be an indicator of the true execution plan.

This may seem like a contradiction, but this is only because, historically, many Oracle professionals have used the terms "execution plan" and "explain plan" interchangeably. They are *not* actually the same, though, and as the Oracle Database optimizer continues to be enhanced and improved, there will be more and more differences between the two.

To make the difference between the result of an `EXPLAIN PLAN` command and the actual execution plan clear, the *execution plan* is the plan that *was used* at runtime during the execution of the SQL statement. Having this execution plan is critical to

the SQL tuning process, so how does a developer get access to it? `V$SQL_PLAN` is the performance view shown in **Listing 3** that exposes the true execution plan that was used, and this can be queried for the plan of a SQL statement identified by its `SQL_ID` and `CHILD_NUMBER` values.

Listing 3: The V$SQL_PLAN performance view, holding execution plans for each SQL statement in the library cache

```
SQL> desc V$SQL_PLAN

 Name                          Null?    Type
 _____________________________ _______  __________________
 ADDRESS                                RAW(8)
 HASH_VALUE                             NUMBER
 SQL_ID                                 VARCHAR2(13)
 PLAN_HASH_VALUE                        NUMBER
 FULL_PLAN_HASH_VALUE                   NUMBER
 CHILD_ADDRESS                          RAW(8)
 CHILD_NUMBER                           NUMBER
 TIMESTAMP                              DATE
 OPERATION                              VARCHAR2(30)
 OPTIONS                                VARCHAR2(30)
 OBJECT_NODE                            VARCHAR2(40)
 OBJECT#                                NUMBER
 OBJECT_OWNER                           VARCHAR2(128)
```

OBJECT_NAME	VARCHAR2(128)
OBJECT_ALIAS	VARCHAR2(261)
OBJECT_TYPE	VARCHAR2(20)
OPTIMIZER	VARCHAR2(20)
ID	NUMBER
PARENT_ID	NUMBER
DEPTH	NUMBER
POSITION	NUMBER
SEARCH_COLUMNS	NUMBER
COST	NUMBER
CARDINALITY	NUMBER
BYTES	NUMBER
OTHER_TAG	VARCHAR2(35)
PARTITION_START	VARCHAR2(64)
PARTITION_STOP	VARCHAR2(64)
PARTITION_ID	NUMBER
OTHER	VARCHAR2(4000)
DISTRIBUTION	VARCHAR2(20)
CPU_COST	NUMBER
IO_COST	NUMBER
TEMP_SPACE	NUMBER
ACCESS_PREDICATES	VARCHAR2(4000)
FILTER_PREDICATES	VARCHAR2(4000)
PROJECTION	VARCHAR2(4000)
TIME	NUMBER

```
QBLOCK_NAME                    VARCHAR2(128)
REMARKS                        VARCHAR2(4000)
OTHER_XML                      CLOB
CON_ID                         NUMBER
```

Fortunately, there is an API to act as a wrapper around the information exposed in the `V$SQL_PLAN` performance view, which can simplify the process of getting a nicely formatted execution plan. That wrapper, `DBMS_XPLAN.DISPLAY_CURSOR`, is a table function—that is, it can be queried as if it were a database table to return the true execution plan from an execution of a SQL statement. By default, it will output the execution plan of the most recently executed SQL statement in the current database session. **Listing 4** shows an example of this (some sections of the output have been omitted for simplicity).

