ORACLE MAGAZINE

NOVEMBER/DECEMBER 2018

ORACLE AUTONOMOUS DATABASE CLOUD

BREAKING FREE

With self-managing, self-securing, and self-repairing services releasing you from tedious and mundane IT tasks, what will you do next?

SHINING AN AUTONOMOUS LIGHT

SPEEDING DEVELOPMENT STRATEGIES

PIPELINED TABLE FUNCTIONS

ORACLE®

VISIT US AT ORACLE OPENWORLD
Oct. 22-25 • Booth #2619

Cloud | Hosted | On-Premise

Vertex and Oracle® Partnering for Global Tax Solutions

- Oracle® ERP Cloud
- Oracle® E-Business Suite
- Oracle® + NetSuite
- JD Edwards EnterpriseOne
- PeopleSoft Enterprise

Learn More

ORACLE Platinum Partner Cloud Standard

ORACLE Validated Integration Oracle E-Business Suite

Available on ORACLE Cloud Marketplace

ORACLE NETSUITE

VERTEX®

ORACLE MAGAZINE

30 Breaking Free

FEATURES

UP FRONT

12 Interview

CONTENTS

COMMUNITY

26 Peer-to-Peer

TECHNOLOGY

ORACLE MAGAZINE

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.

ORACLE Platinum Partner Cloud Standard

Peak Performance for your Applications

World-Class Services and Solutions

Today's systems are extremely complex running across hybrid cloud environments, consisting of many layers of technology components and business integration services. Cybernoor's deep technical expertise, holistic methodology, and world class solutions transform your business delivering maximum performance and availability for your systems and applications. Contact us to learn more about our products and solutions including Managed Services, Upgrades, Cloud Deployments, Application Development and Integrations.

To learn more visit our booth **#1908 at OpenWorld in Moscone South**.

cybernoor™

ORACLE Platinum Partner
Specialized
Oracle Exadata Database Machine

ORACLE Platinum Partner
Specialized
Service-Oriented Architecture

cybernoor.com | 925.924.0400

Tom Haunert

It’s Not a Data Problem

People may not always like what the data shows, but heroes have the data ready.

As a tech watcher, I notice references to IT in fiction—movies, television, novels, and so on. Depending on the reference, I may laugh, groan, wonder why, or—at least this once—find a reason for a thank you.

I was watching the new *Tom Clancy’s Jack Ryan* television series recently and was surprised to hear the titular character say, “…dealing with two databases that aren’t meant to talk to one another.... That’s why I’ve actually written a custom SQL query.”

“Next,” says Ryan’s new boss. And someone else starts talking.

Wait. What?

Putting aside any explanation of how this “custom SQL query” works, what is this reference doing in a meeting where staffers are introducing themselves to their new boss and describing what they’re working on? And why is the main character, not “the IT guy,” talking about IT?

Ryan’s statement about SQL followed what was supposed to be a snarky, unanswerable or, at the very least, rhetorical question. The boss asked Ryan, an analyst at the CIA, how he alone had gotten the data to support his conclusion, but the boss really didn’t want that explanation or support—because Ryan’s conclusion was that there may be a deadly new terrorist threat in the making.

ANSWERS AT THE READY

Inspired by this particular fictional scene, I tip my cap to all the real-world data experts who are providing support for unpopular or unwelcome—but accurate—conclusions. You may not hear a lot of thank yous, but you are real-life data heroes. Thank you.

Finally, I'm looking forward to seeing all data heroes and aficionados who can make it to Oracle OpenWorld and Oracle Code One 2018 in San Francisco, October 22 to 25. Stop by the Oracle Publishing booth in Moscone West to say hello!

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/STUDIO AT GETTY IMAGES FOR ORACLE**

NEXT STEPS

BUILD your OpenWorld 2018 schedule (registration required).

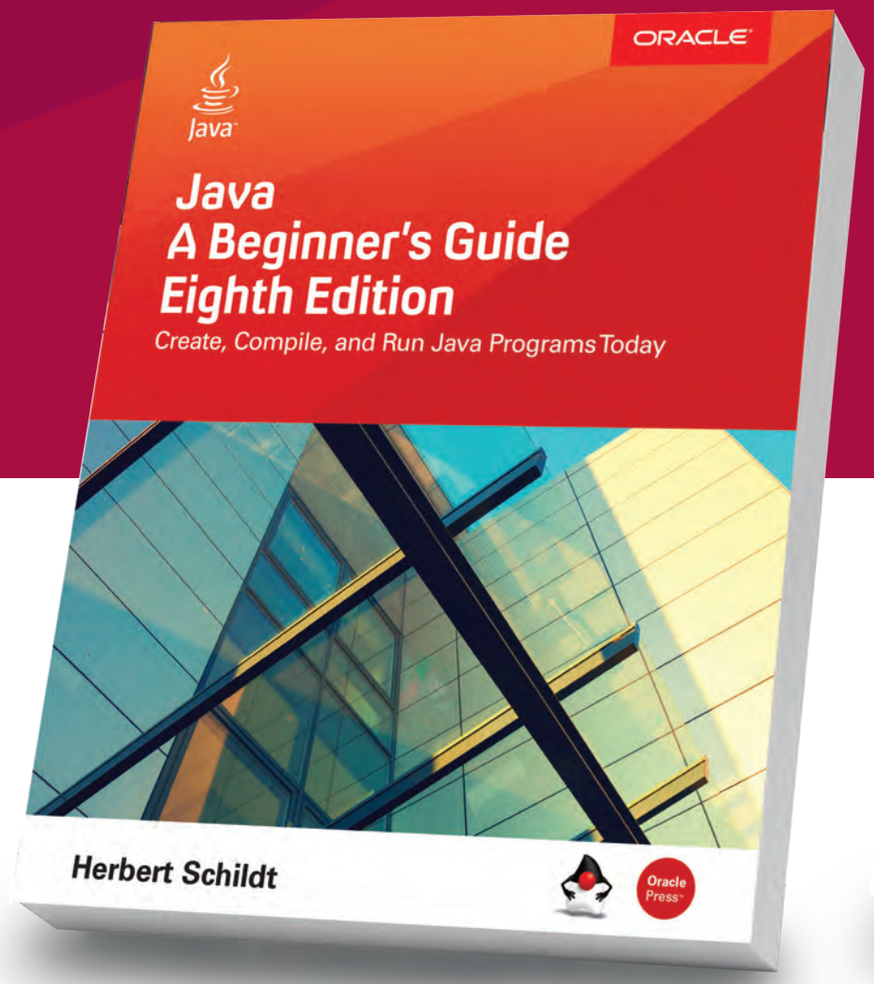
TRY Oracle Autonomous Database Cloud.

LEARN more about Oracle security.

Oracle Press™

Your Destination for Oracle and Java Expertise

Visit the Oracle Store in the Moscone West, Level 2 Lobby at Oracle OpenWorld and Oracle Code One 2018 to see our latest releases.



GET A FREE TEDDY BEAR

when you buy two or more Oracle Press or McGraw-Hill Education computing books!*

** Offer applies while supplies last*

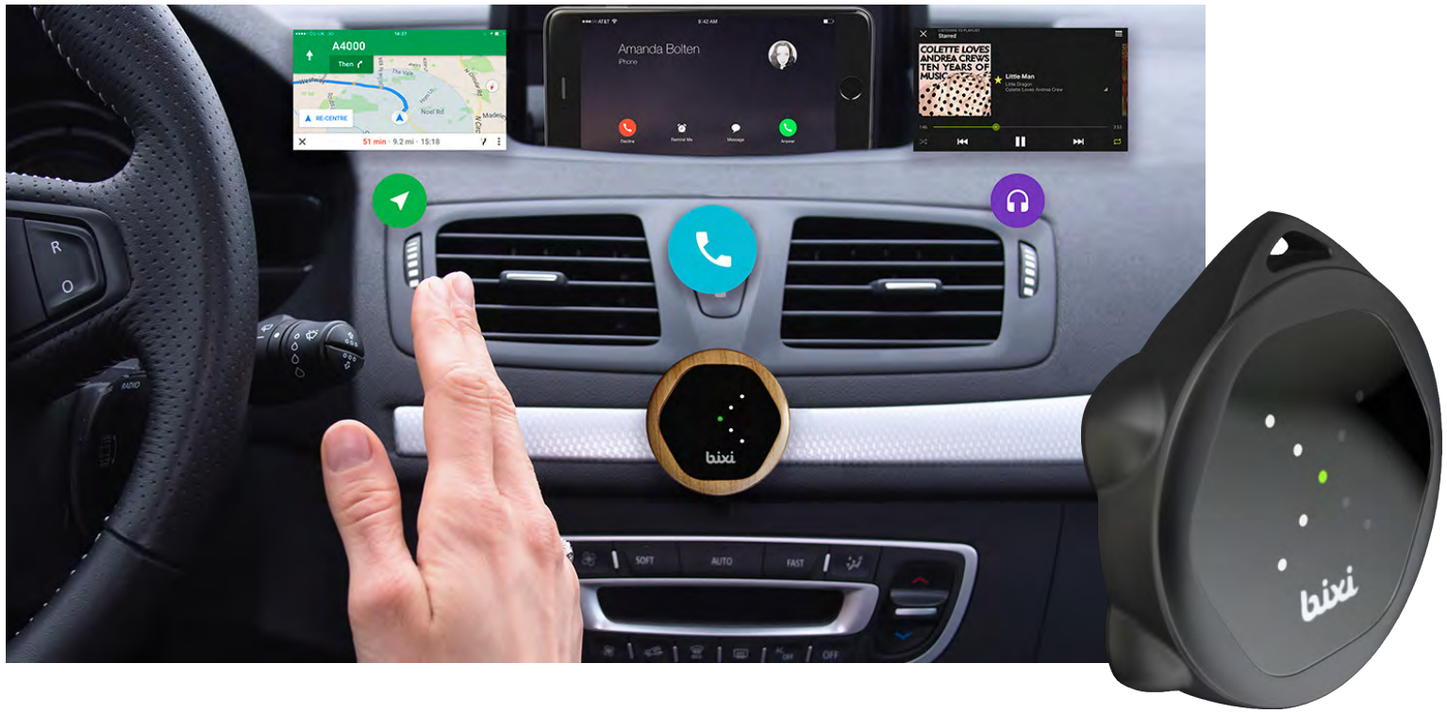
Available in print and ebook formats.

Oracle Press™

www.OraclePressBooks.com • @OraclePress

It’s a Wrap!

Presents, promises, and predictions to ring in 2019



Bixi Touch-Free Gesture Controller

Who hasn’t stealthily fiddled with a smartphone while at the wheel—to activate GPS, send a message, or control music, for example? Keep your focus on the road (or bike trail) with Bixi, a customizable device that uses machine learning to recognize in-air hand movements and deliver commands to your smartphone via Bluetooth. Attach the portable gadget to any dashboard—or use it in other situations where you need touch-free control for your mobile apps. US$99. Bluemint Labs

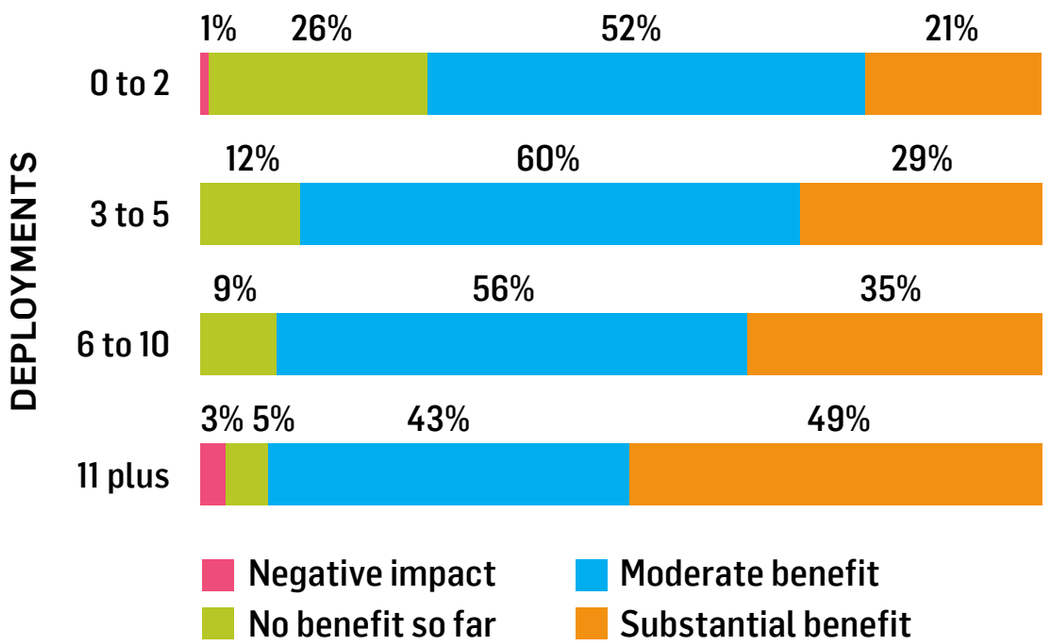
Open Sesame

Forget keys and combos: If you’re gift shopping for someone who loses both, consider the Tapplock One fingerprint padlock. This weather- and water-resistant lock opens in less than 0.8 seconds at the tap of a finger—on the lock itself or remotely from a smartphone. Programmable for 500 fingerprints, the lock boasts a capacitive fingerprint sensor with an adaptive algorithm that makes it faster and more accurate with each access. Set access rules and track history with the (free) app. US$99. Tapplock

PHOTOGRAPHS COURTESY OF BIXI AND TAPPLOCK

AI: The More You Do, the Better the Outcome

Is AI measuring up to its promises? When Deloitte surveyed 250 of "the most aggressive" early adopters of cognitive and AI technology, it found 83% of respondents' companies had already achieved either moderate or substantial benefits, integration with existing systems was a principal challenge, and most companies had either added jobs related to their cognitive technology projects or experienced "little or no" job loss. Not surprisingly, the more experience a company gained, the more economic benefit it enjoyed.



Source: "The 2017 Deloitte State of Cognitive Survey"

DO YOU SPEAK TECH? QUIZ YOURSELF!

1. Which of the following describes a generative adversarial network?

A. A network created to reproduce itself in such a way that each reproduction destroys its earlier version.

B. A euphemism high-tech executives employ to describe expanding gang networks within their industry.

C. A class of AI algorithms used in unsupervised machine learning in which two neural networks duel. One network (generative) creates a realistic photo (for example) in an attempt to fool the other network (discriminative), which attempts to determine whether that creation is real or artificial.

2. In the context of IT, CaaP stands for

A. Common language infrastructure as a platform

B. Conversation as a platform

C. City as a platform

D. Computer as a platform

3. One term for a chatbot that acts as a conversational interface for the Internet of Things is

A. CaaP

B. ThingTon

C. TalkaLoT

Answers: 1. C; 2. B; 3. B

HABIT-FORMING APPS FOR THE NEW YEAR

Three apps that help you keep those pesky resolutions

Habitica

Gamify good habits and productivity with Habitica, an app that motivates via in-game rewards (and punishments!) and connects with a large social network to help you achieve your goals. Create an avatar, customize your task list, and progress toward leveling up your character and unlocking new gear every time you tick off a task. Participate in quests alone or with friends, and stay accountable by tracking and managing your habits and achievements. Developer bonus: Habitica is an open source project. Free (Android, iOS)

Cyclemeter

Built from the ground up for mobile devices, Cyclemeter transforms your smartphone into a killer fitness computer complete with maps; graphs; splits; intervals; laps; training plans; more than 120 configurable announcements including distance, time, speed, elevation, and heart rate; and more. Sensors let you record heart rate, bike speed, bike cadence, and bike power, and you can share your location and progress as well as your workout calendar with family, friends, and coaches. Free (Android, iOS)

Ada

Looking to improve your health in 2019? Meet Ada, a personal health companion designed by a team of 100 doctors, data scientists, and engineers that uses AI and machine learning to help people to understand and manage their health. Launched with the mission to give everyone access to high-quality, personalized health information and care, Ada asks simple, relevant questions; compares your answers to thousands of similar cases to help you find possible explanations for your symptoms; and suggests next steps. Free (Android, iOS)

"Databases are good at managing data, and that's how they should be used—as data management systems," says Gerald Venzl, senior principal product manager at Oracle.

Shining an Autonomous Light

Developers see the beginning of a beautiful friendship with Oracle Autonomous Database Cloud. **BY TOM HAUNERT**

For enterprise application developers, some technologies may never be more than black boxes in a company's enterprise IT. But technology that can improve application performance does a much better job when developers know more about it and, better yet, when the technology can improve application performance on its own.

Oracle Magazine sat down with Gerald Venzl, senior principal product manager at Oracle, to talk about application development with Oracle Database, challenges database developers are facing, and how Oracle Autonomous Database Cloud offers new solutions.

Oracle Magazine: What does it mean to be an Oracle Database application developer?

Venzl: When we talk about developers doing application development with Oracle Database, we mean any kind of developer out there who is leveraging the data within Oracle Database and the features of Oracle Database to make use of the data.

This can include the front-end or web developers who start out with a focus on the UI, only consuming data via a REST call that comes out of Oracle Database, all the way to the back-end developers for whom database interactions in their code are part of their day-to-day life.

A web developer, for example, may focus only on the front-end technologies at hand and treat the database as a black box. However, that developer can and often should also learn to see what the database technology can do for that particular project and future projects.

Oracle Database has many features that can help developers accomplish their goals more quickly and easily and cut down the time to market, without having to reinvent the wheel. This leads to less application code that needs to be written and maintained and usually to better application performance overall.

Oracle Magazine: What are the barriers to developers using Oracle Database to do more within application development?

Venzl: There are certainly some misconceptions and misperceptions about databases in general.

One of the most common misconceptions I have seen is that a database should just be used as a "bit bucket." Some developers actively try to do as little as possible with the database technology, using it only as a better file system for storing and retrieving information. The issue

with that approach is that you basically use the lowest common denominator for database technology. Later on you wonder why you use a database at all, and then you try to reengineer a database inside your application logic. In the end, this just becomes a waste of your time and costs lots of money. Databases are good at managing data, and that's how they should be used—as data management systems.

