Echtzeit-Visualisierung von Twitter & Co.

Kai Donato und Oliver Lemm, MT AG

Dieser Artikel zeigt, wie der Bedarf an Visualisierungen im Datenbank-Umfeld mehr und mehr zunimmt. Viele Daten zu erheben und zu analysieren, ist das Eine – sie für den Menschen anschaulich und verständlich zu machen, das Andere.

Viele Unternehmen erheben Metadaten, um diese für ihre Zwecke zu analysieren und zu ihrem Vorteil zu nutzen. So sammeln Onlineshops Suchbegriffe und Zugriffszahlen, um gewisse Trends frühzeitig zu erkennen und darauf zu reagieren. Soziale Netzwerke verwenden diese gesammelten Daten, um passende Werbung und interessante Artikel zustellen zu können. Staatliche Institutionen nutzen große Datenmengen, um Straftaten aufzudecken oder sogar präventiv zu verhindern.

All diese Einsatzzwecke stimmen zumindest in dem Punkt überein, dass sie für den jeweiligen Einsatzzweck aufbereitet dargestellt werden müssen. Sobald die menschliche Komponente in diesem Analyse-Prozess auftritt, sind Daten in aggregierter Form und nicht selten auch visualisiert aufzubereiten.

Echtzeit-Visualisierung

Häufig sind nicht nur die Daten selbst von hohem Wert, sondern auch deren Aktualität. Was nützt ein Dashboard, das kritische Prozesse überwacht, wenn es nur die alten Daten vom Vortag anzeigt? Um den vollen Nutzen eines Dashboards auszureizen, müssen die Daten in Echtzeit übermittelt und verarbeitet werden.

Dieses Ziel haben in der Vergangenheit viele Entwickler gelöst, indem sie mit dem sogenannten „Polling“ arbeiteten. Dies ist nicht von Vorteil und es gibt eine wesentlich bessere Lösung für diese Problematik, wie das Beispiel der Echtzeit-Visualisierung zeigt.

Für dieses Beispiel wurde ein Online-Dienst ausgesucht, der seine Existenzberechtigung gerade aus der schnellen Interaktion von vielen Benutzern bezieht. Die Mikroblogging-Plattform Twitter ist bestens dafür geeignet, den Nutzen der Verarbeitung in Echtzeit darzustellen.

Um einen konkreten Use-Case zu generieren, wurde für das Projekt ein Rahmen gewählt, der sich auf eine bestimmte Themengruppe einschränkt. Für die anstehende Konferenz „APEXConnect 2016“ in Berlin existiert bereits ein sogenannter „Hashtag“, der eine Kurznachricht einem Thema zuordnet und sich von allen, die sich für dieses Thema interessieren, finden und abonnieren lässt *(siehe Abbildung 1)*.

Da Twitter die Tweets und auch die Antworten auf bestimmte Tweets standardmäßig in Listenform darstellt, ist es naheliegend, diese Tweets und Beziehungen zueinander in einer grafischen Form auszuwerten. Hierzu ist es notwendig, die Tweets und Beziehungen anhand der erhobenen Daten zu ermitteln und an eine geeignete Visualisierungs-Engine weiterzugeben. Anschließend werden die Daten aufbereitet und dargestellt.

Woher die Daten kommen

Um die Daten zu beziehen, ist von vornherein der Aspekt zu berücksichtigen, dass sie möglichst schnell nach der Veröffentlichung vorliegen. Dies lässt sich anhand des Streaming-API von Twitter optimal und vor allem in Echtzeit realisieren. Der Vorteil dabei ist, dass die Daten, sobald sie veröffentlicht wurden, direkt an einen lauschenden Server weitergeleitet werden und dort bereits wenige Sekunden später verarbeitet werden können. Sofern ein Server aufgesetzt und konfiguriert wurde, liefert Twitter die Daten in JSON-Form. *Listing 1* zeigt einen Auszug aus einer solchen Datenübertragung.