Listing 4: The true execution plan for the problematic SQL statement

```
SQL> select e.department_id, sum(salary)
  2  from   employees e,
  3         job_history j
  4  where  e.employee_id = j.employee_id
  5  and    extract(year from e.hire_date) > 1985
  6  and    j.end_date > j.start_date + 1
  7  and    j.start_date >= e.hire_date
  8  group by e.department_id;
```

```
DEPARTMENT_ID  SUM(SALARY)
_____________  ___________
           50         7900
           90        17000
           30        11000
           20        13000
           80        17200

SQL> select * from dbms_xplan.display_cursor();

_____________________________________________________________________________________
| Id | Operation           | Name        | Rows | Bytes | Cost (%CPU)| Time     |
_____________________________________________________________________________________
|  0 | SELECT STATEMENT    |             |      |       |     5 (100)|          |
|  1 |  HASH GROUP BY      |             |    1 |    39 |     5  (20)| 00:00:01 |
|* 2 |   HASH JOIN         |             |    1 |    39 |     4   (0)| 00:00:01 |
|* 3 |    TABLE ACCESS FULL| JOB_HISTORY |    1 |    20 |     3   (0)| 00:00:01 |
|* 4 |    TABLE ACCESS FULL| EMPLOYEES   |    1 |    19 |     1   (0)| 00:00:01 |
_____________________________________________________________________________________

Predicate Information (identified by operation id):
___________________________________________________

   2 - access("E"."EMPLOYEE_ID"="J"."EMPLOYEE_ID")
       filter("J"."START_DATE">="E"."HIRE_DATE")
   3 - filter("J"."END_DATE">INTERNAL_FUNCTION("J"."START_DATE")+1)
   4 - filter(EXTRACT(YEAR FROM INTERNAL_FUNCTION("E"."HIRE_DATE"))>1985)
```

EXECUTION PLAN NOTES

Here are a couple of additional notes about the contents of execution plans:

- Do not confuse `DBMS_XPLAN.DISPLAY_CURSOR` with `DBMS_XPLAN.DISPLAY_PLAN`, which (by default) is used to format the output of a plan saved to the `PLAN_TABLE` dictionary table. `DBMS_XPLAN.DISPLAY_PLAN` reports on optimizer plans that have been generated via the `EXPLAIN PLAN` command and can give misleading results to the developer. A call to `DBMS_XPLAN.DISPLAY_PLAN` can be modified to source its data from `V$SQL_PLAN`, but `DBMS_XPLAN.DISPLAY_CURSOR` will do the job more easily and succinctly.
- Querying a table function such as `DBMS_XPLAN.DISPLAY_CURSOR` can be done directly, as shown in **Listing 4** in Oracle Database 12.2 and later. In earlier database releases, such queries need to be written with the `TABLE` operator to let the database know that a table function is being used:

```
select * from TABLE(dbms_xplan.display_cursor());
```

```
Note
-----
   - this is an adaptive plan
```

Note that the execution plan that was used in **Listing 4** is not the same as the plan described in **Listing 2**. As previously mentioned, there could be many reasons for this, but the "Note" section in **Listing 4** gives a hint about one possible cause: Adaptive query optimization played a part in determining the execution plan. For more information on adaptive plans, consult the "About Adaptive Query Optimization" section in *SQL Tuning Guide*.

The cause of the difference between the `EXPLAIN PLAN` output in **Listing 2** and the true execution plan in **Listing 4** is less important than knowing that the true plan has now been discovered. Armed with the true plan, we can begin analysis of the problematic SQL statement for tuning.

ANALYZING THE PLAN

Since the cost-based optimizer became available, in Oracle Database 7, there have been many articles, books, and blog posts published with "best practices" for evaluating an execution plan to tune the performance of a SQL statement. Without naming and shaming sources, common advisories were

- "`TABLE ACCESS FULL` means the plan is bad."
- "High values for `COST` mean the plan is bad."
- "`MERGE JOIN CARTESIAN` means the plan is bad."

and other similar sound bites. Like many such advisories, they may occasionally be correct, but they are incorrect just as often. Worse still, these advisories remove the focus from the user experience. There is no such thing as a "good" plan or a "bad" plan—there is only a SQL statement that meets performance requirements ("good SQL") or fails to meet them ("bad SQL").