"The database will do everything necessary to make sure that the application is getting the best performance."

Some developers don't use any features or functionality inside a database because they want their apps to be database agnostic. But if you do this, you cannot bring any computation to the data. Instead, you must bring the data to the application for computation—regardless of the data volume. This might be OK when you are developing an application with only a few rows of test data in the database, but it's not a scalable approach for production systems. Really, developers should put the computation and business logic where it makes the most sense for their application—the database, midtier, or front end.

Beyond misconceptions about the value of the database itself in application development, there is the issue I mentioned earlier about treating the database as a black box. Even if a developer is not treating the database as a bit bucket, a particular developer's knowledge of the database's application development power may be very limited. If you just treat a piece of technology—whether it's a framework or a database or something else—as a black box, and you think everything you need will just magically happen, things *might* work out OK. But if you don't use a technology to its full extent because you don't know what its full extent is, you're very likely going to miss out on a variety of different features and ways of making your life easier and ultimately miss out on the best application performance that you could have achieved.

Oracle Magazine: How does Oracle Autonomous Database Cloud change database development?

“Oracle Autonomous Database Cloud will be the new best friend of all the developers out there who want a data management solution that just works,” says Gerald Venzl, senior principal product manager at Oracle.

Venzl: Oracle Autonomous Database Cloud means, in a nutshell, that developers no longer need to worry about query performance, whether they need an index on a column, how much CPU the database has available, and so on.

In the 1990s, developers and DBAs had to know exactly how to write and structure a well-performing SQL statement. If they got the SQL statement wrong, it could have a heavy performance impact on their application. Over the years, Oracle Database provided more and more features allowing developers and DBAs to not have to worry about these things anymore. A cost-based query optimizer took away

the need to know how to structure a query, which tables to join first, and so on. Then came the SQL Tuning Advisor feature, telling DBAs how to tune Oracle Database for the best SQL performance. Now, with Oracle Autonomous Database Cloud, the database doesn't even tell developers and DBAs what to do anymore—it simply performs all of these steps itself.

With Oracle Autonomous Database Cloud, developers can now fully focus on the applications they're writing and the value their applications should provide to their companies or users. They no longer have to spend time thinking about whether an index has been created on a particular column, whether the table has been partitioned, or whether the database back end will perform. The database will do everything necessary to make sure that the application is getting the best performance and quickest response time. If a SQL query needs tuning, the database will do it. If a column needs an index, the database will create it. If the data needs to be reorganized on disk or in memory, the database will do it. If you give it more CPU, it will scale up automatically, taking advantage of the added processing power. It's all fully transparent to the application, and there's no downtime required whatsoever.

Oracle Autonomous Database Cloud will be the new best friend of all the developers out there who want a data management solution that just works.

PHOTOGRAPHY BY **BOB ADLER/STUDIO AT GETTY IMAGES FOR ORACLE**

NEXT STEPS

DOWNLOAD Oracle Database application development drivers and tools.

LEARN more about Oracle Autonomous Database Cloud.

TRY Oracle Autonomous Database Cloud.

25,000+ Companies Run Their Business in the Oracle Cloud

More Enterprise Cloud Applications Than Anyone Else

ORACLE®

From Analog Potatoes to a Coding Career

Oracle Developer Champion Loiane Groner shapes the real world.

By **Bob Rhubart**

As a teenager, when Loiane Groner selected computer science as a course of university study, she wasn't thinking about computer programming or coding. "I thought that I was going to learn how to fix computers," she explains.

"In Brazil you have to choose your major before you go to college," Groner says. "I chose computer science because I liked computers. That was the only reason."

Groner was 10 years old when she got her first computer. But with no access to the internet, she used the computer to play games and to install trial versions of software that came on discs in magazines. Then she took a course in Windows

Loiane Groner liked computers, chose computer science, and now she shares her expertise in books, blogs, trainings, and more.

RECOGNIZE

The Oracle Developer Champion program recognizes modern expert developers who blog; write articles; and present on topics such as containers, microservices, SQL, NoSQL, open source technologies, machine learning, and chatbots. Learn more and follow the Oracle Developer Champions.

and DOS. “Software was kind of like magic,” she says.

That started to change in college. Her first class was an introduction to algorithms. Her professor explained algorithms using the preparation of potatoes as an analog. “I remember this as if it were today,” Groner says. “He explained the whole process—get the potatoes from the refrigerator, wash them, don’t peel them, boil them, and all that.”

At first, Groner had some issues understanding the programming logic. “But I’m a little bit stubborn,” she says. “Whenever I try to do something and I’m not able to do it, I will try until I’m able to do it. So I studied a lot after school.”

Her introduction to Java came through a study group organized with the help of one of her professors. For six months, Groner and the others in the study group focused on Java basics.

At that time, Groner lived in Vitoria, a small town in Brazil, where she became part of the local Java community. “I thought that was really cool, and it was also my first contact with a community beyond my college colleagues.”

While still at university, Groner got her first job in 2006 and stayed for two years, an experience she describes as her first contact with the real world of programming. “I didn’t know about version control. I didn’t know how to do a SQL query or even a SELECT from the database.”

Her first assignment at that job was to develop a simple Java application. “It read XML files from the FTP, parsed the XML, and saved the data to the database.”

Her second full-time job allowed her to expand her skill set to include JavaScript and front-end development, all while still attending university. Upon graduating from university, Groner was offered a job at IBM in São Paolo, where her work with JavaScript expanded.

After two years with IBM, Groner joined Citibank in 2011 as a senior developer analyst. In 2015, she became a vice president business analyst, still active in development. Now in her seventh year

with Citibank, Groner moved to Florida early in 2018.

Throughout her career, Groner has been actively involved in sharing her expertise, earning her status as an Oracle Developer Champion through a variety of activities.

Since 2012, she has written eight books for Packt Publishing. The latest, *Learning JavaScript Data Structures and Algorithms,* Third Edition, was published in April 2018.

In addition to writing her books, Groner is a prolific technical blogger, writing in English and her native Portuguese. She also produces an extensive series of tutorial videos on YouTube, which are focused on Java, JavaScript, Angular, Cordova, and other technologies, all in Portuguese.

Groner also regularly speaks at a wide variety of industry events, including QCon, DevFest, Oracle Code São Paolo, and Oracle Code One.

So although Groner didn't know what to expect when she showed up for that first computer science class, she has packed an incredible level of experience into a decade of coding, which should give you a very good idea of what to expect from her.

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

PHOTOGRAPHY BY **JOCK FISTICK/STUDIO AT GETTY IMAGES FOR ORACLE MAGAZINE**

NEXT STEPS

LEARN more about Oracle Developer Champions.

FOLLOW Loiane Groner's blog.

By **Alexandra Weber Morales**

Don’t Just React; Have a Plan

Oracle Developer Champion Rolando Carrasco’s top productivity tips

Today’s application landscape abounds with REST-based web services and APIs that enable you to share a program’s functionality with ease. That in itself is a productivity boost—but have you thought about how to make the process of creating APIs themselves more productive? That’s a question to which Mexico City–based Oracle Developer Champion Rolando Carrasco has devoted some thought, because a good API can be a gateway to all kinds of success. So how does he quickly create and validate service-oriented web hooks?

“One of the most useful things for my work nowadays is Oracle Apiary technology,” says Carrasco. The technology

Oracle Developer Champion Rolando Carrasco chooses plans, music, sleep, mornings, and more on the road to productivity.

"If you want to create good code and be productive, you must have a plan and a good design—something you can target."

—Rolando Carrasco, Oracle Developer Champion

includes a powerful command-line interface (CLI) he can customize to programmatically run his API creation process. "After I design and model APIs, I use some of its CLI utilities to test and validate their design. Because those are CLI utilities, they can be automated and facilitate the work once the APIs are mocked up and passed to the developers for both consumption and customization."

As a Java developer, Carrasco has been using Oracle JDeveloper since 2002—and has stayed faithful to the tool as both it and his Java work have expanded. "It was a very preliminary version, but because that was my first contact with an IDE, I continued using it. Then, because of all my work with Oracle Fusion Middleware, I kept using it." For microservices development, he uses IntelliJ IDEA, and Atom is his preferred Python code editor. "Because I am a Windows guy, I use Babun as my shell, and I use Gradle to build my applications."

RESPECT FOR SLEEP

Structuring his day according to his energy levels, Carrasco dedicates the first 60 to 90 minutes to studying a new technical area. After his morning studies he jumps into development, aiming to get most of his work done in the morning and avoiding going too late into the afternoon. "I'd rather be at top speed at the beginning of the day than at the end. I normally try to avoid leaving things for the next day. If there is something I can conclude today, I try to have the discipline to do it."

He also believes in the value of rest. "I have a lot of respect for my sleep, so I try not to get back home too late. In Mexico it's very normal to have long workdays, but I try to finish my tasks before 7 p.m. and then get back home," says Carrasco. At home he still aims to read or study something new, such as his current focus: learning to speak Japanese.

FINDING FLOW

Carrasco's favorite shortcut to finding flow when coding? "Music. No doubt about it. I like many genres, from Big Band to '90s alternative music. So if I really want to get into a good flow, a set of good music from artists such as Todd Terje, Erasure, Tim Maia, the Supremes, Pet Shop Boys, or the Ramones will definitely do the job," he says.

Like any good citizen of a major metropolitan area, Carrasco views his commute time not as wasted but, rather, as an opportunity to learn more. "I am not a gadget guy, but what I do use is the Amazon Kindle—both the device and the app—to read books when I am moving from one place to another," he says. Whether he's in a cab, on a bus, or riding the subway, he absorbs technical articles, either reading them or listening via the Kindle's ability to read the book aloud.

Ultimately, Carrasco's success as an Oracle Developer Champion boils down to his ability to consistently contribute to the community, stay abreast of new information, and optimize the different segments of his day for the activities that keep his career moving forward. "Always have a plan," he advises. "If you start your day with no plans, if you rely on your reactions and pretend that having no plans in your life will let you achieve goals, you are probably wrong. The same thing applies to programming: if you want to create good code and be productive, you must have a plan and a good design—something you can target."

Alexandra Weber Morales is Oracle director of developer content.

PHOTOGRAPHY BY **MARTIN VARGAS/STUDIO AT GETTY IMAGES FOR ORACLE**

NEXT STEPS

FOLLOW the Oracle Developers blog.

LEARN more about Oracle Developer Champions.

Make Your Web App Look and Act Like a Mobile App

An Oracle APEX developer presents the progressive web app approach.

By **Jeff Erickson**

Just about any mobile phone user would rather click an icon to open an app full-screen than type a long URL in a browser. So web developer and Oracle Application Express (Oracle APEX) expert Vincent Morneau went looking for a way for his web apps to deliver something closer to that native mobile app experience.

He found it in a relatively new mobile publishing approach: using progressive web apps (PWAs). PWAs are applications built by web developers using web technologies but that offer a look and feel similar to native mobile apps.

Morneau has been exploring and evangelizing this approach ever since: speaking at user group events and posting to his blog and GitHub account.

"People prefer using native apps on their phone," for many reasons, he says. "A home screen icon just makes the app more accessible, it works when it's offline, and it can also push notifications out to users." Now, Morneau says, web applications can do all these things with PWA technologies.

I caught Morneau's PWA session at ODTUG Kscope18 in Orlando, Florida, where Morneau showed how to use

Oracle APEX in Oracle Database, along with JavaScript and a small set of modern browser APIs, to build a PWA.

PWAs open up a lot of possibilities for developers, "because we're not just building web apps anymore," he says. "We're simply building apps, which are no longer different from native mobile apps."

Like native mobile apps, PWAs also support offline operations. "A native app can still work offline by offering partial functionality" and will synchronize pending requests with the server when the connectivity comes back, Morneau says.

There are a ton of mobile use cases where a stable internet connection just isn't a reality. "In my research, I came across an Oracle APEX application that needed to be usable on off-grid farms," he says. Because it was delivered as a PWA, farmers could enter measurements directly into the web application, which was automatically synchronized with the server when connectivity came back. Another common use case is the subway, "where you might lose the signal for a minute or two but you're not interrupted," he says.

In the end, PWAs fit the way Morneau likes to develop Oracle APEX apps. "I've always been a web developer. I love the openness of the web and the freedom it gives," he says. "I don't like having to learn multiple languages to publish the same app on iOS and Android…and I love the fact that PWAs can be released seamlessly via the web and don't require system updates."

Jeff Erickson is editor at large for Oracle Content Central.

NEXT STEPS

WATCH Morneau's Oracle APEX–to-PWA full walk-through.

FOLLOW Morneau's PWA project on GitHub.

Only Connect

Three peers extol the virtues of a good conversation.

Gianni Ceresa
Lausanne, Switzerland

ORACLE ACE

Company/URL: DATAlysis GmbH
Job title: Managing director
Length of time using Oracle products: 10 years

How are you using cloud computing in your work these days? I think of cloud as a sandbox in which I can easily play with almost any kind of tool. With just a few clicks, I can provision an Oracle Analytics Cloud instance or a database service instance, and in minutes I can test tools and solutions. But for cultural reasons and some regulations limits here in Switzerland, "foreign" hosted clouds are not a well-accepted solution, so I really look forward to the plans Oracle has for regional expansion.

What's the most common cause you see when IT projects go wrong? The main cause is a lack of vision and clear requirements. This happens every time a project is set up to "just" replace something without first asking the users what they really need. Because of that tendency, there's also often a mismatch between the planned resources and the skills and experience of team members that the project truly requires.

What's the next big thing driving change in your industry? There are three: AI; machine learning; and automation of repetitive tasks, improving reaction time and information usage. The risk with all three is that they will become "buzzwords" that everybody talks about but nobody really masters—so I wouldn't mind for the industry to adopt a small-steps approach.

Roy Salazar Valverde
San Jose, Costa Rica

ORACLE ACE Associate

Company/URL: Pythian
Job title: Oracle Database consultant
Length of time using Oracle products: 20 years

How did you get started in IT? While at university, I chose my courses carefully, with an eye to getting employed before graduating. I was still a student when I got the chance to work as a Y2K consultant in 1998. It was really hard to work all day and then study at night and on weekends, but it was important to me to both work and learn as much as possible at university. After my Y2K consulting experience, I continued working as a developer and then became a technical consultant and systems analyst.

Which new features in Oracle Database are you currently finding most valuable? I've checked out the Oracle Database 18c new features for security, and I'm very intrigued by the possibility in this release of having a schema that is just that—a schema—rather than both a schema and a user, as in the past. Nowadays, when security is more and more important, this feature is very relevant to force appropriate settings for accessing the data in the database. And I truly love the Oracle Database Security Assessment Tool, which was recently improved to help companies address GDPR compliance. I've been using this tool since the beginning of this year and have found it very useful for security audits and reviews.

How are you using social media in your work these days? I was really resistant to getting into social networks, but some years ago I decided to join Facebook and LinkedIn. It was only in November of last year, when I was accepted into the Oracle ACE Program, that I created a Twitter account. Now I love having interactions with tech "gurus" in the social networks where I'm active.

Yasuo Honda
Tokyo, Japan

ORACLE ACE

Company/URL: freee K.K.
Job title: Software developer
Oracle credentials: Oracle Certified Professional, MySQL 5.6 Database Administrator
Length of time using Oracle products: 22 years

What technology has most changed your life? I would say Oracle Database and Ruby on Rails. Since I started working as an Oracle consultant, Oracle Database has given me a great view of IT. I have gained a deeper understanding of everything from storage, networking, and operating systems to application-level performance tuning. When I decided I wanted to learn a programming language, I discovered Ruby on Rails, one of the best web application frameworks ever written in the Ruby language. I now attend RailsConf every year, where many Rails developers give inspiring talks and you can have great conversations about solving technical problems.

What's your favorite tool on the job? Soon after discovering and becoming a fan of Ruby on Rails, I learned about ActiveRecord, an open source adapter that provides Oracle Database access from Ruby on Rails applications. With this tool, I've been able to make use of my Oracle Database knowledge, and it's given me a lot of new and interesting programming challenges. I'm now one of the maintainers of the project.

How are you using social media in your work these days? I communicate with other Ruby developers, otherwise known as Rubyists. I'm most active on GitHub. Once I open pull requests regarding Ruby on Rails projects, I get great feedback from Rails committers. There are many Rails committers and contributors who frequently open pull requests and fix issues. Even if most of this communication is done online, when I meet them in person at RailsConf or other meetups, we have lots to talk about.

World's First "Self-Driving" Database

Oracle Autonomous Database

No Human Labor
Half the Cost

No Human Error
100x More Reliable

Human labor refers to tuning, patching, updating, and maintenance of database.

ORACLE®

BREAKING FREE

Oracle Autonomous Database Cloud puts database administration on autopilot.

BY DAVID BAUM

Today's AI systems bring a high degree of automation to everyday activities, from selecting the fastest driving routes to getting to know your music preferences. The most-advanced AI systems use machine learning algorithms to analyze current conditions and learn from experience. As these self-aware systems get more capable, they are transforming the IT industry as well—including Oracle Database, the world's most popular enterprise data management system.

"With Oracle Autonomous Database Cloud, all management tasks are fully automated, including all database tuning chores," says Çetin Özbütün, Oracle senior vice president. "Your data is automatically compressed and encrypted, and your queries and transactions run fast, because our autonomous database service is built on Oracle Exadata and Oracle Database, which represent 10,000 person-years of engineering effort."

Oracle Autonomous Database Cloud currently includes two services: Oracle Autonomous Data Warehouse Cloud is designed for data warehouse and analytics workloads, while Oracle Autonomous Transaction Processing Cloud enables in-memory OLTP access.

Fueling Administrative Efficiency

Like all other Oracle Cloud services, Oracle Autonomous Database Cloud services scale to fit your capacity requirements and are delivered via a pay-per-use model—which Özbütün says can cut runtime costs by as much as 90%. These capabilities are especially attractive to organizations such as Drop Tank that want to eliminate mundane database management tasks by standardizing on a self-driving, self-securing, self-repairing database.