Anhand dieser Daten wird die weitere Verarbeitung vorgenommen. Beim Twitter-API ist zu beachten, dass es Einschränkungen bezüglich der Frequenz und Anzahl der Anfragen gibt. Bei dem von den Autoren hauptsächlich genutzten Streaming-API gibt es lediglich die Einschränkung der Anzahl an aktiven Verbindungen. Für die Visu-

Abbildung 1: Ansicht zweier Tweets zum Thema „#apexconn16“

alisierung von spezifischen Hashtags haben sie sich für die Variante „User-Streams" entschieden. Diese ist nach der Anmeldung als Entwickler und der Einrichtung einer Applikation kostenfrei. Im Beispiel ist ausschließlich auf die Anzahl der aktiven Streaming-Verbindungen zu achten. Um mit dem genannten NodeJS-Server nur eine aktive Verbindung zu benötigen, wird dieses Limit nicht überschritten.

Für die manuell initiierten Anfragen, die nach dem REST-Prinzip funktionieren, muss jedoch auf deren Anzahl geachtet werden. Für die Suche nach konkreten Begriffen werden 180 Anfragen pro 15 Minuten zugelassen und bei allen anderen Interaktionen mit dem REST-API dürfen 15 Anfragen pro 15 Minuten durchgeführt werden. Damit der NodeJS-Server sich als Stream-Endpunkt anmelden darf, benötigt dieser eine Kombination aus mehreren Sicherheitstoken:

- Consumer Key
- Consumer Secret
- Access Token Key
- Access Token Secret

Diese Sicherheitstoken werden beim Erstellen einer Applikation im Entwicklerbereich generiert und anschließend benötigt, um die Daten von Twitter abzurufen.

```
[{
  [...]
  "created_at": "Tue Jan 12 07:09:54 +0000 2016",
  "id": 686807297725698000,
  "id_str": "686807297725698048",
  "text": "Ich freue mich auf die #apexconn16 !",
  "source": "<a href=\"http://tapbots.com/software/tweetbot/mac\"
     rel=\"nofollow\">Tweetbot for Mac</a>",
  "truncated": false,
  "in_reply_to_status_id": null,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null,
  "in_reply_to_user_id_str": null,
  "in_reply_to_screen_name": null,
  [...]
}]
```

Listing 1

Mit den genannten Schlüsselpaaren authentifiziert sich der NodeJS-Server beim Twitter-API und hat fortan die Möglichkeit, auf diverse Daten zugreifen zu können *(siehe Abbildung 2)*.

Die Daten ohne merkliche Verzögerung verarbeiten

Nun profitiert man davon, das Twitter die Daten ohne Verzögerung liefert, steht jedoch vor dem Problem der verzögerungsfreien Weiterleitung an die Visualisierungs-Engine. An dieser Stelle haben sich die Autoren für ein serverseitiges JavaScript entschieden und einen NodeJS-Server eingerichtet, der permanent auf Meldungen von Twitter wartet und diese anschließend weiterleitet. Der Vorteil dabei ist, dass die Einrichtung des Servers nur einen minimalen Arbeitsaufwand verursacht. Mithilfe eines bereits entwickelten NPM-Pakets ist auch die Anbindung an Twitter innerhalb von kurzer Zeit und wenigen Code-Zeilen realisiert. Für die Weiterleitung an den Browser und somit an die Visualisierungs-Engine muss