When it comes to analyzing an execution plan, rather than looking for particular keywords or patterns in the plan, I prefer a more human metaphor to understand the optimizer's role, because that will explain how to dissect the plan and get more information about it in order to convert a bad SQL statement into a good one. Each week I go to my local supermarket to get food and other necessities, but like many other people, I make an occasional midweek trip to grab a few missed items or replenish a staple such as milk or bread. On such trips, I'll have a small basket containing just a few items when I get to the checkout counters to pay and exit. At this point, I do not simply go to the closest checkout aisle. As I suspect most other people do, I'll take a quick scan of the checkout aisles to see which has the least amount of customer traffic and then I'll walk to that one. With any luck, it will either be empty or the people being served will have just a few items remaining to scan. That is just a commonsense approach to paying and getting my shopping completed as quickly as possible. With that strategy, I am being exactly like the database optimizer. There are several potential execution plans for running a SQL statement (the checkout aisles); the optimizer will estimate the effort each plan would entail (see how many people are in each checkout line); and the optimizer will decide to

use the plan that will yield the quickest result (pick the shortest checkout line).

Returning to the shopping metaphor, sometimes things don't work out quite as I expect. I'll scan the checkout aisles and see a candidate aisle that has only one person in line. I'll race down there with my small basket, confident that I've beaten the system, only to find that when I get there, that one person has the shopping cart that trumps all other shopping carts! Hundreds of items in the cart, compounded by many being fresh produce, each of which will need to be weighed and priced by the cashier. From my perspective, this is a disaster. I have just two or three things to purchase, and now I am going to be stuck waiting for 30 minutes. This too is the same difficulty that may befall the database optimizer. It may choose an execution plan that appears to be optimal (just one lone person in the checkout aisle), but when the plan is actually used to execute a SQL statement, that plan or parts of it that the optimizer estimated would be quick and efficient turn out to be more costly than expected (like the killer cart!).

This metaphor drives the strategy for the SQL tuning database developer. If the tuner/developer can locate where the optimizer estimates differed significantly from the reality of the SQL execution, that is the place to best target the tuning effort.

ESTIMATES VERSUS REALITY

Oracle Database provides an optimizer hint to provide this estimate-versus-reality information directly to the developer. Whereas most optimizer hints instruct the optimizer to use a certain operation within the execution plan, the `GATHER_PLAN_STATISTICS` hint differs, instructing the database engine to record execution plan statistics when the SQL statement is executed. Thus both the optimizer estimates and the runtime actuals are available once the execution completes. Displaying these

involves calling `DBMS_XPLAN.DISPLAY_CURSOR` with a modified format parameter, as shown in **Listing 5**.

Listing 5: Obtaining estimated versus actual statistics

```
SQL> select /*+ gather_plan_statistics */ e.department_id, sum(salary)
  2  from   employees e,
  3         job_history j
  4  where  e.employee_id = j.employee_id
  5  and    extract(year from e.hire_date) > 1985
  6  and    j.end_date > j.start_date + 1
  7  and    j.start_date >= e.hire_date
  8  group by e.department_id;

DEPARTMENT_ID  SUM(SALARY)
_____________  ___________
           50         7900
           90        17000
           30        11000
           20        13000
           80        17200

SQL> select * from table(dbms_xplan.display_cursor(format=>'ALLSTATS LAST'));
```

```
PLAN_TABLE_OUTPUT
_____________________________________________________________________________


_____________________________________________________________________________________________________
|Id|Operation         |Name       |Str|E-Rows|A-Rows|A-Time|Buffers| OMem|1Mem |Used-Mem|
_____________________________________________________________________________________________________
| 0|SELECT STATEMENT  |           |  1|      |     5|:00.01|     12|     |     |        |
| 1|HASH GROUP BY     |           |  1|     1|     5|:00.01|     12|1200K|1200K|897K (0)|
|*2|HASH JOIN         |           |  1|     1|     6|:00.01|     12|1572K|1572K|959K (0)|
|*3|TABLE ACCESS FULL|JOB_HISTORY|  1|     1|    10|:00.01|      6|     |     |        |
|*4|TABLE ACCESS FULL|EMPLOYEES  |  1|     1|   107|:00.01|      6|     |     |        |
_____________________________________________________________________________________________________

Predicate Information (identified by operation id):
___________________________________________________

   2 - access("E"."EMPLOYEE_ID"="J"."EMPLOYEE_ID")
       filter("J"."START_DATE">="E"."HIRE_DATE")
   3 - filter("J"."END_DATE">INTERNAL_FUNCTION("J"."START_DATE")+1)
   4 - filter(EXTRACT(YEAR FROM INTERNAL_FUNCTION("E"."HIRE_DATE"))>1985)

Note
_____

   - this is an adaptive plan
```