Drop Tank, a leading loyalty technology and rewards company, is using Oracle Autonomous Data Warehouse Cloud to create new loyalty solutions for gas station opera-

DROP TANK
Burr Ridge, Illinois

INDUSTRY:
Loyalty technology and awards

EMPLOYEES:
25

REVENUE:
Undisclosed

ORACLE PRODUCTS:
Oracle Autonomous Data Warehouse Cloud
Oracle Autonomous Database Cloud
Oracle SOA Cloud Service

"We're seeing a future that involves a lot more transactions, a lot more data, and a lot more need to make use of that data," says David VanWiggeren, CEO of Drop Tank (left), shown here with Drop Tank Vice President of Technology Tim Miller.

tors. The company partners with loyalty programs to help them add fuel purchases to their overall member experience and then works with service stations and their attached convenience stores to present compelling offers to consumers.

In addition to creating effective loyalty solutions, Drop Tank mines insights from point-of-sale data to help gas station operators and consumer packaged goods (CPG) companies identify trends and make intelligent decisions about future promotions and sales strategies. The data helps these partner companies understand and influence consumer purchase behavior at thousands of service stations and convenience stores. Oracle Cloud services, anchored by Oracle Autonomous Data Warehouse Cloud, make it possible.

"By collecting point-of-sale data, we can discover trends and figure out what offers are working, and where," says Tim Miller, vice president of technology at Drop Tank. "More and more, the data is

> **“Oracle Autonomous Data Warehouse Cloud allows us to focus on analyzing the data rather than on managing systems. We don’t want to have to deal with management, patching, and security—and we can’t afford to have our systems down for maintenance.”**
>
> —**Tim Miller**, Vice President of Technology, Drop Tank

driving our marketing and promotional calendars. We use analytics to get the right offers to the right people at the right time.”

Previously, Drop Tank used on-premises databases and management software for these tasks, but as the company grew, maintaining this complex IT environment became progressively more difficult. “We had to administer the virtual machines, patch the software, manage the application servers, and tune the databases, which meant we either had to hire system administrators for each technology component or become experts ourselves,” Miller recalls. “We’re not a large company, and as we grow, we don’t want to have to hire a team of database administrators to manage databases, update software, create tables and indexes—all of the things DBAs historically do. Oracle Autonomous Data Warehouse Cloud allows us to focus on analyzing the data rather than on managing systems. We don’t want to have to deal with management, patching, and security—and we can’t afford to have our systems down for maintenance.”

CPG companies typically sell through distributors that place goods at retail stores, but the consumer experience is a “black hole” because the point-of-sale data rarely reaches the manufacturer. Many of these firms don’t even know how much their products sell for. Drop Tank can capture the point-of-sale data and share it with the pertinent companies to unlock these customer insights. Everybody in this interlinked value chain wins: Gas stations can boost

sales and drive more repeat business through greater customer loyalty, CPG companies can produce more of the items that sell well and yield the best margins, and consumers are happy to find their favorite products on the shelves.

"We're seeing a future that involves a lot more transactions, a lot more data, and a lot more need to make use of that data in order for us to be efficient," reports David VanWiggeren, CEO of Drop Tank. "Universal product codes, quantities, prices, taxes—all of those offers and items can now be related to individual consumers."

According to VanWiggeren, service stations love Drop Tank's loyalty programs because they "drive incremental purchases"—and not just gallons of gasoline, but convenience store items as well. Drop Tank's largest retail network includes Marathon Petroleum, whose gasoline is sold through approximately 5,600 independently owned retail outlets across 20 states and the District of Columbia. Given the size of the network, other national loyalty programs are now using Drop Tank to engage with their members each time members fill up or make purchases in the convenience store.

The Value of an Integrated Cloud Platform

Drop Tank has found it easy to extend the capabilities of Oracle Autonomous Data Warehouse Cloud with other complementary Oracle Cloud services. For example, Drop Tank uses Oracle SOA Cloud Service to gather point-of-sale data and upload it into Oracle Autonomous Data Warehouse Cloud, where it is tied together to drive analytic insights. In the future, Drop Tank plans to use Oracle Analytics Cloud to analyze and visualize the data—both at the store level to track sales and in aggregate to help its partners understand short- and long-term purchasing trends.

Drop Tank also plans to use Oracle Autonomous Transaction Processing Cloud to execute a growing volume of point-of-sale transactions—from approximately 5,000 transactions per day now to an estimated 100,000 transactions per day by the end of 2019. "We anticipate a huge increase in memberships and transactions," Miller says. "Oracle lets us automatically scale its cloud services to handle increasing volume. It's easy to add capacity during big promotional campaigns and then scale back down again. If we aren't using it, we aren't paying for it."

Although the premise of "pay only for what you use" has become an anthem for the cloud computing industry, Miller says that, in practice, not all cloud services are easy to scale. "With some vendors, you have to shut everything down and rebuild your information systems," he explains. "You can't just say, I need another node. With Oracle Cloud, on the other hand, we can spin up new nodes and within five minutes everything will be ready to go."

Unlimited Scalability and Exceptional Performance

Having instant scalability is equally important to Aker BP, one of the largest independent oil companies in Europe, with full-fledged exploration and production operations concentrated on the Norwegian continental shelf. Previously, if Aker BP needed a new data warehouse, it could take weeks to provision the infrastructure, install the software, create the database tables, load the data, and tune the system for the workload at hand. Now, using Oracle Autonomous Data Warehouse Cloud, the company can implement a new data warehouse in minutes—and the database tunes itself to ensure maximum performance.

"The whole process of using Oracle Autonomous Database Cloud is very easy, from setting up an account to creating an instance to loading data," says Erik Dvergsnes, senior architect at Aker BP.

According to Dvergsnes, creating a new data warehouse with Oracle Autonomous Database Cloud required four simple steps: naming the database, specifying the number of CPUs, determining how much storage was needed, and entering a password. After that, he was able to connect the new cloud database with Oracle SQL Developer and immediately start running queries. "I copied some Data Pump .dmp files to the cloud and ran some Data Pump imports on those files," he explains. "The process is clearly documented on Oracle's

AKER BP
Fornebu, Norway

INDUSTRY:
Oil and gas

EMPLOYEES:
1,300

REVENUE:
US$2.6 billion in 2017

ORACLE PRODUCTS:
Oracle Autonomous Data Warehouse Cloud
Oracle Autonomous Database Cloud
Oracle Exadata

"My focus is to migrate more and more databases to the cloud," says Aker BP Senior Architect Erik Dvergsnes, whose company is looking into taking advantage of the Oracle Bring Your Own License program.

website. I could immediately start querying and playing with the data."

Dvergsnes says the performance of Oracle Autonomous Data Warehouse Cloud is exceptional because of the way it automatically handles storage cells and indexes. As a result, Aker BP reduced report runtimes from 20 minutes to less than 1 minute for a 1 TB data warehouse with 1.2 billion rows, without any manual tuning by the IT staff. "It's like having the world's best DBA working for you," he adds.

In addition, thanks to the Hybrid Columnar Compression feature of Oracle Database, Aker BP was able to reduce its data storage requirements by more than 70%. For example, one database went from 900 GB to 240 GB, once it was compressed.

Partly as a result of these performance and efficiency gains with Oracle Autonomous Database Cloud, Aker BP is now looking into decommissioning one of its Oracle Exadata systems and

> **“It’s like having the world’s best DBA working for you.”**
>
> —**Erik Dvergsnes**, Senior Architect, Aker BP

transferring the associated database licenses to Oracle Cloud as part of the Oracle Bring Your Own License program. “I’m really excited after testing Oracle Autonomous Database Cloud,” Dvergsnes states. “My focus is to migrate more and more databases to the cloud, with the goal of switching off our Oracle Exadata by the end of 2019.”

A Self-Scaling Database That Eliminates Downtime

Aker BP runs a 24/7 oil production and drilling operation, which means its information systems for issuing work orders and monitoring projects can’t tolerate downtime. These round-the-clock operations have always made it difficult to schedule database maintenance, especially since work schedules on the rigs can be interrupted at any time due to rough weather in the North Sea. The self-patching capabilities of Oracle Autonomous Data Warehouse Cloud could alleviate the need to ever shut down this important database for maintenance. “Many of our systems are mission-critical, so planning downtime for software patches and security updates is extremely difficult,” Dvergsnes says.

While the self-patching capabilities will eliminate downtime for essential reporting activities, having capacity on demand will reduce the need to maintain excessive capacity to accommodate peak analytic loads. For example, Aker BP must periodically generate reports about the status of its offshore equipment, which involves investigating a constant barrage of alarms and alerts. The IT team sometimes has to review months of alert data to trace issues with a particular piece of gear, such as an oil pump. Which systems were involved? What alarms were generated? Who followed up, and what actions were taken?

“We might have to submit a query on a specific tag on a specific time stamp to find out exactly what types of alarms were generated from a specific incident,” Dvergsnes explains. “Due to legal reasons, we need to be able to always query this data.”

These report requests are sporadic—sometimes daily, other times monthly—and the associated queries can consume massive compute

and storage resources. "It's a perfect candidate for Oracle Autonomous Data Warehouse Cloud, because we can turn off the database when it's not in use and then scale it back up again when we need it," Dvergsnes adds. "It's critical to produce these reports, both for the government and for our partners. If the reports are delayed, Aker BP has to pay big fines. If a deadline is looming, we can simply scale up to add CPUs and get the reports done in time. We can spin down a CPU to zero if nobody is using it. Oracle Autonomous Database Cloud is outstanding in the market, as far as I can see."

Enterprise-Grade Technology for Smaller Firms

One of the driving forces behind Oracle's new autonomous services is to enable small and midsize organizations to benefit from enterprise-grade technology that was formerly available only to large companies with many IT professionals on staff. Quality Metrics Partners (QMP), a healthcare holdings company specializing in ancillary service management, is a case in point. QMP used Oracle Autonomous Data Warehouse Cloud to create a cloud-based healthcare technology platform called CAREiQ that enables the company to improve its internal processes and offer cloud-based diagnostics, billing analysis, and other lab services to healthcare providers. Rather than allotting months to creating an on-premises analytics platform, QMP's clients can typically implement CAREiQ within a week. Once they're on the system, they can get diagnostic screening results within hours—a cycle that formerly took days.

"Perhaps the most important outcome of our new cloud service is better care for the

QUALITY METRICS PARTNERS
Dallas, Texas

INDUSTRY:
Healthcare IT

REVENUE:
US$60 million in 2017

EMPLOYEES:
200

ORACLE PRODUCTS:
Oracle Autonomous Data Warehouse Cloud
Oracle Autonomous Database Cloud
Oracle Analytics Cloud
Oracle Database Cloud Service
Oracle Data Integration Cloud
Oracle Data Integrator Cloud Service

> **We simply create a new instance in the cloud, and we can load a new client's data immediately. Performance and scalability tuning take place automatically behind the scenes. If you throw a bigger workload at it, the system accommodates it for you.**
>
> —**Michael Morales**, CEO, Quality Metrics Partners

patient population," says QMP CEO Michael Morales. "Oracle tools enable us to educate physicians and help them be more aware. They can discover patterns in the data that assist with their diagnosis and detect potentially overlooked disease markers. They can also receive alerts, so they can be proactive about conditions that they may not have caught otherwise."

QMP began as an owner of diagnostics laboratories and now delivers data analytics and other services to hospitals, clinics, and laboratories. Its analytics systems pull client data from various sources, including multiple types of electronic medical records systems, laboratory information systems, and billing systems. It uses Oracle Data Integrator Cloud Service to gather the information and prepare the data. Oracle Data Integrator Cloud Service reformats transactions, combines data from multiple tables, and then loads the data into Oracle Autonomous Data Warehouse Cloud.

Having a fully managed data warehouse service helps QMP eliminate manual, error-prone management processes. "We don't need to have a DBA constantly managing the system," says Steve Chamberlin, CIO of QMP. "We can grab data from another database, another cloud service, or lab files and then load it directly into the warehouse. Oracle establishes the architecture and manages the process. It's actually more efficient than what our DBAs constructed."

The new platform enabled QMP to avoid hiring a projected 15 to 20 additional analysts

> “Perhaps the most important outcome of our new cloud service is better care for the patient population,” says Quality Metrics Partners CEO Michael Morales.

per laboratory to manage its growing data, saving the company hundreds of thousands of dollars in employee costs. Routine management tasks such as provisioning, patching, performing backups, tuning, and applying security patches have been automated by Oracle’s autonomous cloud service. “We help other healthcare organizations leverage their data so they can provide better service to their patients and prosper in this competitive industry,” Morales explains. “We can tap into the database of a clinic or hospital and pull data into Oracle Autonomous Data Warehouse Cloud. It loads and scales immediately to the required size.”

Built-in adaptive machine learning technology automatically tunes, upgrades, and patches the database while it’s in operation, even as workloads increase and decrease. “We simply create a new instance in the cloud, and we can load a new client’s data immediately,” Morales continues. “Performance and scalability tuning take place

automatically behind the scenes. If you throw a bigger workload at it, the system accommodates it for you."

According to Chamberlin, Oracle's new database also helps QMP comply with the Health Insurance Portability and Accountability Act (HIPAA) regulations governing the security and privacy of patient data. "Oracle provides a secure environment that uses encryption technology and other advanced database security technologies to help us with the HIPAA regulations," he adds. "In addition, the Oracle data centers are audited every quarter. That's way more often than what most small companies can afford to do."

Looking ahead, Morales hopes to use Oracle Autonomous Transaction Processing Cloud as well, saying the new OLTP service will represent a "big win" for everyone involved: QMP can improve operational efficiency and expand its business model; doctors and clinicians can make better, more-knowledgeable decisions; and patients will get test results and diagnoses more quickly than ever before.

"We help doctors make intelligent decisions and focus on what they're good at," Morales concludes. "Autonomous technology from Oracle enables us to provide them with simple tools that can help them run their practices better so they can more effectively manage the entire patient population under their care."

David Baum is a freelance business writer specializing in science and technology.

ILLUSTRATION BY **PEDRO MURTEIRA**
PHOTOGRAPHY BY **PAUL S. HOWELL, TON HENDRIKS**

NEXT STEPS

LEARN more about Oracle Autonomous Database Cloud.

TRY Oracle Autonomous Database Cloud.

SPEEDING DEVELOPMENT STRATEGIES

Oracle Autonomous Database Cloud offers more choices and easier development.

BY DAVID A. KELLY

Twenty years ago, IT organizations generally included two types of developers: back-end database developers and front-end application developers. Today's developers, however, work beyond those limitations to create applications and solutions that call on data, logic, and processes that might reside on back-end databases or datasources, midtier application servers, microservices, and multiple and varied front-end targets.

What's the key to being a modern developer, building both back-end and front-end logic and interfaces? Do you need to master SQL, PL/SQL, database tuning, Java, *and* JavaScript? In short, no. The key to being a modern developer isn't mastery of any one specific tool or language. Instead, it's a learning mindset. Developers must always be pushing to learn more—more about new technologies; more about new tools; and, perhaps most important of all, more from other developers and developer communities. (See "Oracle Developer Advocates Team" for more information on developer and community support.)

And while PL/SQL continues to be a powerful language for modern Oracle Database application development, a lot of new database application development is being driven by new API and interface requirements and by developers with a wide range of backgrounds and technology skills, from PHP and Python experts to developers using REST APIs with JavaScript. And there's good news for developers who may have been slowed by database tuning issues in the past: With database technology advances such as Oracle Autonomous Database Cloud, being a database application development expert doesn't require knowing how to tune or administer the internals of relational databases.

Finding the Right Everything

Application developers have more languages, drivers, frameworks, APIs, and services to choose from now than ever before. Although that's great for flexibility and optimization, it can be somewhat overwhelming for less-experienced or new developers. How do you know what the right approach is for your needs?

As an enterprise-class solution that supports open standards, Oracle Database provides drivers for every popular language, including Node.js, PHP, Python, .NET, C, C++, PHP, Go, Argo, and more. Members of the open source developer community provide additional lan-

> **“Developers want to create something. They’re excited when they dig in and build a solution. But I think it’s also important for them to not reinvent functionality just because it’s fun to build.”**
>
> —**Gerald Venzl,** Senior Principal Product Manager, Oracle

guage support by contributing code on places such as GitHub, and they share their knowledge and code examples on their blogs, forums, and community portals.

Data access to an Oracle Database instance can also be provided via standard RESTful web services that can be consumed by any language. With Oracle REST Data Services, a developer can write SQL and publish that SQL as a REST API. Oracle REST Data Services will automatically map input values in REST GET or POST operations, bind them to the SQL statement, and return the results in JSON.

“REST is the most common way you’re going to do something, unless there’s some complex reason you can’t do it that way,” says Mike Hichwa, vice president of software development at Oracle. “It’s becoming the lingua franca.”

Oracle Database development tools, all free to use, include Oracle SQL Developer; Oracle SQL Developer Data Modeler; Oracle SQLcl; and Oracle Application Express, which enables developers who are comfortable with SQL to create modern, responsive web applications very quickly.

“If you want a tool for application development or data access, Oracle has you covered,” says Hichwa. “We have the industry’s leading data modeler, the most capable REST-to-relational database tool, drivers for all the most popular languages, and higher-level tools that can provide a visual or 4GL experience.” (The Oracle Database Application Development page includes links to language support and drivers for Node.js, Python, PHP, .NET, Java, C, C++, and more and Oracle Database development tools.)

Oracle Developer Advocates Team

Oracle Database powers tens of thousands of applications around the world. The Oracle Developer Advocates team is on a mission to help developers make full use of Oracle Database to build high-performance, ultrasecure, and easily maintained applications. This small but busy team publishes blogs and videos, shares tips and how-tos on Twitter and other social media platforms, and much more. Oracle's Chris Saxon and Connor McDonald have taken over the famous Ask TOM site after Tom Kyte's retirement, and they have raised it to new heights of activity. Ask TOM now offers monthly Office Hours sessions with dozens of Oracle Database experts. The team also publishes quizzes, workouts, and classes on Oracle Dev Gym, a unique "active learning" site that uses quizzes to reinforce knowledge gleaned from videos and tutorials.