Two critical columns for the developer here are `E-Rows` and `A-Rows`. These are the rows (E)stimated by the optimizer for each line of the plan and the (A)ctual rows that

resulted at SQL statement runtime, respectively. In the plan output, line 4 of the "Predicate Information" section shows that the scan of the `EMPLOYEES` table would be filtering those rows where the `HIRE_DATE` value was more recent than 1985. The `E-Rows` and `A-Rows` information in the last line of the `SELECT` result shows that the optimizer expected to find only a single row from `EMPLOYEES` for this predicate (`E-Rows` = 1) but in fact 107 rows (`A-Rows` = 107) were found. This does not necessarily mean that the execution plan is a poor one, but it does help guide the developer to focus on the part of the SQL statement that most probably needs attention. The information leads to hypotheses the developer can explore, such as

- The optimizer statistics on the `EMPLOYEE` table might be incorrect.
- Perhaps the data on `HIRE_DATE` is skewed and requires a statistics histogram to provide the optimizer with more information.
- Maybe an index is needed on the `HIRE_DATE` column.
- Maybe the SQL predicate `extract(year from e.hire_date) > 1985` needs to be altered so that there is no expression around the `HIRE_DATE` column.

The key thing is that now the developer has a more granular area to focus on to tune the SQL.

Sometimes it is not possible to modify the candidate SQL statement to add the `GATHER_PLAN_STATISTICS` hint. In such instances, if you have access to the database session itself—for example, via a login trigger—you will be able to temporarily set the parameter `STATISTICS_LEVEL` value to `ALL` for that session and that will also collect the additional statistics.

`DBMS_XPLAN.DISPLAY_CURSOR` can then be used in a similar fashion to obtain the estimated-versus-actual SQL statement runtime comparison. Because it is unlikely in this circumstance that `DBMS_XPLAN` will be run in the same session as the SQL state-

ment, the SQL_ID can be used to interrogate the database to determine the runtime statistics. For the problematic SQL statement run earlier, **Listing 6** shows how to get the runtime statistics by locating the SQL_ID from V$SQLSTATS.

Listing 6: Obtaining estimated versus actual statistics for a known SQL_ID

```
SQL> select sql_id, sql_text from v$sqlstats
  2  where sql_text like '%extract(year%';

SQL_ID        SQL_TEXT
_____________ ____________________________________________________________
1afzpsbuadkbs select sql_id, sql_text from v$sqlstats where sql_text like '%ex
              tract(year%'
ct38j4c0rbhnj select e.department_id, sum(salary) from   employees e,       j
              ob_history j where  e.employee_id = j.employee_id and    extract
              (year from e.hire_date) > 1985 and    j.end_date > j.start_date
              + 1 and    j.start_date >= e.hire_date group by e.department_id

SQL> select *
  2  from table(dbms_xplan.display_cursor(sql_id=>'cx025dqycvcmy',
                                          format=>'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
_____________________________________________________________________________
```

```
------------------------------------------------------------------------------------------------------
|Id|Operation         |Name       |Str|E-Rows|A-Rows|A-Time|Buffers| OMem|1Mem |Used-Mem|
------------------------------------------------------------------------------------------------------
| 0|SELECT STATEMENT  |           |  1|      |     5|:00.01|     12|     |     |        |
| 1|HASH GROUP BY     |           |  1|     1|     5|:00.01|     12|1200K|1200K|897K (0)|
|*2|HASH JOIN         |           |  1|     1|     6|:00.01|     12|1572K|1572K|959K (0)|
|*3|TABLE ACCESS FULL|JOB_HISTORY|  1|     1|    10|:00.01|      6|     |     |        |
|*4|TABLE ACCESS FULL|EMPLOYEES  |  1|     1|   107|:00.01|      6|     |     |        |
------------------------------------------------------------------------------------------------------

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

   2 - access("E"."EMPLOYEE_ID"="J"."EMPLOYEE_ID")
       filter("J"."START_DATE">="E"."HIRE_DATE")
   3 - filter("J"."END_DATE">INTERNAL_FUNCTION("J"."START_DATE")+1)
   4 - filter(EXTRACT(YEAR FROM INTERNAL_FUNCTION("E"."HIRE_DATE"))>1985)

Note
-----

   - this is an adaptive plan
```