"Everyone knows how powerful Oracle Database is, but not enough developers realize how much they can benefit by fully leveraging all the development features of the database," says Steven Feuerstein, a longtime PL/SQL expert and team lead. "The Oracle Developer Advocates are doing everything we can to get that word out and help developers be more successful."

The Cloud and Autonomous Services

Many Oracle Database application developers already have used Oracle Cloud services to create solutions faster and with fewer resources. But for even-more-rapid development and deployment cycles, Oracle Autonomous Database Cloud is the next big advance. Oracle Autonomous Database Cloud uses machine learning to automatically maintain and tune itself, eliminating manual tuning, errors, and labor.

"Oracle Autonomous Database Cloud will basically provide faster time to market because developers won't have to worry about back-end details. They can simply build their application and rely on Oracle Autonomous Database Cloud to handle the workload and retune the system as needed," says Gerald Venzl, senior principal product manager for application development at Oracle.

This can be a real advantage for developers when they scale their solutions from the development or pilot stage to production. Instead of having to invest resources and time to take a solution live on a larger scale, developers can rely on Oracle Autonomous Database Cloud to automatically take care of the scaling aspect.

Recognize

The Oracle Developer Champion program recognizes modern expert developers who blog, write articles, and present on topics such as containers, microservices, SQL, NoSQL, open source technologies, machine learning, and chatbots. Learn more and follow the Oracle Developer Champions.

"Autonomous is the next step, where you have a database service and don't have to care about how it's set up and what you have to do to keep it running optimally," says Venzl. "I think it's an exciting time for developers."

Delivering Innovation Faster

The future has never looked brighter for database application development with a great combination of modern standards such as REST and JSON and with drivers, services, and tools that address all mainstream development languages and needs. Perhaps now, with so many great solutions, the challenge for developers is not just to get the job done, but to once in a while take a step back and ask themselves if they're doing the job in the most efficient way.

"I believe that the best developers are builders," says Venzl. "Developers want to create something. They're excited when they dig in and build a solution. But I think it's also important for them to not reinvent functionality just because it's fun to build. Instead, developers need to take a look at what's out there and see what they can leverage—whether it's functionality in Oracle Database or frameworks and resources available through developer communities—so they don't reinvent the wheel over and over again. If developers do that, they'll be able to drive innovation even faster."

David A. Kelly is an established children's book author, travel writer, and technology analyst.

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

DOWNLOAD Oracle Database application development drivers and tools.

LEARN more about Oracle Autonomous Database Cloud.

TRY Oracle Autonomous Database Cloud.

BY FRANK NIMPHIUS

ORACLE MOBILE CLOUD ENTERPRISE | ORACLE INTELLIGENT BOTS

Adding Remote Data Access to Bot Conversations

How to build custom components for use in Oracle Intelligent Bots

My personal definition for chatbots is that they are front-end intelligent systems that use artificial intelligence and machine learning to understand what a user wants and to gather all information required to complete a task. For this, the bot needs to engage in a dialog with the user, for which Oracle Intelligent Bots uses built-in components to render bot responses and to request user input.

Where there is a front-end intelligent system, there must also be a back-end system that the bot calls to complete a task after the user intent is understood and all information is gathered or to query data for display. In Oracle Intelligent Bots, this remote data access is implemented through custom components you build with JavaScript and Node.js and that you declaratively add to a state in the bot dialog flow.

In this article, you will learn everything you need to know about custom components in Oracle Intelligent Bots. In the hands-on instructions, you are going to

extend the pizza bot I explained how to build in the course of my three most recent *Oracle Magazine* articles, with a dynamic menu that reads its contents from a custom component.

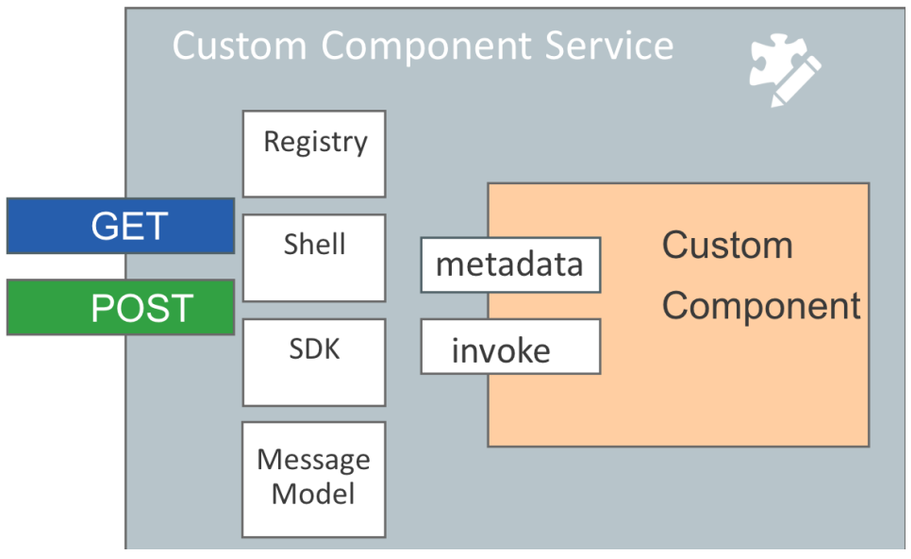
Understanding how to build custom components is a must-have skill for Oracle Intelligent Bots bot designers implementing custom business and data logic. So let's get started.

ABOUT BOT CUSTOM COMPONENTS

As **Figure 1** illustrates, a custom component service in Oracle Intelligent Bots is a Node.js-based REST service that hosts one or many custom components and that exposes two HTTP methods: GET and POST.

The GET method is invoked at design time by the bot when the custom component is registered with the Oracle Intelligent Bots instance. The POST method is invoked at runtime when the dialog flow navigates to a state that is associated with

Figure 1: Bot custom component service architecture



a custom component hosted by the component REST service.

When invoked, the custom component REST service receives the name of the custom component to invoke; looks up the custom component implementation JavaScript module; and dispatches the request, along with a reference to the custom component SDK.

A custom component is a Node module built with JavaScript that exposes two functions:

metadata. This function returns information about the custom component upon custom component service registration in the bot. The information includes the component name, the definition of the component input parameters (properties), and the action strings returned by the component to enable the bot designer to decide how to continue the dialog flow.

invoke. This function is called at runtime and executes the custom component business logic. The logic may access dependent resources such as Oracle Mobile Cloud connectors, platform services, and custom APIs.

The custom component SDK consists of four files and provides functions for reading the incoming bot message and sending a component response:

Registry. The registry.js file contains a mapping of custom component names and their physical JavaScript files.

Shell. The shell.js file contains the dispatcher logic that reads file locations from the registry.js file and loads the requested custom component implementation.

SDK. The sdk.js file is passed as a reference to the custom component's invoke function.

MessageModel. The MessageModel.js file contains helper functions for the Oracle Intelligent Bots conversation message model and helps component developers build rich UI bot responses.

HANDS-ON PREREQUISITES

You develop custom components in JavaScript with Node.js. For this you need Node.js and the node package manager (npm) to have been installed on your local machine.

To test whether you have Node.js installed, open a command-line window and type the `node -v` command. If you have Node.js installed, a version number should appear. Try the same with the `npm -v` command to verify the installation of the node package manager (npm).

Note: You can download Node.js and npm from npmjs.com/get-npm.

A second prerequisite for building custom components is the Oracle Intelligent Bots custom component SDK. You can download the latest bot custom component SDK from Oracle Technology Network (OTN).

Note: For this article, the bot custom component SDK files are provided within the download for the hands-on instructions.

A last prerequisite is a JavaScript editor, which can be as simple as a text editor—commercial or free—such as Atom or Microsoft Visual Studio Code.

GETTING STARTED

To follow the hands-on instructions in this article, you need to have access to Oracle Mobile Cloud Enterprise, which is available as a free trial.

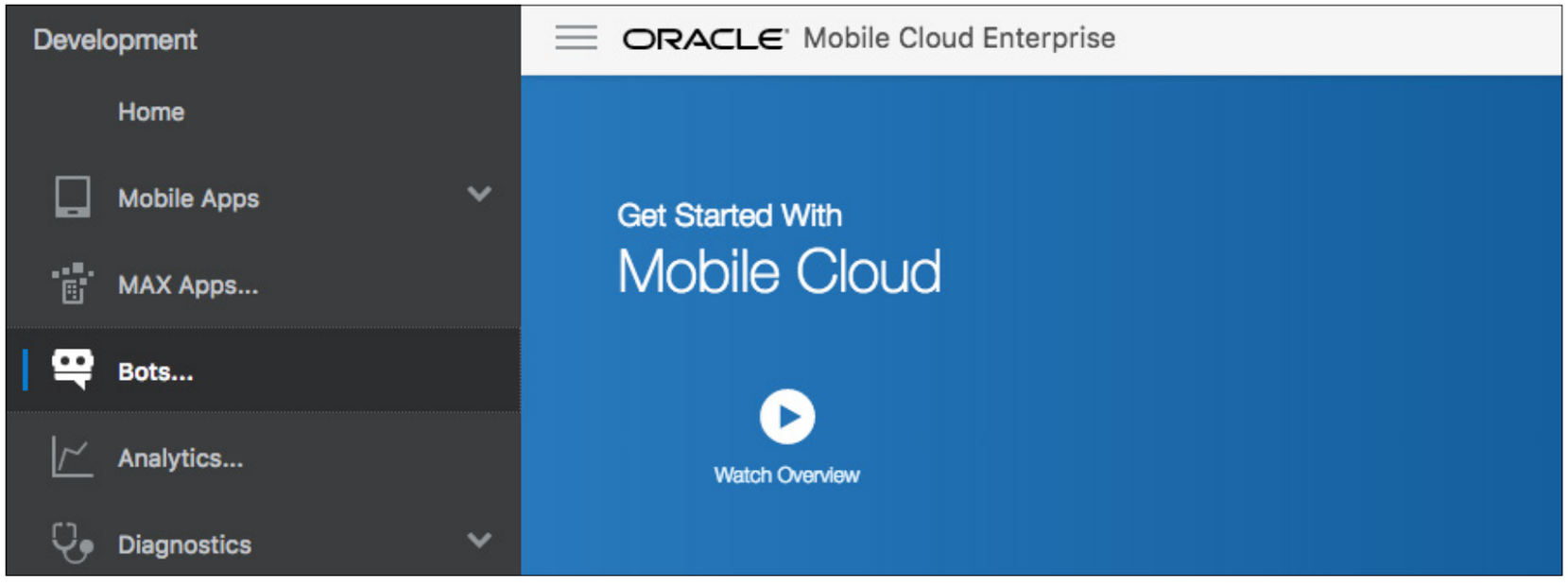
The download for this article contains, in the starter-bot folder, a bot you need to import to your Oracle Intelligent Bots instance to complete the hands-on steps for this article. To import the bot, after downloading and unzipping the resources file for this article, do the following:

1. Open a browser, and access the Oracle Mobile Cloud Enterprise homepage.

2. Authenticate with the user credentials you defined when you provisioned the cloud service.
3. Click the hamburger icon at the upper left to open the Oracle Mobile Cloud Enterprise menu.
4. Click the **Bots** menu item (shown in **Figure 2**), which opens the Oracle Intelligent Bots dashboard in a separate browser window or tab.
5. Click **Import Bot** at the upper right of the dashboard.
6. In the opened dialog box, navigate to the location to which you extracted the downloaded zip file and open the **starter-bot** folder.
7. Select the **OracleMagazineOnlinePizzaBot4.zip** import file, and click **Open**.

Note: You cannot import two bots with the same name. If you need to import the starter bot more than once or if you share an Oracle Intelligent Bots instance, rename any previous imports of the starter bot before importing the new bot. You

Figure 2: Oracle Mobile Cloud Enterprise with Oracle Intelligent Bots selected



can rename a bot by opening it, clicking the **Settings** icon (), renaming the bot, and exiting the bot.

8. Close the upload confirmation dialog box, by clicking the **X** at the right side.
9. Keep the browser window open.

TRAIN AND TEST THE SAMPLE PIZZA BOT

The hands-on instructions in this article continue using the pizza bot created in my last *Oracle Magazine* article. Note, however, that for this article, I did remove the dependency on translation services to make it easier to set up the bot, which means that this bot works only for English. To train and test the pizza bot, follow these instructions:

10. Open the browser window or tab with the Oracle Intelligent Bots dashboard in it.
11. If you already have existing bots defined in your environment, chances are good that you won't immediately spot the downloaded bot. To find the bot, type `OracleMagazine` into the **Filter** field above the green **New Bot +** icon.
12. Click the **OracleMagazineOnlinePizzaBot4** bot to open it.
13. Click **Train** (at the top right). Before you click it, **Train** should show an exclamation point, because the bot is not yet trained after the download and import. After you click **Train**, a dialog box appears that confirms that the bot has been trained. Training the model is required for the bot to understand free-form text input.
14. Run the embedded bot tester, by clicking the **Test** icon () at the upper right.
15. Type `Hi` into the **Message** field, and click **Send**.
16. Type `Show me the menu` into the **Message** field, and click **Send**.
17. Choose a pizza, and complete the order process.
18. Click the **Reset** button at the top right of the embedded bot tester.
19. Try the bot again, but this time type `I like to order a pizza`, and click **Send**.

20. Choose a pizza, and complete the order process.
21. Click **Reset**.

WHAT YOU ARE GOING TO BUILD

The sample pizza bot has a section defined in the dialog flow that writes the pizza menu card to a context variable. To see the menu definition, in the bot, select the **Flows** icon in the left toolbar ( ) and explore the `InitializeMenu` state. As you can see, all items on the pizza menu are defined in this state.

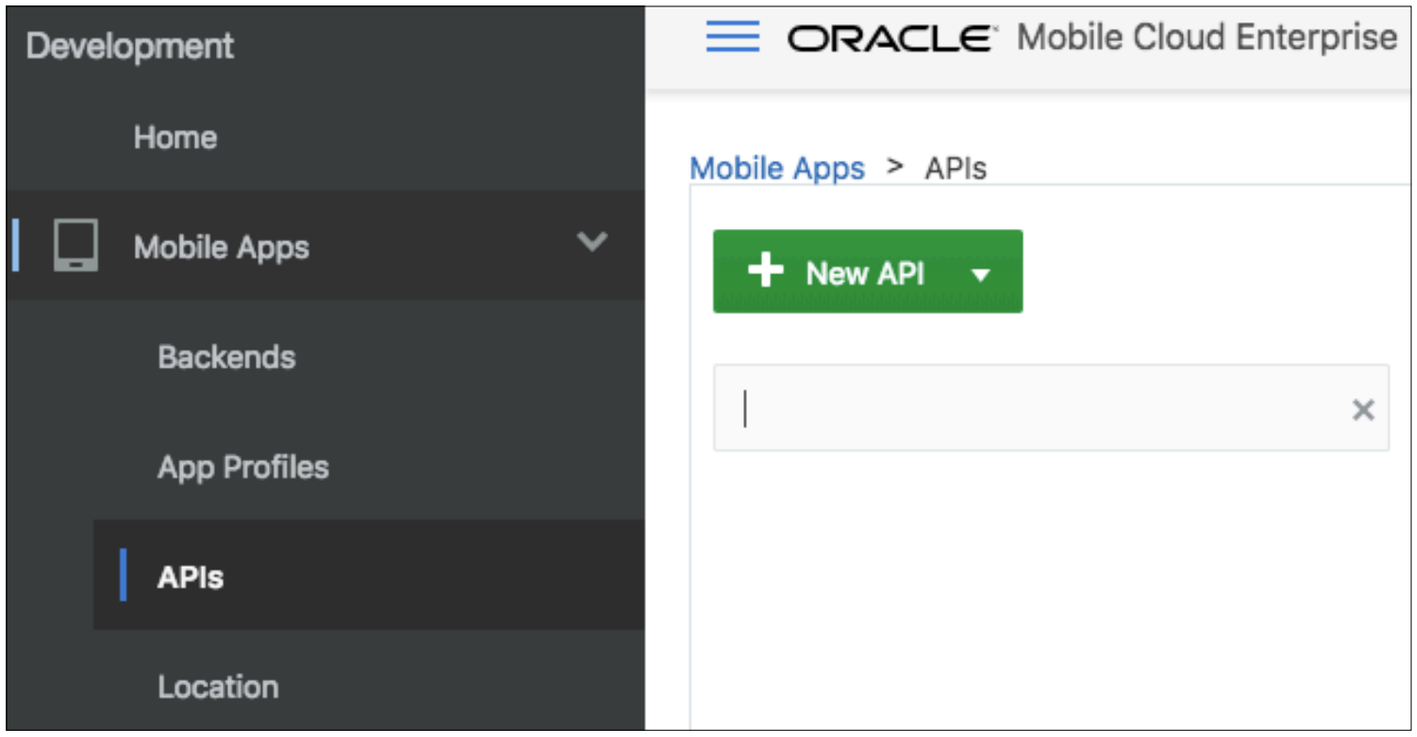
Because it is very unrealistic for a pizza bot to have the pizza menu defined in the dialog flow, you are going to replace the menu definition with a custom component that would have the option to *query* the pizza menu contents from, for example, a remote web service.

CREATING A NEW CUSTOM COMPONENT SERVICE IN ORACLE MOBILE CLOUD

Custom component services in Oracle Mobile Cloud are built as custom APIs. Let's create a new custom component.

22. In Oracle Mobile Cloud Enterprise, select the **APIs** menu item from the **Mobile Apps** menu category, as shown in **Figure 3**. To see this option, you may have to click the hamburger icon next to the Oracle Mobile Cloud Enterprise label.
23. Click the **+ New API** button, and choose **API** from the menu.
24. In the New API dialog box, click the **Upload a RAML** document link.
25. In the file system, locate the directory where you unzipped the download for this article, open the **raml-template** folder, and select the **mcebots.raml** file.
26. Click **Open**.
27. Change the **API Display Name** field value from **MceBots** to `MenuService`.
28. In the **Short Description** field, add `Oracle Magazine pizza bot menu`.

Figure 3: Creating a custom API menu item



29. Click **Create**.
30. Select the **Security** (🔒) icon from the left menu, and toggle the **Login Required** button off (so that it's greyed out).
31. Click the **Save** button at the top right.
32. Select the **Implementation** (</>) menu option at the left side.
33. On the implementation page, click the green **JavaScript Scaffold** button to download the Node.js starter project (zip file) for building the custom component service.
34. Keep the browser window open.

What you just did: An Oracle Mobile Cloud custom API is a Node.js REST Express project. By using the mcebots.raml template to create the custom API, you created a

custom component service base project with the GET and POST method signatures preconfigured. By disabling the login requirement for the custom API, you allowed anonymous user access, which will be used by the bot when registering the component. And finally, you downloaded the project so that, as explained in the next section, you can configure the component service with the custom component SDK and build the custom component.

PREPARING THE COMPONENT SERVICE PROJECT STRUCTURE

You downloaded the Oracle Mobile Cloud custom API project, so next you need to turn this into a custom component service project by adding and configuring the custom component SDK.

35. Navigate to the folder to which you downloaded the Oracle Mobile Cloud custom API (in step 33).

36. Find and unzip the downloaded **menuservice_1.0.zip** file.

It's my personal preference to organize the component service project with custom component SDK files located in their own folder and the custom components created in subfolders. You may come up with your own location strategy at some point, but for now, let's use mine.

37. Open the newly created **menuservice** folder, and create a new folder named **js** therein.

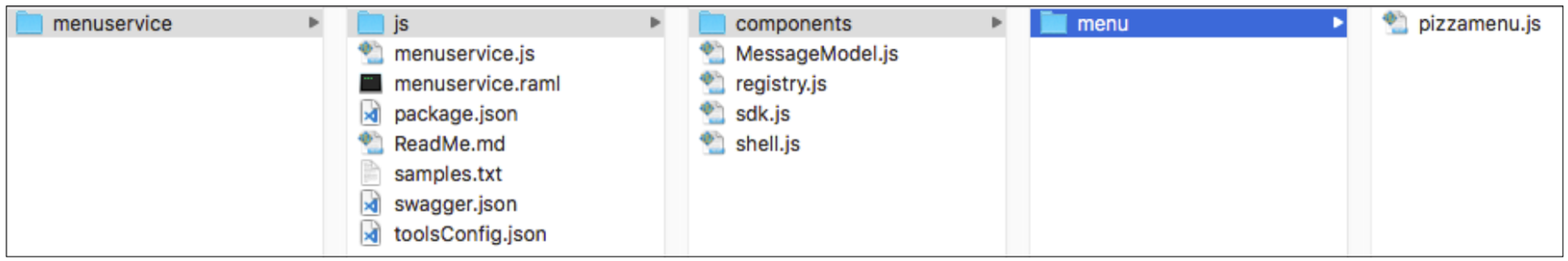
38. Open the **js** folder, and create a folder named **components** therein.

39. Open the **components** folder, and create a folder named **menu** therein.

40. Copy the **shell.js**, **registry.js**, **MessageModel.js**, and **sdk.js** files, located in the **bot-component-sdk** folder of the download for this article, and paste them into the **js** subfolder in the **menuservice** folder.

41. Copy the **pizzamenu.js** file, located in the **menu-component** folder in the download for this article, and paste it into the **menu** subfolder in the **menuservice** -> **js** -> **components** folder. Your overall project structure should look like what's shown in **Figure 4**.

Figure 4: Custom component service project folder structure



What you did: You created a custom component service project structure and copied the bot custom component SDK provided in the download for this article into it. You then copied the **pizzamenu.js** file, a predefined custom component that holds the pizza menu for the sample bot, into the menu folder.

EDITING THE COMPONENT SERVICE PROJECT

Now it is time for you to open your favorite JavaScript editor and edit some JavaScript. I chose Atom.

42. In Atom, select **File** -> **Add Project Folder**, select the **menuservice** folder, and click **Open**.
43. Open the **menuservice.js** file, shown in the project menu at the left side.
44. Take note of the REST URI /mobile/custom/MenuService/components, configured for service.post and service.get.

45. Replace all of the **menuservice.js** file contents with the contents of the **mcebots.js** file, located in the component-service-src folder in the download for this article.

46. Find the **const apiURL** variable declaration, and change it to

```
const apiURL = '/mobile/custom/MenuService/components';
```

47. Next, find the code line

```
var shell = require('./shell')();
```

and edit it to match the project structure created earlier:

```
var shell = require('./js/shell')();
```

48. Save the changes.

What you just did: The custom component service code doesn't differ much from one service to another. The only difference is in the REST endpoint URI. The mcebots.js file you copied the contents from is a generic template provided by Oracle in the samples on OTN that I included in the download for this article. And because the SDK files are saved in the component service js folder, you updated the reference in the value for the shell variable.

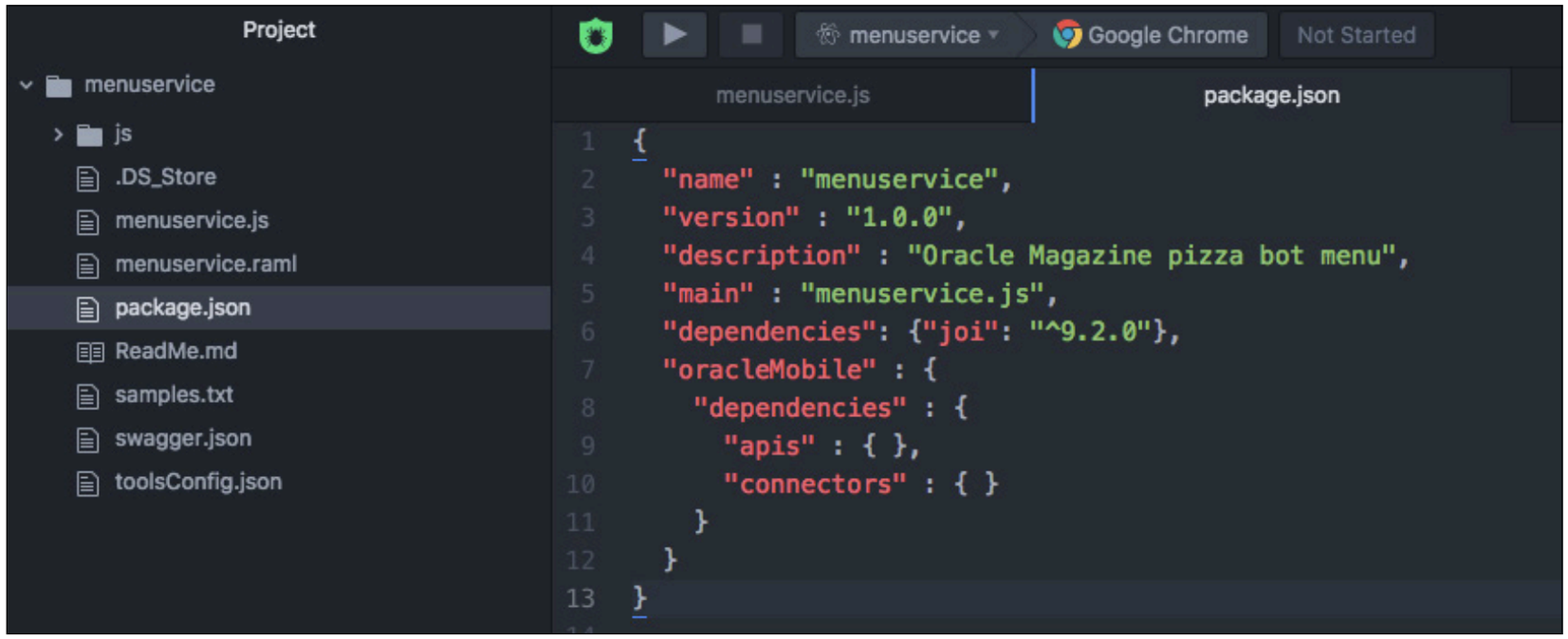
49. Open the **package.json** file, located in the project menu on the left side.

50. Create a new line *after* `"main" : "menuservice.js",` and add the following:

```
"dependencies": {"joi": "^9.2.0"},
```

The file contents should now look like they do in **Figure 5**.

Figure 5: package.json file after editing



51. Save the file.
52. Expand the **js** folder under the **menuservice** project root folder.
53. Open the **registry.js** file for editing.
54. Change the following line in the file from

```
'say_hello': require('./examples/say_hello')
```

to

```
'oramag.sample.pizzamenu': require('./components/menu/pizzamenu)
```

55. Save the file.

What you just did: The package.json file contains information about the Node project, such as the starter JavaScript file, but it also helps with tracking the project dependencies to other Node.js modules. The custom component SDK has a dependency on joi, a Node.js schema validator module, which needs to be listed in this file. The registry.js file tracks the name of custom components, along with their physical file locations. The existing file contents reference an example component that is not part of the project you are building and needed to be replaced with the name and the location of the pizzamenu.js file, located in the components/menu folder.

56. To understand where the name `'oramag.sample.pizzamenu'` comes from, open the **pizzamenu.js** file, located in the /components/menu folder in Atom, which is also the relative path reference you added to the menu.js reference in the registry.js file (note that in Node.js, you don't need to specify the .js file extension).

The file contains two functions: metadata and invoke. The metadata function describes the custom component and also defines the component name, which, in this example, is `'oramag.sample.pizzamenu'`. The invoke function then holds the code that is executed at runtime to display the menu.

57. Save all files.

58. Open a command-line window.

59. On the command line, navigate to the menuservice folder.

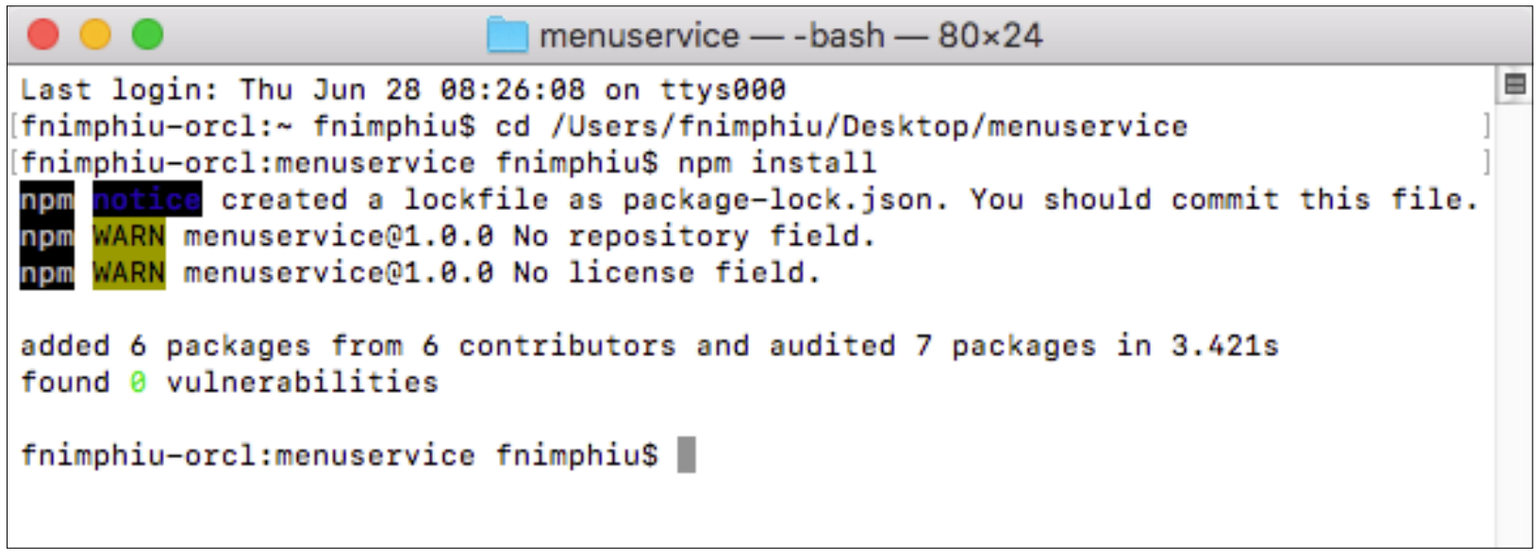
60. On the command line, type `npm install`. You should see a success result similar to that in **Figure 6**.

61. Close the command-line window.

62. In the file system, navigate to the **menuservice** folder and zip it to **menuservice.zip**.

What you just did: With the package.json file containing the reference to the joi Node.js module, you needed to install the module dependency. For this you used

Figure 6: npm installation result



the `npm install` command. The zip file you created at the end now contains the whole component service project, with all its dependencies, ready for you to upload to Oracle Mobile Cloud.

What the pizzamenu.js invoke function does: The custom component you copied and configured in the component service uses its invoke method to write a JSON object to a context variable in the dialog flow. The name of the context variable will be passed by the bot designer in a component property (defined in the metadata function). The invoke function, when called at runtime, obtains a handle to the custom component SDK through the conversation argument. It uses this handle to read the name of the context variable and to write the JSON object to the variable. It then triggers the dialog flow to navigate to the next state in the bot conversation before closing the call. With only a few modifications, you could change the sample to read the pizza menu from a remote web service instead of holding the menu statically.

UPLOADING THE COMPONENT SERVICE

To make the custom component available to the bot, you need to upload the component service implementation (**menuservice.zip**) and expose the component service on a mobile back end.

63. Go back to the browser window you left open in step 34.
64. Click the **Upload an implementation archive** link, and browse to the location in your file system where you saved the **menuservice.zip** file.
65. Select the file, and click **Open**.
66. Click the hamburger icon at the top left (next to the Oracle logo).
67. Select **Mobile Apps** and then **Backends**.
68. Click **New Backend**.
69. Provide the back-end name `OraMagPizzaMenu`. If you use Oracle Mobile Cloud Enterprise, set **Application** (under the Analytics label) to **None**.
70. Click **Create**.
71. Click the hamburger icon again to close the side menu.
72. In the OraMagPizzaMenu back-end editor, select the **APIs** (⚙) menu option at the left.
73. Click **Select APIs**.
74. Type `MenuService` into the search field, and click the plus icon on the MenuService card.
75. Click the **OraMagPizzaMenu 1.0** link to return to the back-end editor.
76. Keep the window open.

What you just did: First, you imported the custom component service implementation to Oracle Mobile Cloud so it can be exposed to the bot. To expose a custom API in Oracle Mobile Cloud, a back end is required. The back end is what you created

next. You then associated the custom component service API with the back end to make it accessible for the bot.

CONFIGURING THE CUSTOM COMPONENT SERVICE WITH THE BOT

We're almost done. The two steps missing before the pizza menu can be read from the custom component are to register the component service with the bot and to replace the menu defined in the dialog flow with a reference to the custom component.

77. Go back to the browser window you left open in step 9. It's the window with the sample bot.
78. Select the **Components** menu item () from the left-side menu to get to the custom component service registration page.
79. Click the **+ Service** button.
80. Define the **Name** field value as `PizzaMenuService`.
81. Ensure that **Mobile Cloud** is selected.
82. Select the checkbox with the label **use anonymous access**.

Next you need three types of information from the Oracle Mobile Cloud back end you left open in step 75.

83. In the back-end editor, click the **Settings** menu option.
84. Copy the following values to a text file: **Backend Id**, **Anonymous Key** (you need to click the **Show** link to see the key), and **Base URL**.
85. Go back to the bot.
86. Copy **Backend Id** into the **Backend Id** field and **Anonymous Key** into the **Anonymous Key** field.
87. In the **Metadata URL** field, add the following value:

```
<Base URL>/mobile/custom/MenuService/components
```

88. Click **Create**. If all went well, you should see a message such as “You successfully created the PizzaMenuService service.”
89. Expand the **PizzaMenuService** menu that now shows on the page, and click the `"oramag.sample.pizzamenu"` item.
90. Take note of the component name `"oramag.sample.pizzamenu"` and the name of the required property, `MenuVariableName`.
91. Click the **Flows** menu item ( ).
92. Find and delete the whole `InitializeMenu` state down to and including the `transitions: {}` line. Don’t touch the setCardsRangeStart state beneath.
93. Add the following code snippet above `setCardsRangeStart`:

```
InitializeMenu:
        component: "oramag.sample.pizzamenu"
        properties:
          menuVariableName: "pizzaMenu"
        transitions: {}
```

Note: It is important to get the indenting right, because otherwise validation in the dialog flow editor will fail. Each indent is exactly two spaces. The `InitializeMenu` string is indented two spaces from the left margin; the `component`, `properties`, and `transitions` strings are indented four spaces from the left margin; and the `menuVariableName` string is indented six spaces from the left margin.

94. Click the **Validate** link at the top right of the dialog flow editor.
95. If validation succeeded, click the **Test** icon (▶).
96. Type `Hi` into the **Message** field, and click **Send**.
97. Then type `Show me the menu` into the **Message** field, and click **Send**.
98. Choose a pizza, and complete the order process.
99. Click the **Reset** button at the top right of the embedded tester.
100. Try the bot again, but this time type `I like to order a pizza`.
101. Choose a pizza, and complete the order process.
What you just did: In this last section of the hands-on instructions, you configured the custom component service you created and deployed it to Oracle Mobile Cloud with the sample bot. You then replaced the existing menu definition with a reference to the custom component so that the next time the bot runs, the menu will be read from the cloud.

CONCLUSION

Custom components are an important part of the Oracle Intelligent Bots architecture. They enable you to add custom logic to and integrate back-end services into your bot conversations.

Building custom components and custom component services for Oracle Intelligent Bots is a developer-specific task. In a bot project, you may separate the team by having bot designers building the dialog flow and JavaScript developers providing the custom components.