Even richer functionality is available with the Real-Time SQL Monitoring facility in the optional Oracle Tuning Pack for Oracle Database, Enterprise Edition. The same information that can be obtained from `DBMS_XPLAN.DISPLAY_CURSOR` can be extracted and presented in graphical form with Oracle Enterprise Manager or via the `DBMS_SQL_MONITOR.REPORT_SQL_MONITOR` package. For SQL statements that run for more than 10

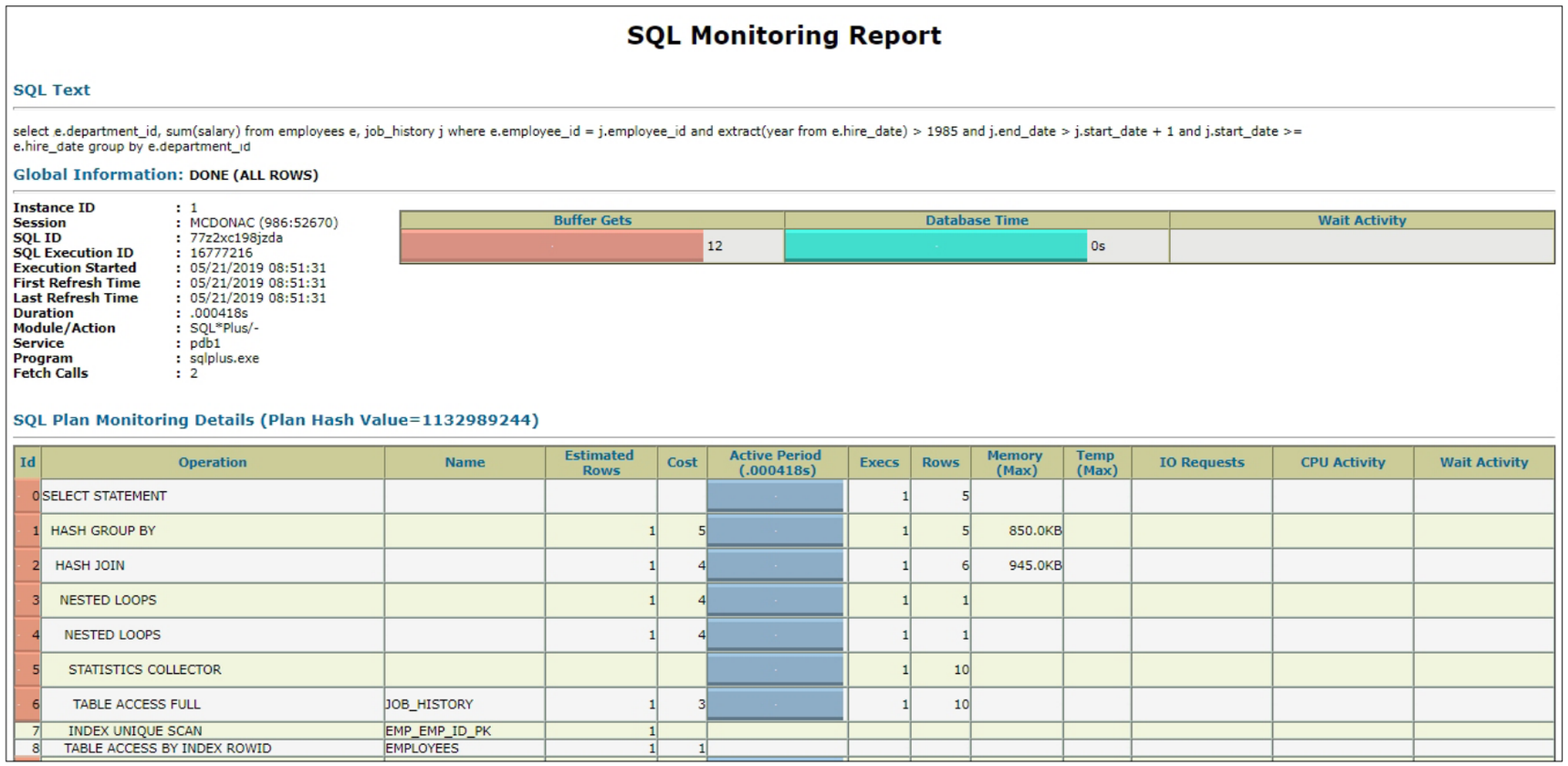
seconds, the Real-Time SQL Monitoring infrastructure automatically collects runtime statistics without the need for special hints or alterations to the `STATISTICS_LEVEL` parameter. **Listing 7** shows the reporting captured and presented via HTML with Real-Time SQL Monitoring. (The reporting output can also be spooled to a file that presents a graphical view of the runtime execution.) **Figure 1** shows the Real-Time SQL Monitoring report output for SQL_ID from **Listing 7**.

Listing 7: Real-Time SQL Monitoring report

```
SQL> select dbms_sql_monitor.report_sql_monitor
  2          (sql_id       =>'77z2xc198jzda',
  3           report_level =>'all',
  4           type         =>'HTML') report
  5  from dual;
<html>
  <head>
    <title> SQL Monitor Report </title>
    <style type="text/css">
         body, table, input, select, textarea
         {font:normal normal 8pt Verdana,Arial;text-decoration:none;
          color:#000000; empty-cells:show;}
         .s8 {font-size:8pt;color:#006699}
...
```

If you can edit the SQL text directly, you can compel any SQL statement to activate monitoring—even if it runs for less than 10 seconds—using the `MONITOR` hint.

Figure 1: Real-Time SQL Monitoring report output for SQL_ID from Listing 7



SQL Monitoring Report

SQL Text

select e.department_id, sum(salary) from employees e, job_history j where e.employee_id = j.employee_id and extract(year from e.hire_date) > 1985 and j.end_date > j.start_date + 1 and j.start_date >= e.hire_date group by e.department_id

Global Information: DONE (ALL ROWS)

Instance ID	: 1
Session	: MCDONAC (986:52670)
SQL ID	: 77z2xc198jzda
SQL Execution ID	: 16777216
Execution Started	: 05/21/2019 08:51:31
First Refresh Time	: 05/21/2019 08:51:31
Last Refresh Time	: 05/21/2019 08:51:31
Duration	: .000418s
Module/Action	: SQL*Plus/-
Service	: pdb1
Program	: sqlplus.exe
Fetch Calls	: 2