Note that for this article, I simplified and shortcut the custom component service creation process with starter templates and files. (How to connect custom components with remote services is explained in one of the Oracle TechExchange articles

linked to at the end of this article.) The main focus in this hands-on demo was on the configuration of the custom component service and less on the programming. Building custom components, however, is easy to do and actually fun too.

Frank Nimphius is a master principal product manager in the Oracle Mobile Platform Product Management group, where he focuses on Oracle Mobile Cloud Enterprise, chatbots, and content experience and analytics (CxA).

ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

TRY Oracle Mobile Cloud Enterprise and Oracle Intelligent Bots.

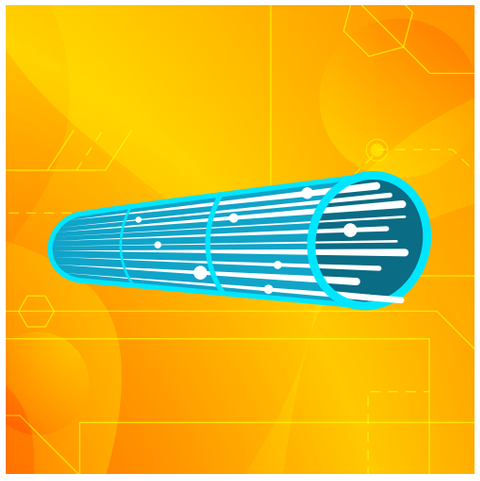
DOWNLOAD the resources for this article.

LEARN more about Oracle Mobile Cloud Enterprise and Oracle Intelligent Bots.

building custom components (TechExchange article).

back-end system integration (TechExchange article).

building card layout bot responses from custom components (TechExchange article).



ORACLE DATABASE

Pipelined Table Functions

Pass data back to the calling query before the function is completed.

In my two previous articles "When Is a Function Like a Table? When It's a Table Function!" and "Streaming Table Functions," I introduced table functions—functions that can be called in the `FROM` clause of a `SELECT` statement as if they were tables—and then took a closer look at streaming table functions. In this article, I explore a very special kind of table function: the *pipelined* table function.

Pipelined table functions are table functions that return or "pipe" rows back to the calling query as the function produces the data in the desired form—and *before* the function has completed all of its processing.

Before we dive into the implementation and application of pipelined table functions, it is important to understand how unusual the above statement is. PL/SQL is *not* a multithreaded language. Generally, when a PL/SQL block (anonymous, nested, procedure function, and so on) is invoked, further processing in that session is "on hold" (suspended) until the block returns control to the host that invoked the block—

whether that host is another PL/SQL block, a SQL statement, or a host language such as Java.

Normal (nonpipelined) table functions act in precisely this way. Each time the table function is invoked in the `FROM` clause, the SQL engine must wait until a `RETURN` statement is executed to pass back the collection to the `SELECT` statement for conversion into rows and columns.

This blocking behavior can have a negative impact on overall performance of the `SELECT` statement, especially in the ETL (extract, transform, and load) operations of a data warehouse. In addition, with each element added to the collection in the table function, more and more Program Global Area (PGA) memory is consumed. For very large datasets, this can lead to further performance degradation and even errors.

Pipelined table functions get around both of these problems. Let's take a look.

A VERY SIMPLE EXAMPLE

Let's start our exploration of pipelined table functions (which I also refer to as PTFs in this article) by comparing a very simple regular (nonpipelined) table function with a very simple PTF.

First, I create a schema-level nested table type of strings.

```
CREATE OR REPLACE TYPE strings_t IS TABLE OF VARCHAR2 (100);
/
```

Next, I compile a table function that returns a nested table of that type with a single string inside it:

```
CREATE OR REPLACE FUNCTION strings
   RETURN strings_t
   AUTHID DEFINER
IS
   l_strings strings_t := strings_t ('abc');
BEGIN
   RETURN l_strings;
END;
/
```

When I call the table function in a `SELECT` statement, it returns `abc`:

```
SELECT COLUMN_VALUE my_string FROM TABLE (strings ())
/
MY_STRING
_________
abc
```

Now I will create a pipelined version of that same table function.

```
CREATE OR REPLACE FUNCTION strings_pl
   RETURN strings_t
   PIPELINED
   AUTHID DEFINER
IS
BEGIN
```

```
   PIPE ROW ('abc');
   RETURN;
END;
/
```

And when I run it, I see the same results:

```
SELECT COLUMN_VALUE my_string FROM TABLE (strings_pl ())
/
MY_STRING
_________
abc
```

Now let's explore the differences between the code in these regular and pipelined table functions:

Regular table functions	Pipelined table functions
No keyword in header	`PIPELINED` appears in header
Declares local collection	No declaration of a local collection
Populates collection with string	Passes string to `PIPE ROW`
Returns the collection	Returns nothing but control

And there's one other big difference between regular and pipelined table functions: regular table functions can be called in PL/SQL, but pipelined table functions can be called only from within a `SELECT` statement, as you can see in the error message below:

```
DECLARE
   l_strings   strings_t := strings_t ();
BEGIN
   l_strings := strings_pl ();
END;
/

PLS-00653: aggregate/table functions are not allowed in PL/SQL scope
```

This should not be entirely unexpected. Because PL/SQL is not a multithreaded language, it cannot accept rows that are "piped" back before the function terminates execution.

Of course, in this incredibly basic pipelined table function, there will be no issue of blocking or memory consumption. But it gets across the basic differences in the process flow of a pipelined table function:

1. Add the `PIPELINED` keyword to the function header.
2. Use `PIPE ROW` to send the data back to the calling SELECT statement instead of adding the data to a local collection.
3. Return nothing but control.

A NONTRIVIAL PIPELINED TABLE FUNCTION

Before taking a look at how pipelined table functions can help improve performance over nonpipelined table functions and also reduce PGA memory consumption, let's build a more interesting PTF.

My transformation in this example is fairly trivial—and could even be handled with a table function, with "pure SQL." The data transformation requirement is to

split each row in the `STOCKS` table into two rows in the `TICKERS` table.

I create the two tables:

```
CREATE TABLE stocks
(
   ticker        VARCHAR2 (20),
   trade_date    DATE,
   open_price    NUMBER,
   close_price   NUMBER
)
/

CREATE TABLE tickers
(
   ticker      VARCHAR2 (20),
   pricedate   DATE,
   pricetype   VARCHAR2 (1),
   price       NUMBER
)
/
```

I then create the object type and nested table type needed for the table functions:

```
CREATE TYPE ticker_ot AS OBJECT
(
   ticker VARCHAR2 (20),
```

```
   pricedate DATE,
   pricetype VARCHAR2 (1),
   price NUMBER
);
/

CREATE TYPE tickers_nt AS TABLE OF ticker_ot;
/
```

As discussed in the streaming table function article, I will need to pass a dataset to the table functions, and for that I need a `REF CURSOR` type:

```
CREATE OR REPLACE PACKAGE stock_mgr
   AUTHID DEFINER
IS
   TYPE stocks_rc IS REF CURSOR
      RETURN stocks%ROWTYPE;
END stock_mgr;
/
```

I am now ready to create my table functions. The nonpipelined version is shown in **Listing 1**. It follows the typical streaming table function pattern:

- Fetch rows from the incoming dataset.
- Perform the transformation.
- Put the transformed rows into the nested table.
- Return the nested table.

Now let's see how this pattern unfolds in my `doubled_nopl` function—one `STOCKS` row doubled to two `TICKER` rows—and the line descriptions that follow.

Listing 1: Nonpipelined table function

```
1  CREATE OR REPLACE FUNCTION doubled_nopl (rows_in stock_mgr.stocks_rc)
2     RETURN tickers_nt
3     AUTHID DEFINER
4  IS
5     TYPE stocks_aat IS TABLE OF stocks%ROWTYPE
6        INDEX BY PLS_INTEGER;
7
8     l_stocks    stocks_aat;
9
10    l_doubled   tickers_nt := tickers_nt ();
11 BEGIN
12    LOOP
13       FETCH rows_in BULK COLLECT INTO l_stocks LIMIT 100;
14
15       EXIT WHEN l_stocks.COUNT = 0;
16
17       FOR l_row IN 1 .. l_stocks.COUNT
18       LOOP
19          l_doubled.EXTEND;
20          l_doubled (l_doubled.LAST) :=
21             ticker_ot (l_stocks (l_row).ticker,
22                        l_stocks (l_row).trade_date,
```

```
23                      'O',
24                      l_stocks (l_row).open_price);
25
26         l_doubled.EXTEND;
27         l_doubled (l_doubled.LAST) :=
28            ticker_ot (l_stocks (l_row).ticker,
29                       l_stocks (l_row).trade_date,
30                       'C',
31                       l_stocks (l_row).close_price);
32      END LOOP;
33   END LOOP;
34
35      RETURN l_doubled;
36   END;
```

Line(s)	Description
1	Use the `REF CURSOR` type defined in the package for the rows passed in. Because I am selecting from the `STOCKS` table, I use the `stocks_rc` type.
2	Return an array, each of whose elements looks *just like* a row in the `TICKERS` table.
5–6	Declare an associative array to hold rows fetched from the `rows_in` cursor variable.
8	Declare the local variable to be filled and then returned back to the `SELECT` statement.
12	Start up a simple loop to fetch rows from the cursor variable. It's already open—the `CURSOR` expression takes care of that.
13–15	Use the `BULK COLLECT` feature to retrieve as many as 100 rows with each fetch. I do this to avoid row-by-row processing, which is not efficient enough. Exit the loop when the associative array is empty.

Line(s)	Description
17	For each element in the array (row from the cursor variable)...
19–24	Use `EXTEND` to add another element at the end of the nested table, then call the object type constructor to create the first ("opening") of the two rows for the `TICKERS` table, and put the element into the new last index value of the collection.
26–31	Do the same for the second ("closing") row of the `TICKERS` table.
35	Send the nested table back to the `SELECT` statement for streaming.

Now let's create the *pipelined* version of the above function (see **Listing 2**) and call it `doubled_pl`. Take a look at the line descriptions that follow.

Listing 2: Pipelined table function

```
1  CREATE OR REPLACE FUNCTION doubled_pl (rows_in stock_mgr.stocks_rc)
2     RETURN tickers_nt
3     PIPELINED
4     AUTHID DEFINER
5  IS
6     TYPE stocks_aat IS TABLE OF stocks%ROWTYPE
7        INDEX BY PLS_INTEGER;
8
9     l_stocks   stocks_aat;
10 BEGIN
11    LOOP
12       FETCH rows_in BULK COLLECT INTO l_stocks LIMIT 100;
13
```

```
14      EXIT WHEN l_stocks.COUNT = 0;
15
16      FOR l_row IN 1 .. l_stocks.COUNT
17      LOOP
18         PIPE ROW (ticker_ot (l_stocks (l_row).ticker,
19                              l_stocks (l_row).trade_date,
20                              'O',
21                              l_stocks (l_row).open_price));
22         PIPE ROW (ticker_ot (l_stocks (l_row).ticker,
23                              l_stocks (l_row).trade_date,
24                              'C',
25                              l_stocks (l_row).close_price));
26      END LOOP;
27   END LOOP;
28
29   RETURN;
30 END;
```

Line(s)	Description
1	Because this is a streaming table function, it should come as no surprise that the parameter to the function is a cursor variable passing in rows of data from the invoking SELECT statement.
3	Specify that this is a pipelined table function.
6–7	Declare the associative array used to retrieve rows fetched from the cursor variable.
12–14	Fetch the next 100 rows; stop when the collection is empty.
16	For each row of data moved to the collection…

Line(s)	Description
18–21	Use the object type constructor function to "transfer" opening-price data from the element in the collection to the attributes of the object type instance. Then immediately send that data back to the calling query with `PIPE ROW`.
22–25	Do the same thing for the closing-price data.
29	Return control back to the query.

Note that you do not need to explicitly close the cursor variable; Oracle Database will automatically close a cursor variable created with the `CURSOR` expression when the table function terminates.

Before exploring the impact on performance and memory, let's verify that this pipelined table function can be used in a `SELECT` statement just like the nonpipelined version.

```
SELECT COUNT (*) FROM stocks
/

100000

INSERT INTO tickers
   SELECT * FROM TABLE (doubled_pl (CURSOR (SELECT * FROM stocks)))
/

SELECT COUNT (*)
  FROM tickers
```

```
/

200000
```

OK, that looks good.

IMPACT OF SWITCH TO PIPELINED TABLE FUNCTIONS

Let's compare the behavior of streaming table functions, nonpipelined and pipelined. As discussed in my previous article on table functions, a streaming table function streams data directly from one process or transformation to the next without intermediate staging. It is a common use case in data warehouses.

Now let's prove that the SQL engine is able to take those piped rows and put them immediately to work—and use less PGA memory along the way.

Proofs require hard facts, so I will construct a small utilities package to measure elapsed CPU time and memory consumption.

Note: For this package to compile in your schema, you will need the `SELECT` privilege on `v$mystat` and `v$statname`.

First, I create my API to the utility's functionality:

```
CREATE OR REPLACE PACKAGE utils
IS
   PROCEDURE initialize (context_in IN VARCHAR2);

   PROCEDURE show_results (message_in IN VARCHAR2 := NULL);
END;
/
```

Then I compile the utility's package body. **Listing 3** includes the code, and a description of all the interesting bits follows.

Listing 3: utils package body code

```
CREATE OR REPLACE PACKAGE BODY utils
IS
   last_timing   INTEGER := NULL;
   last_pga      INTEGER := NULL;

   FUNCTION pga_consumed
      RETURN NUMBER
   AS
   l_pga   NUMBER;
BEGIN
   SELECT st.VALUE
     INTO l_pga
     FROM v$mystat st, v$statname sn
    WHERE st.statistic# = sn.statistic# AND sn.name = 'session pga memory';

   RETURN l_pga;
END;

PROCEDURE initialize (context_in IN VARCHAR2)
IS
BEGIN
```

```
      DELETE FROM tickers;

      COMMIT;
      DBMS_OUTPUT.put_line (context_in);
      last_timing := DBMS_UTILITY.get_cpu_time;
      last_pga := pga_consumed;
   END;

   PROCEDURE show_results (message_in IN VARCHAR2 := NULL)
   IS
      l_count   INTEGER;
   BEGIN
      SELECT COUNT (*) INTO l_count FROM tickers;

      DBMS_OUTPUT.put_line ('Ticker row count: ' || l_count);
      DBMS_OUTPUT.put_line (
            '"'
         || message_in
         || '" completed in: '
         || TO_CHAR (DBMS_UTILITY.get_cpu_time - last_timing)
         || ' centisecs; pga at: '
         || TO_CHAR (pga_consumed () - last_pga)
         || ' bytes');
   END;
END;
/
```

Line(s)	Description
11–14	Query the v$ views to retrieve the current value for PGA memory consumption in my session.
19–28	Get things ready to test. Clear out the TICKERS table; get the start time and the starting level of PGA memory consumption.
30–45	Post the execution of the code. Get the number of rows in the TICKERS table, calculate and display the number of elapsed centiseconds (hundredths of a second), and calculate and display the change in PGA memory.

Now I am ready to run my test code as follows:

- Initialize starting points in my session.
- Call the pipelined version of the doubled function (`doubled_pl`) to insert rows into the `TICKERS` table. But *add a ROWNUM clause* to say, "I want only the first 9 rows."
- Do the same for the nonpipelined version of the function (`doubled_nopl`).

```
BEGIN
   utils.initialize ('Pipelined');

   INSERT INTO tickers
      SELECT *
        FROM TABLE (doubled_pl (CURSOR (SELECT * FROM stocks)))
       WHERE ROWNUM < 10;

utils.show_results ('First 9 rows');

   utils.initialize ('Not Pipelined');
```

```
   INSERT INTO tickers
      SELECT *
        FROM TABLE (doubled_nopl (CURSOR (SELECT * FROM stocks)))
       WHERE ROWNUM < 10;

   utils.show_results ('First 9 rows');
END;
/
```

Here are the results I got for an initial load of 200,000 rows into the `STOCKS` table.

```
Pipelined
"First 9 rows" completed in: 8 centisecs; pga at: 528345 bytes
Ticker row count: 9

Not Pipelined
"First 9 rows" completed in: 93 centisecs; pga at: 96206848 bytes
Ticker row count: 9
```

The significantly shorter response time with the pipelined function demonstrates clearly that the `INSERT-SELECT` statement was able to keep track of the rows returned by the function. As soon as nine rows were passed back, the SQL engine terminated execution of the pipelined table function and inserted the rows.

With the nonpipelined version, I have to wait for 10,000 rows to be doubled into 20,000 rows (consuming lots of PGA memory as well). Then all those rows are

NO CLAUSE NEEDED

As of Oracle Database 12c Release 2, you no longer need to include the TABLE clause with table functions. This query, calling the `strings` table function, will work just as well:

```
SELECT COLUMN_VALUE my_string
FROM strings ()
```

passed back to the `SELECT` statement, at which point the SQL engine says, "Well, I wanted just the first nine" and throws away the rest.

Very inefficient.

From a memory standpoint, the nonpipelined table function consumes much more PGA memory than the pipelined version. That should make perfect sense to you, because I do not declare and fill up a collection to be returned by the function.

THE NO_DATA_NEEDED EXCEPTION

Sometimes—as in the performance and memory test in the previous section—you will want to terminate the pipelined table function before all rows have been piped back. Oracle Database will then raise the `NO_DATA_NEEDED` exception. This will terminate the function but will *not* terminate the `SELECT` statement that called it.

So for the most part, you don't have to worry about this exception. You *do* need to explicitly handle this exception if either of the following applies:

- You include an `OTHERS` exception handler in a block that includes a `PIPE ROW` statement.
- Your code that feeds a `PIPE ROW` statement must be followed by a cleanup procedure. Typically, the cleanup procedure releases resources the code no longer needs.