Buffer Gets	Database Time	Wait Activity
12	0s	

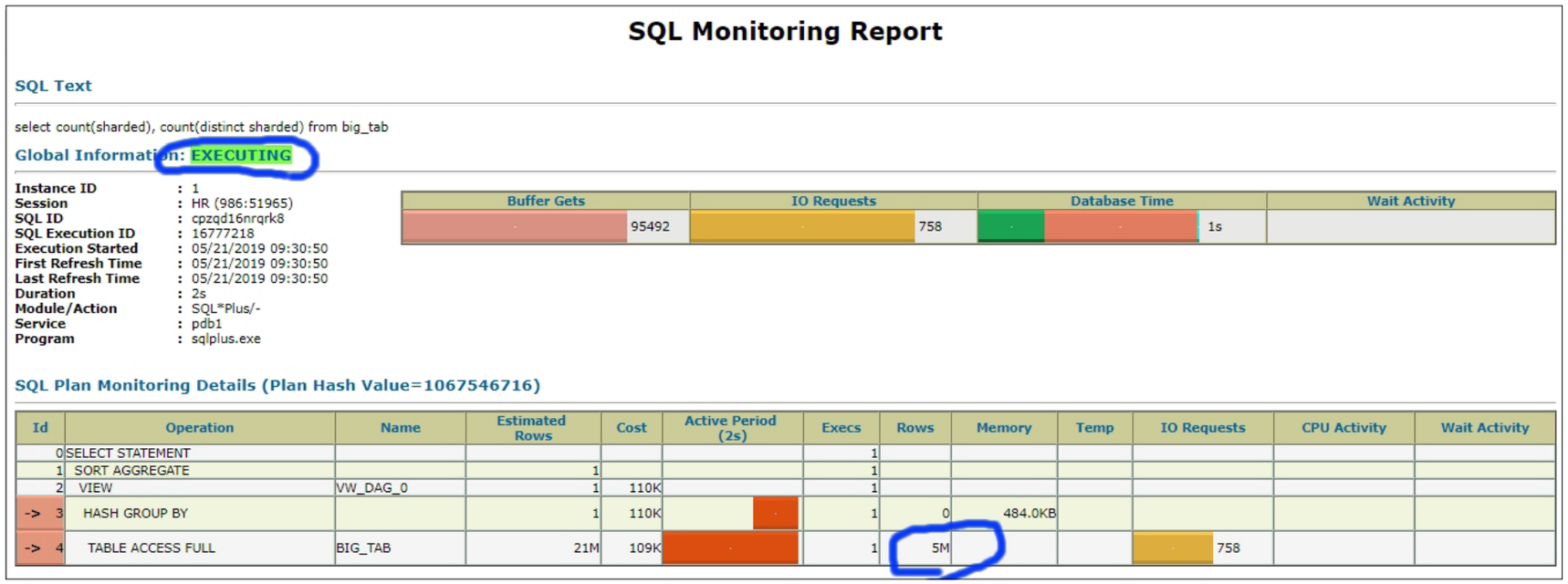
SQL Plan Monitoring Details (Plan Hash Value=1132989244)

Id	Operation	Name	Estimated Rows	Cost	Active Period (.000418s)	Execs	Rows	Memory (Max)	Temp (Max)	IO Requests	CPU Activity	Wait Activity
0	SELECT STATEMENT					1	5					
1	HASH GROUP BY		1	5		1	5	850.0KB				
2	HASH JOIN		1	4		1	6	945.0KB				
3	NESTED LOOPS		1	4		1	1					
4	NESTED LOOPS		1	4		1	1					
5	STATISTICS COLLECTOR					1	10					
6	TABLE ACCESS FULL	JOB_HISTORY	1	3		1	10					
7	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	1									
8	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	1								

The big advantage of using Real-Time SQL Monitoring over gathering runtime statistics for a SQL statement in a standard way is the “Real-Time” part of the feature’s name. Real-Time SQL Monitoring can be invoked on a *currently executing* SQL statement so that the progress of each phase of the execution plan can be monitored. **Figure 2** shows an example of a simple `COUNT` query against a 20-million-row table being monitored *as it executes*. The “Global Information” section shows that the SQL is currently executing, and the "Rows" column shows the rows processed *so far* during the execution.

For more information on Real-Time SQL Monitoring, refer to the SQL tuning documentation.

Figure 2: Real-Time SQL Monitoring report output for active SQL execution



SQL Monitoring Report

SQL Text

select count(sharded), count(distinct sharded) from big_tab

Global Information: EXECUTING

Instance ID	: 1
Session	: HR (986:51965)
SQL ID	: cpzqd16nrqrk8
SQL Execution ID	: 16777218
Execution Started	: 05/21/2019 09:30:50
First Refresh Time	: 05/21/2019 09:30:50
Last Refresh Time	: 05/21/2019 09:30:50
Duration	: 2s
Module/Action	: SQL*Plus/-
Service	: pdb1
Program	: sqlplus.exe

Buffer Gets	IO Requests	Database Time	Wait Activity
95492	758	1s	

SQL Plan Monitoring Details (Plan Hash Value=1067546716)

Id	Operation	Name	Estimated Rows	Cost	Active Period (2s)	Execs	Rows	Memory	Temp	IO Requests	CPU Activity	Wait Activity
0	SELECT STATEMENT					1						
1	SORT AGGREGATE		1			1						
2	VIEW	VW_DAG_0	1	110K		1						
-> 3	HASH GROUP BY		1	110K		1	0	484.0KB				
-> 4	TABLE ACCESS FULL	BIG_TAB	21M	109K		1	5M			758		

THE BEST PLAN

What if all the lines in the execution plan are nicely aligned in terms of estimated versus actual? This would indicate that the statistics provided to the optimizer closely aligned with the reality of the data in the tables within the query and that the plan is very likely the best plan possible for that SQL statement. That is little comfort to the database developer who is still faced with the task of tuning the SQL statement. But knowing that the plan is optimal means that the tuning focus can switch to influencing the performance of the SQL statement *outside the sphere of the optimizer*. This is the time to consider structural changes to the physical database, and the changes could include options such as

- Adding (or removing) indexes to change access methods for tables
- Compressing data to make it more efficient to scan

- Changing the table structure—for example, via partitioning
- Adding resources to the query via parallel slaves
- Making more existing resources available—for example, running the query at a different time
- Adding resources to the entire database machine

So even when the optimizer plan is optimal, there is still value in the `DBMS_XPLAN.DISPLAY_CURSOR` or Real-Time SQL Monitoring output. The other reporting columns, such as "Database Time" and "IO Requests," can help you make decisions on where structural changes will decrease the execution time the most. For example, returning to the problematic `EMPLOYEE` query, if most of the execution time is spent scanning the `JOB_HISTORY` table, there will be little possible benefit to compressing the `EMPLOYEE` table.

SUMMARY

At this point in this series on SQL tuning, problematic SQL statements have been identified and their *true* execution plans have been extracted with `DBMS_XPLAN.DISPLAY_CURSOR`. This is not to discount the value of a simple `EXPLAIN PLAN` command during the application development process as a means of getting an indicative measure of how a SQL statement may perform, but always remember that the plan observed via `EXPLAIN PLAN` is never guaranteed to be the same plan that is used at execution time when the application is released "into the wild."

Once the true execution plan has been identified by analysis of the estimated versus actual rows for each phase of the execution plan, the correctness of the optimizer decisions can be evaluated. This helps the SQL tuner narrow the focus to those elements of the plan that will likely need attention. If the optimizer was mostly

correct in its estimates, the tuning effort may need to consider structural changes such as adding indexes or altering the database design to better serve the business functional requirements.

In the next, final article in this series, I'll explore some typical issues that lead to poorly performing SQL and how to handle them.

Connor McDonald is an Oracle Developer Advocate for SQL. His passions are database design, SQL, and PL/SQL, and he can answer your database questions on Ask TOM.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

LEARN more about SQL tuning.

READ "A Higher-Level Perspective on SQL Tuning."

READ "A Higher-Level Perspective on SQL Tuning, Part 2."

DOWNLOAD Oracle Database 19c.