Let's explore this behavior in more detail. In this first example, I use only one row, so Oracle Database raises `NO_DATA_NEEDED` but no exception is raised.

```
CREATE OR REPLACE TYPE strings_t IS TABLE OF VARCHAR2 (100);
/
```

```
CREATE OR REPLACE FUNCTION strings
   RETURN strings_t
   PIPELINED
   AUTHID DEFINER
IS
BEGIN
   PIPE ROW (1);
   PIPE ROW (2);
   RETURN;
END;
/
```

```
SELECT COLUMN_VALUE my_string
  FROM TABLE (strings ())
 WHERE ROWNUM < 2
/

MY_STRING
_________
1
```

Now I add an `OTHERS` exception handler and nothing else.

```
CREATE OR REPLACE FUNCTION strings
   RETURN strings_t
   PIPELINED
```

```
   AUTHID DEFINER
IS
BEGIN
   PIPE ROW (1);
   PIPE ROW (2);
   RETURN;
EXCEPTION
   WHEN OTHERS
   THEN
      DBMS_OUTPUT.put_line ('Error: ' || SQLERRM);
      RAISE;
END;
/

SELECT COLUMN_VALUE my_string
  FROM TABLE (strings ())
 WHERE ROWNUM < 2
/

MY_STRING
_________
1

Error: ORA-06548: no more rows needed
```

As you can see, the NO_DATA_NEEDED error is trapped by that handler and the reraise does not manifest as an error in the SELECT statement. The problem with taking this approach, though, is that your OTHERS handler might contain specific cleanup code that makes sense for unexpected failures but not for *early termination* of data piping. I therefore recommend that you include a specific handler for NO_DATA_NEEDED.

In the code below, I demonstrate that both NO_DATA_FOUND and NO_DATA_NEEDED are by default "hidden" from the calling query but other exceptions such as PROGRAM_ERROR result in termination of the SQL statement.

```
CREATE OR REPLACE FUNCTION strings (err_in IN VARCHAR2)
   RETURN strings_t
   PIPELINED
   AUTHID DEFINER
IS
BEGIN
   PIPE ROW (1);

   CASE err_in
      WHEN 'no_data_found'
      THEN
         RAISE NO_DATA_FOUND;
      WHEN 'no_data_needed'
      THEN
         RAISE no_data_needed;
```

```
      WHEN 'program_error'
      THEN
         RAISE PROGRAM_ERROR;
   END CASE;

   RETURN;
END;
/

SELECT COLUMN_VALUE my_string FROM TABLE (strings ('no_data_found'))
/

MY_STRING
_________

1

SELECT COLUMN_VALUE my_string FROM TABLE (strings ('no_data_needed'))
/

MY_STRING
_________

1

SELECT COLUMN_VALUE my_string FROM TABLE (strings ('program_error'))
/
```

```
MY_STRING
_________

1
ORA-06501: PL/SQL: program error
```

The basic takeaway regarding `NO_DATA_NEEDED` is not to worry about it unless you are providing a `WHEN OTHERS` handler in your pipelined table function. In such a case, make sure to provide a handler for `NO_DATA_NEEDED`, simply reraising the exception with a `RAISE` statement.

PACKAGE-BASED TYPES IMPLICITLY DECLARED

With pipelined table functions *only*, your types can be defined in the specification of a package, as opposed to in the schema with independent `CREATE OR REPLACE` statements. You can even declare your nested table as a collection of record types.

```
CREATE TABLE stocks2
(
   ticker        VARCHAR2 (20),
   trade_date    DATE,
   open_price    NUMBER,
   close_price   NUMBER
)
/

CREATE OR REPLACE PACKAGE pkg
```

```
   AUTHID DEFINER
AS
   TYPE stocks_nt IS TABLE OF stocks2%ROWTYPE;

   FUNCTION stock_rows
      RETURN stocks_nt
      PIPELINED;
END;
/

CREATE OR REPLACE PACKAGE BODY pkg
AS
   FUNCTION stock_rows
      RETURN stocks_nt
      PIPELINED
   IS
      l_stock   stocks2%ROWTYPE;
   BEGIN
      l_stock.ticker := 'ORCL';
      l_stock.open_price := 100;
      PIPE ROW (l_stock);
      RETURN;
   END;
END;
/
```

```
SELECT ticker, open_price FROM TABLE (pkg.stock_rows ())
/
```

This is more convenient, but behind the scenes, Oracle Database is creating types implicitly for you, as you can see below.

```
SELECT object_name, object_type
  FROM user_objects
 WHERE object_type IN ('TYPE', 'PACKAGE', 'PACKAGE BODY')
/

OBJECT_NAME                    OBJECT_TYPE
------------------------------ ------------
SYS_PLSQL_B85654CF_DUMMY_1     TYPE
SYS_PLSQL_B85654CF_9_1         TYPE
SYS_PLSQL_4DF80292_DUMMY_1     TYPE
SYS_PLSQL_4DF80292_30_1        TYPE
PKG                            PACKAGE
PKG                            PACKAGE BODY
```

SUMMARY

Pipelined table functions are something of an oddity in PL/SQL. They pass data back to the calling query even before the function is completed, and they don’t pass back anything but control with the `RETURN` statement. You cannot call a PTF from within PL/SQL itself, only in the `FROM` clause of a `SELECT` statement.

But those oddities reflect the power of this special type of function: improved performance and reduced memory consumption compared to normal (nonpipelined) table functions.

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.*

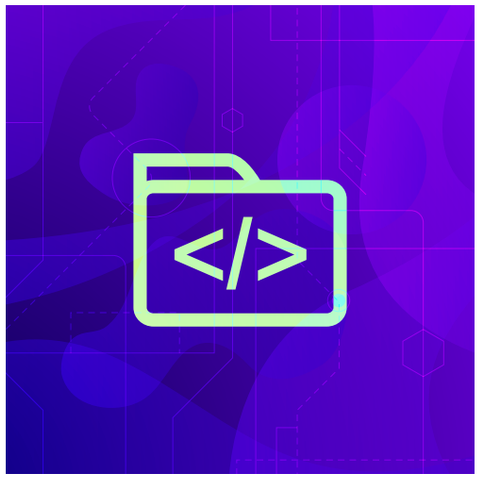
ILLUSTRATION BY **WES ROWELL**

NEXT STEPS

READ documentation about creating pipelined table functions.

READ more about pipelined table functions (ORACLE-BASE).

LEARN more about PL/SQL table functions (a Dev Gym class).



ORACLE DATABASE

Build REST APIs for Node.js, Part 4

Complete the API by adding support for PUT, POST, and DELETE requests.

In the previous article in this series, you added logic to the REST API for `GET` requests, which retrieve data from the database. In this article, you will finish building out the basic create, read, update, and delete (CRUD) functionality of the API by adding logic to handle `POST`, `PUT`, and `DELETE` requests on the employees endpoint.

Note that the instructions and steps in this article assume that you have completed the steps in part 3 of this article series.

ADDING THE ROUTING LOGIC

To keep the routing logic simple, you will route all HTTP methods through the existing route path (with the optional `id` parameter). To add the routing logic, open the services/router.js file and replace the current routing logic (lines 5 and 6) with the following code:

```
router.route('/employees/:id?')
  .get(employees.get)
  .post(employees.post)
  .put(employees.put)
  .delete(employees.delete);
```

The updated routing logic maps the four most common HTTP methods used for basic CRUD operations to the correct controller logic. That controller logic, as well as the related database logic, will be built out over the next three sections of this article.

HANDLING POST REQUESTS

HTTP POST requests are used to create new resources (employee records, in this case). The basic idea is to pull data out of the HTTP request body and use it to create a new row in the database.

To add the controller logic for POST requests, open the controllers/employees.js file and append the following code to the bottom of the file.

```
function getEmployeeFromRec(req) {
  const employee = {
    first_name: req.body.first_name,
    last_name: req.body.last_name,
    email: req.body.email,
    phone_number: req.body.phone_number,
    hire_date: req.body.hire_date,
    job_id: req.body.job_id,
```

```
    salary: req.body.salary,
    commission_pct: req.body.commission_pct,
    manager_id: req.body.manager_id,
    department_id: req.body.department_id
  };

  return employee;
}

async function post(req, res, next) {
  try {
    let employee = getEmployeeFromRec(req);

    employee = await employees.create(employee);

    res.status(201).json(employee);
  } catch (err) {
    next(err);
  }
}

module.exports.post = post;
```

The `getEmployeeFromRec` function accepts a request object and returns an object with the properties needed to create an employee record. The function was declared outside of the `post` function so that it can be used later for `PUT` requests as well.

The `post` function uses `getEmployeeFromRec` to initialize a variable that is then passed to the `create` method of the employees database API. After the create operation, a `201 Created` status code, along with the employee JSON data (including the new employee ID value), is sent to the client.

Now you can turn your attention to the create logic in the database API. Open the db_apis/employees.js file, and append the following code to the bottom of the file.

```
const createSql =
 'insert into employees (
    first_name,
    last_name,
    email,
    phone_number,
    hire_date,
    job_id,
    salary,
    commission_pct,
    manager_id,
    department_id
  ) values (
    :first_name,
    :last_name,
    :email,
    :phone_number,
    :hire_date,
    :job_id,
```

```
    :salary,
    :commission_pct,
    :manager_id,
    :department_id
  ) returning employee_id
  into :employee_id';

async function create(emp) {
  const employee = Object.assign({}, emp);

  employee.employee_id = {
    dir: oracledb.BIND_OUT,
    type: oracledb.NUMBER
  }

  const result = await database.simpleExecute(createSql, employee);

  employee.employee_id = result.outBinds.employee_id[0];

  return employee;
}

module.exports.create = create;
```

The logic above starts by declaring a constant named `createSql` to hold an `INSERT` statement. Note that it uses bind variables, not string concatenation, to reference the

values to be inserted. It's worth repeating how important bind variables are to security and performance. Try to avoid string concatenation with SQL statements whenever possible.

Within the `create` function, an `employee` constant is defined and initialized to a copy of the `emp` parameter, using `Object.assign`. This prevents direct modification of the object passed in from the controller. It's a shallow copy, but that's sufficient for this use case.

Next an `employee_id` property is added to the `employee` object (configured as an "out bind"), so that it contains all the bind variables required to execute the SQL statement. The `simpleExecute` function is then used to execute the `INSERT` statement, and the `outBinds` property of the result is used to overwrite the `employee.employee_id` property before the object is returned.

Because the `oracledb` module is referenced, you'll need to require that in the code. Add the following line to the top of the db_apis/employees.js file.

```
const oracledb = require('oracledb');
```

HANDLING PUT REQUESTS

`PUT` requests will be used to make updates to existing resources. It's important to note that `PUT` is used to replace the entire resource—it doesn't do partial updates. (I will show you how to do this with `PATCH` in the future.) Return to the controllers/employees.js file and append the following code to the bottom of the file.

```
async function put(req, res, next) {
  try {
    let employee = getEmployeeFromRec(req);
```

```
    employee.employee_id = parseInt(req.params.id, 10);

    employee = await employees.update(employee);

    if (employee !== null) {
      res.status(200).json(employee);
    } else {
      res.status(404).end();
    }
  } catch (err) {
    next(err);
  }

module.exports.put = put;
```

The `put` function uses `getEmployeeFromRec` to initialize an object named `employee` and then adds an `employee_id` property from the `id` parameter in the URL. The employee object is then passed to the database API's `UPDATE` function, and the appropriate response is sent to the client, based on the result.

To add the update logic to the database API, append the following code to the bottom of the db_apis/employees.js file.

```
const updateSql =
 'update employees
  set first_name = :first_name,
```

```
    last_name = :last_name,
    email = :email,
    phone_number = :phone_number,
    hire_date = :hire_date,
    job_id = :job_id,
    salary = :salary,
    commission_pct = :commission_pct,
    manager_id = :manager_id,
    department_id = :department_id
  where employee_id = :employee_id';

async function update(emp) {
  const employee = Object.assign({}, emp);
  const result = await database.simpleExecute(updateSql, employee);

  if (result.rowsAffected && result.rowsAffected === 1) {
    return employee;
  } else {
    return null;
  }
}

module.exports.update = update;
```

The `UPDATE` logic is very similar to the `CREATE` logic. A variable to hold a SQL statement is declared, and then `simpleExecute` is used to execute the statement

with the dynamic values passed in (after they have been copied to another object). To determine if the `UPDATE` was successful and to return the correct value, `result.rowsAffected` is used.

HANDLING DELETE REQUESTS

The last method you need to implement is `DELETE`, which, unsurprisingly, deletes resources from the database. Append the following code to the end of the controllers/employees.js file.

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

    const success = await employees.delete(id);

    if (success) {
      res.status(204).end();
    } else {
      res.status(404).end();
    }
  } catch (err) {
    next(err);
  }
}

module.exports.delete = del;
```

The JavaScript engine will throw an exception if you try to declare a function named `delete` by using a `function` statement. To work around this, a function named `del` is declared and then exported as `delete`.

At this point, you should be able to read and understand the logic. Unlike the previous `GET`, `POST`, and `PUT` examples, this HTTP request doesn't have a body; only the `id` parameter in the route path is used. The `204 No Content` status code is often used with `DELETE` requests when no response body is sent.

To complete the database logic, return to the db_apis/employees.js file and append the following code to the end.

```
const deleteSql =
 'begin
    delete from job_history
    where employee_id = :employee_id;

    delete from employees
    where employee_id = :employee_id;

    :rowcount := sql%rowcount;

  end;'

async function del(id) {
  const binds = {
    employee_id: id,
```

```
    rowcount: {
      dir: oracledb.BIND_OUT,
      type: oracledb.NUMBER
    }
  }
  const result = await database.simpleExecute(deleteSql, binds);

  return result.outBinds.rowcount === 1;
}

module.exports.delete = del;
```

Because the JOB_HISTORY table has a foreign key constraint that references the EMPLOYEES table, a simple PL/SQL block is used to delete the necessary rows from both tables in a single round-trip.

PARSING JSON REQUEST BODIES

If you look back at the `getEmployeeFromRec` function in the controller, you'll notice that the request's `body` property is a JavaScript object. This provides an easy way to get values from the body of the request, but it's not something that happens automatically.

The API you are building expects clients to send JSON-formatted data in the body of `POST` and `PUT` requests. In addition, clients should set the `Content-Type` header of the request to `application/json` to let the web server know what type of request body is being sent. You can use the built-in express.json middleware function to have Express parse such requests.

Open the services/web-server.js file and add the following lines just below the `app.use` call that adds morgan to the request pipeline.

```
// Parse incoming JSON requests and revive JSON.
app.use(express.json({
  reviver: reviveJson
}));
```

When JSON data is parsed into native JavaScript objects, only the data types supported in JSON will be correctly mapped to JavaScript types. Dates are not supported in JSON and typically need to be represented as ISO 8601 strings. Using a reviver function passed to the `express.json` middleware function, you can do the conversions manually. Append the following code to the bottom of the services/web-server.js file.

```
const iso8601RegExp = /^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d{3})?Z$/;

function reviveJson(key, value) {
  // revive ISO 8601 date strings to instances of Date
  if (typeof value === 'string' && iso8601RegExp.test(value)) {
    return new Date(value);
  } else {
    return value;
  }
}
```

TESTING THE API

It's time to test the new CRUD functionality. Up until now, you've been using Firefox to test the API, but this will not work for `POST`, `PUT`, and `DELETE` requests. I'll show you how to test the API with curl commands, because those are readily available in the VM. However, you might want to look into using a graphical tool such as Postman (free) or Paw (Mac only, not free).

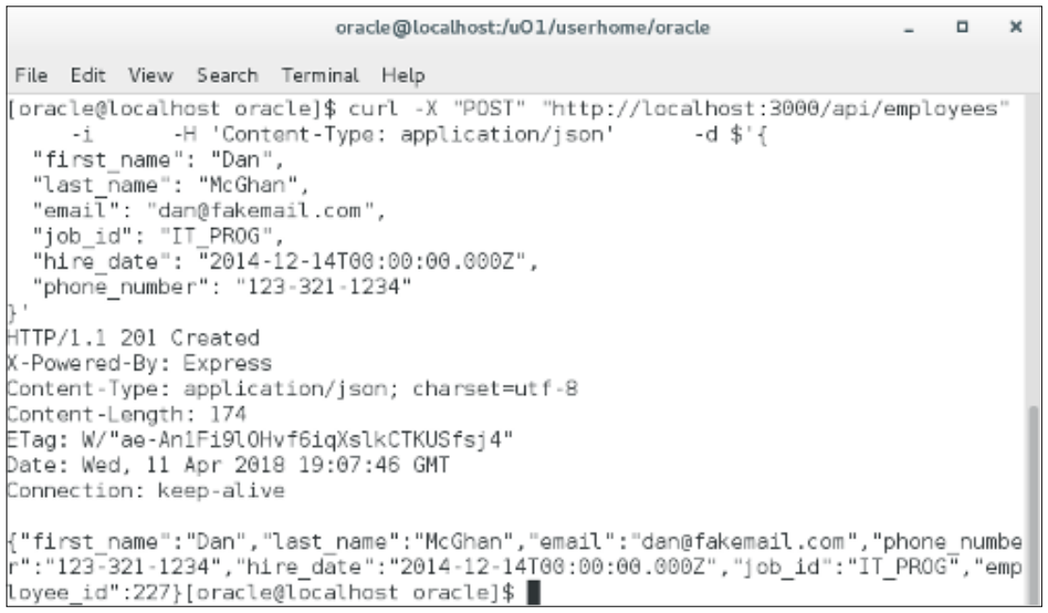
Start the application, open another terminal window, and execute the following command to create a new employee record.

```
curl -X "POST" "http://localhost:3000/api/employees" \
     -i \
     -H 'Content-Type: application/json' \
     -d $'{
  "first_name": "Dan",
  "last_name": "McGhan",
  "email": "dan@fakemail.com",
  "job_id": "IT_PROG",
  "hire_date": "2014-12-14T00:00:00.000Z",
  "phone_number": "123-321-1234"
}'
```

If the request was successful, the response should contain an `employee` object with an `employee_id` attribute. **Figure 1** shows an example.

In my case, the `employee_id` value was 227—you will need to modify the following commands according to the `employee_id` value you get from your own `POST`. For example, to update the new record, issue a `PUT` against the URL with that ID value.

Figure 1: Executing a POST request



According to *my* PUT request, it looks like I'm getting a new job!

```
curl -X "PUT" "http://localhost:3000/api/employees/227" \
     -i \
     -H 'Content-Type: application/json' \
     -d $'{
  "first_name": "Dan",
  "last_name": "McGhan",
  "email": "dan@fakemail.com",
  "job_id": "AD_PRES",
  "hire_date": "2014-12-14T00:00:00.000Z",
  "phone_number": "123-321-1234"
}'
```

The `UPDATE_JOB_HISTORY` trigger in the HR schema will detect the job change and add a row to the JOB_HISTORY table. If you look at that table, you should see a record for the new employee. Execute the following command to delete the job history and employee records.

```
curl -i -X "DELETE" "http://localhost:3000/api/employees/227"
```

And there you have it, full CRUD functionality.

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

READ Part 1 of this article series.

READ Part 2 of this article series.

READ Part 3 of this article series.

LEARN more about JavaScript and Oracle.

TRY Oracle Cloud.

GET this article's code from GitHub.

ORACLE DATABASE

Perform PL/SQL Operations with cx_Oracle

Use Python for PL/SQL operations in Oracle Database.

After you've gotten the hang of performing basic create, retrieve, update, delete (CRUD) operations with the cx_Oracle driver for Python, you're ready to start tapping into some of the real power of Oracle Database.

Python is an excellent language for most things you want your application to do, but when you're processing data, it just goes faster if you do the work where the data is. This article covers how to execute Oracle PL/SQL functions and procedures with Python and cx_Oracle. If you're not already familiar with PL/SQL, you can get some help from Steven Feuerstein and Bryn Llewellyn (see "Additional Resources").

PREREQUISITES

The following are required to run through the steps in this article with me:

- Python 3

- Oracle Database 12c or later
- Basic Oracle PL/SQL and SQL knowledge
- cx_Oracle

To run through the examples in this article, you'll also need to create the following objects in a database schema that is *safe* to experiment in. Make sure you have permissions to create the following objects.

```
CREATE TABLE lcs_people (
 id NUMBER GENERATED BY DEFAULT AS identity,
 name VARCHAR2(20),
 age NUMBER,
 notes VARCHAR2(100)
)
/

ALTER TABLE LCS_PEOPLE
ADD CONSTRAINT PK_LCS_PEOPLE PRIMARY KEY ("ID")
/

CREATE TABLE LCS_PETS (
 id NUMBER GENERATED BY DEFAULT AS IDENTITY,
 name VARCHAR2(20),
 owner NUMBER,
 type VARCHAR2(100)
)
/
```

```
ALTER TABLE LCS_PETS ADD CONSTRAINT PK_LCS_PETS PRIMARY KEY ("ID")
/

ALTER TABLE LCS_PETS ADD CONSTRAINT FK_LCS_PETS_OWNER FOREIGN KEY ("OWNER") REFERENCES
"LCS_PEOPLE" ("ID")
/

INSERT INTO lcs_people (name, age, notes)
 VALUES ('Bob', 35, 'I like dogs')
/

INSERT INTO lcs_people (name, age, notes)
 VALUES ('Kim', 27, 'I like birds')
/

INSERT INTO lcs_pets (name, owner, type)
 VALUES ('Duke', 1, 'dog')
/

INSERT INTO lcs_pets (name, owner, type)
 VALUES ('Pepe', 2, 'bird')
/

COMMIT
/
```

To keep everything clean, I'll be putting my PL/SQL code into a package called `pet_manager`.

```
CREATE OR REPLACE PACKAGE pet_manager AS
    PROCEDURE reset_data;

    PROCEDURE add_pet (
        name_p       IN VARCHAR2,
        owner_id_p   IN NUMBER,
        pet_type_p   IN VARCHAR2
    );

FUNCTION add_pet (
    name_p       IN VARCHAR2,
    owner_id_p   IN NUMBER,
    pet_type_p   IN VARCHAR2
) RETURN NUMBER;

FUNCTION add_pet (
    name_p           IN VARCHAR2,
    owner_id_p       IN NUMBER,
    pet_type_p       IN VARCHAR2,
    need_license_p   OUT VARCHAR2
) RETURN NUMBER;
```

```
END;
/

CREATE OR REPLACE PACKAGE BODY pet_manager IS

  /*
   * Reset the example data
   */

   PROCEDURE reset_data
      AS
   BEGIN
      DELETE FROM lcs_pets;

      EXECUTE IMMEDIATE ('alter table lcs_pets modify id generated BY DEFAULT as
identity (START WITH 3)');

      DELETE FROM lcs_people;

      EXECUTE IMMEDIATE ('alter table lcs_people modify id generated BY DEFAULT as
identity (START WITH 3)');

      INSERT INTO lcs_people (id, name, age, notes)
      VALUES (1, 'Bob', 35, 'I like dogs');
```

```
      INSERT INTO lcs_people (id, name, age, notes)
      VALUES (2, 'Kim', 27, 'I like birds');

      INSERT INTO lcs_pets (id, name, owner, type)
      VALUES (1, 'Duke', 1, 'dog');

      INSERT INTO lcs_pets (id, name, owner, type)
      VALUES (2, 'Pepe', 2, 'bird');

      COMMIT;
   END reset_data;

   /*
    * Add a new Pet
    */

    PROCEDURE add_pet (
       name_p       IN VARCHAR2,
       owner_id_p   IN NUMBER,
       pet_type_p   IN VARCHAR2
    )
       IS
    BEGIN
       INSERT INTO lcs_pets (
          name,
```

```
      owner,
      type
   ) VALUES (
      name_p,
      owner_id_p,
      lower(pet_type_p)
   );

   COMMIT;
END add_pet;

/*
* Add a new Pet return new id
*/

FUNCTION add_pet (
   name_p       IN VARCHAR2,
   owner_id_p   IN NUMBER,
   pet_type_p   IN VARCHAR2
) RETURN NUMBER IS
   new_pet_id   NUMBER;
BEGIN
   INSERT INTO lcs_pets (
      name,
      owner,
```

```
      type
   ) VALUES (
      name_p,
      owner_id_p,
      lower(pet_type_p)
   ) RETURNING id INTO new_pet_id;

   COMMIT;
   RETURN new_pet_id;
END add_pet;

/*
* Add a new Pet return new id and if it needs a license.
*/

FUNCTION add_pet (
   name_p           IN VARCHAR2,
   owner_id_p       IN NUMBER,
   pet_type_p       IN VARCHAR2,
   need_license_p   OUT VARCHAR2
) RETURN NUMBER IS
   new_pet_id   NUMBER;
BEGIN
   INSERT INTO lcs_pets (
      name,
```

```
      owner,
      type
   ) VALUES (
      name_p,
      owner_id_p,
      lower(pet_type_p)
   ) RETURNING id INTO new_pet_id;

   IF lower(pet_type_p) IN ('dog', 'cat') THEN
      need_license_p := 'yes';
   ELSE
      need_license_p := 'no';
      END IF;

      COMMIT;
      RETURN new_pet_id;
   END add_pet;

END pet_manager;
/
```

CLEANUP

To clean up the database when you are finished with the series, you need to drop the two tables and the PL/SQL package. *When you execute the drops, please make sure you're connected to the correct schema where you created the tables*.

```
drop table lcs_pets
/

drop table lcs_people
/

drop package pet_manager
/
```

BOILERPLATE TEMPLATE HEADER

The boilerplate template header for this article assumes that cx_Oracle is installed. Here is the boilerplate code

```
import cx_Oracle
import os
connectString = os.getenv('db_connect')
con = cx_Oracle.connect(connectString)
cur = con.cursor()

# Your code here
```

and a description of what the code does.

1. Imports the cx_Oracle driver
2. Imports the OS module used to read the environment variable
3. Gets the connection string from the environment variable
4. Creates the connection object

5. Creates the cursor object

I will include this code section with all Python examples and use the connection object `con` and the cursor object `cur` throughout this article. For each example in the article, replace the `# Your code here` line above with the code for that example.

ANONYMOUS PL/SQL BLOCK

Let's start off with the most basic process and simply execute an anonymous block of PL/SQL code to reset the database tables.

```
# reset data
statement = """
   BEGIN
DELETE FROM lcs_pets;

EXECUTE IMMEDIATE ('alter table lcs_pets modify id generated
BY DEFAULT as identity (START WITH 3)');

DELETE FROM lcs_people;

EXECUTE IMMEDIATE ('alter table lcs_people modify id generated
BY DEFAULT as identity (START WITH 3)');

INSERT INTO lcs_people (id, name, age, notes)
VALUES (1, 'Bob', 35, 'I like dogs');

INSERT INTO lcs_people (id, name, age, notes)
```

```
        VALUES (2, 'Kim', 27, 'I like birds');

        INSERT INTO lcs_pets (id, name, owner, type)
        VALUES (1, 'Duke', 1, 'dog');

        INSERT INTO lcs_pets (id, name, owner, type)
        VALUES (2, 'Pepe', 2, 'bird');

        COMMIT;
    END;"""

cur.execute(statement)
```

When I run the above code in my Python template, named `anon_plsql.py` for this example,

```
python3 anon_plsql.py
```

the code is successful and does not return a response.

You can execute any data definition language (DDL) or data manipulation language (DML) statement this way, but if you're going to run PL/SQL, it's usually best to compile it to the database.

EXECUTE A PL/SQL PROCEDURE

Using the code from this anonymous block, I create a procedure in the `pet_manager` PL/SQL package called `reset_data`.

To call this procedure from Python, I use the `cursor.callproc` method and pass in the package.procedure name to execute it.

```
cur.callproc('pet_manager.reset_data')
```

When I run the this code line in my Python template, named `execute_procedure.py` for this example,

```
python3 execute_procedure.py
```

the code is successful and does not return a response. When everything works, there will not be any response. So this works as a "fire and forget" way to call database procedures.

PASS PARAMETERS

I have a procedure in my `pet_manager` PL/SQL package that I can use to create a new pet in the LCS_PETS table. The `add_pet` procedure will accept the Python `pet_name`, `owner_id`, and `pet_type` variable values. Using these values, it will insert a new entry into the LCS_PETS table.

```
/*
 * Add a new Pet
 */

  PROCEDURE add_pet (
      name_p       IN VARCHAR2,
```

```
    owner_id_p   IN NUMBER,
    pet_type_p   IN VARCHAR2
)
    IS
BEGIN
    INSERT INTO lcs_pets (
        name,
        owner,
        type
    ) VALUES (
        name_p,
        owner_id_p,
        lower(pet_type_p)
    );

    COMMIT;
END add_pet;
```

On the Python side, I prefer to set my values with variables so that my code is easier to read. I create and set the `pet_name`, `owner_id`, and `pet_type` values. Next, I call the `cursor.callproc` method and add an array containing the values to pass in in the order in which they are defined in the database.

```
pet_name = 'Roger'
owner_id = 1
```

```
pet_type = 'fish'

# Add a new pet
cur.callproc('pet_manager.add_pet',
      [pet_name, owner_id, pet_type])
```

When I run the above code in my Python template, named `pass_parameters.py` for this example,

```
python3 pass_parameters.py
```

the code is successful and does not return a response. Once again, if everything works, there will be no response—no value returned.

GET PL/SQL FUNCTION RETURN VALUES

When a row is added to the LCS_PETS table, a new ID is automatically generated. Having this ID can be useful, so I created an `add_pet` function in my `pet_manager` PL/SQL package that creates a new pet in the LCS_PETS table, and it will return the new ID.

```
/*
   * Add a new Pet return new id
   */

    FUNCTION add_pet (
        name_p       IN VARCHAR2,
```

```
    owner_id_p   IN NUMBER,
    pet_type_p   IN VARCHAR2
) RETURN NUMBER IS
    new_pet_id   NUMBER;
BEGIN
    INSERT INTO lcs_pets (
        name,
        owner,
        type
    ) VALUES (
        name_p,
        owner_id_p,
        lower(pet_type_p)
    ) RETURNING id INTO new_pet_id;

    COMMIT;
    RETURN new_pet_id;
END add_pet;
```

Using Python to call a function in the database and get the return value, I use the `cursor.callfunc` method to do the following:

1. Set the variables to use as arguments to the function.
2. Define a `new_pet_id` variable and assign it the value returned from `callfunc`.
3. Define the type of the data being returned—the second argument of the `callfunc` method does this. I set it to `int`. (cx_Oracle handles the `NUMBER`-to-`int` conversion.)

4. Pass in the array of values (just as I did when I used `callproc`).
5. Print the returned value for `new_pet_id`.

When I run the following code in my Python template, named `execute_function.py` for this example,

```
python3 execute_function.py
```

the code is successful and returns a value for `new_pet_id`.

```
pet_name = 'Roger'
owner_id = 1
pet_type = 'fish'

# add a new pet
new_pet_id = cur.callfunc('pet_manager.add_pet',
      int,
      [pet_name, owner_id, pet_type])

print (new_pet_id)

4
```

OUT PARAMETERS

`OUT` parameters can be very handy when you need to pass back more than one piece of information. I have an `add_pet` function in the `pet_manager` PL/SQL package that

checks to see if the pet type I'm adding needs a license. The function returns the new ID, as before, in addition to a `yes` or `no` value through the `OUT` parameter.

```
/*
   * Add a new Pet return new id and if it needs a license.
   */

   FUNCTION add_pet (
      name_p           IN VARCHAR2,
      owner_id_p       IN NUMBER,
      pet_type_p       IN VARCHAR2,
      need_license_p   OUT VARCHAR2
) RETURN NUMBER IS
      new_pet_id   NUMBER;
BEGIN
      INSERT INTO lcs_pets (
         name,
         owner,
         type
      ) VALUES (
         name_p,
         owner_id_p,
         lower(pet_type_p)
      ) RETURNING id INTO new_pet_id;

      IF lower(pet_type_p) IN ('dog', 'cat') THEN
```

ADDITIONAL RESOURCES

The following resources provide deeper dives into cx_Oracle, Python, PL/SQL, and more.

- cx_Oracle documentation
- Getting started with Python
- Steven Feuerstein's blog
- Bryn Llewellyn's blog
- Oracle Dev Gym
- Oracle LiveSQL

If you have any problems using cx_Oracle, you can get help here.

```
      need_license_p := 'yes';
   ELSE
      need_license_p := 'no';
   END IF;

   COMMIT;
   RETURN new_pet_id;
END add_pet;
```

To work with the `OUT` parameter in Python, I add a string variable called `need_license`. It can be defined with `cursor.var(str)`. Then I add the new variable to the `values` array in the correct position. This works the same way when you're using OUT parameters with the `callproc` method.

To get the value from `need_license`, I call its `getvalue()` function. When I run the following code in my Python template, named `execute_function_out.py` for this example,

```
python3 execute_function_out.py
```

the code is successful and returns values for `new_pet_id` and `need_license`.

```
pet_name = 'Roger'
owner_id = 1
pet_type = 'dog'
need_license = cur.var(str)
```

```
# Add a new pet
new_pet_id = cur.callfunc('pet_manager.add_pet',
      int,
      [pet_name, owner_id, pet_type, need_license])

print ("New pet id: {}\nLicense needed: {}".format(new_pet_id, need_license.getvalue()))

New pet id: 5
License needed: yes
```

ACCEPT ARGUMENT VALUES

So far I've hardcoded the variable values in the Python code, and the methods are fairly simple, so there's a low chance of errors. But for most methods, I want to accept parameter values that can be passed into the Python code and then on to the PL/SQL functions. I modify the Python method to accept command-line arguments.

I need to import `sys` so that I can use `sys.argv[]` to grab the command-line arguments and assign them to the variables.

```
import cx_Oracle
import os
import sys

connectString = os.getenv('db_connect')
con = cx_Oracle.connect(connectString)
cur = con.cursor()
```

```
pet_name = sys.argv[1]
owner_id = sys.argv[2]
pet_type = sys.argv[3]
need_license = cur.var(str)

# Add a new pet
new_pet_id = cur.callfunc('pet_manager.add_pet',
      int,
      [pet_name, owner_id, pet_type, need_license])

print ("New pet id: {}\nLicense needed: {}".format(new_pet_id, need_license.getvalue()))
```

When I run the above code in my Python template, named `accept_input.py` for this example, and add a dog as input

```
python3 accept_input.py rover 2 dog
```

the code is successful and returns values for `new_pet_id` and `need_license`.

```
New pet id: 4
License needed: yes
```

Or if I run the same code and add a fish as input instead,

```
python3 accept_input.py roger 1 fish
```

the code is successful and returns values for `new_pet_id` and `need_license`.

```
New pet id: 4
License needed: no
```

PL/SQL EXCEPTIONS

Now that I'm accepting outside argument values, the chance that I'll eventually get errors with the above code is almost a certainty. If an error happens in the Python code, I can handle it as I normally would. But what if the PL/SQL code throws an error?

It's easy enough to test this. Make the same call as before to add a dog, but pass in a *string* for the second value (`2x`).

```
python3 accept_input.py rover 2x dog
```

It returns the following error:

```
Traceback (most recent call last):
  File "accept_input.py", line 22, in <module>
    [pet_name, owner_id, pet_type, need_license])
cx_Oracle.DatabaseError: ORA-06502: PL/SQL: numeric or value error:
character to number conversion error
ORA-06512: at line 1
```

I recommend that you handle errors as close to where they happen as you can. In this example, you could catch the error in the PL/SQL function and either handle it

or raise it. If you don't handle it in PL/SQL, it will be passed back to cx_Oracle, which will throw a cx_Oracle.DatabaseError. At that point, you can handle it as you would when any other error is thrown in your Python application.

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

DOWNLOAD cx_Oracle.

GET this article's code examples from GitHub.

READ the "Perform Basic CRUD Operations with cx_Oracle" series:

Part 1

Part 2

Part 3

Part 4