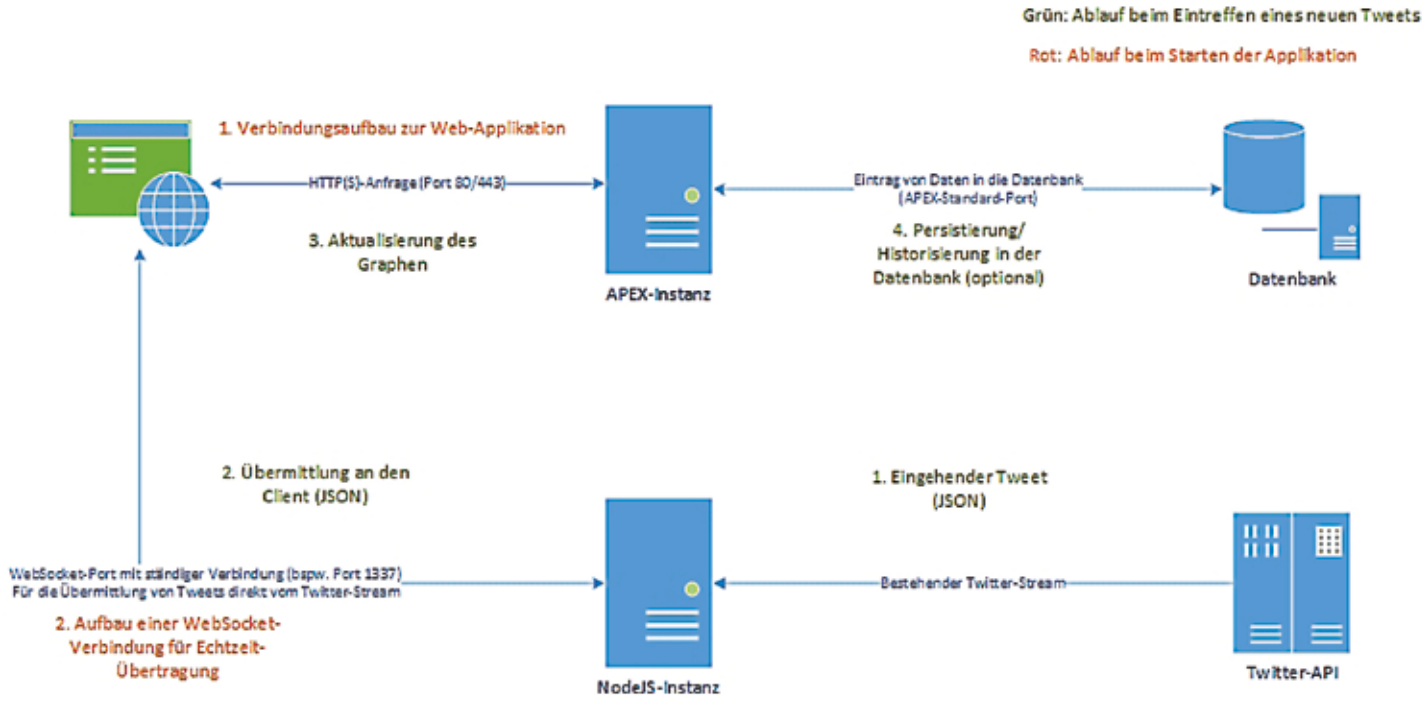

Abbildung 2: Schematische Darstellung des Ablaufs

hinsichtlich der unerwünschten Verzögerung auch auf eine Technologie gesetzt werden, die sich von dem genannten Polling-Verfahren unterscheidet.

WebSockets

Um die permanenten Anfragen des Client-Browsers zu unterbinden, stellt es sich als äußerst sinnvoll dar, die Datenübertragung von der Seite des Servers zu initiieren. Dies lässt sich mithilfe der in HTML5 eingeführten WebSocket-Technologie erreichen.

Der einmalige Aufbau einer WebSocket-Verbindung zwischen Server und Client ermöglicht es, jederzeit Daten in beide Richtungen zu transferieren und im Anschluss definierte Aktionen auszuführen. So gelangen die von Twitter übermittelten und von NodeJS aufbereiteten Daten über die WebSocket-Verbindung an den Client-Browser.

Um diesen Prozess zu realisieren, wurde im NodeJS-Server ein Stream-Endpunkt für das Twitter-API implementiert und in Form von Callback-Funktionen auf eingehende Daten reagiert. Durch die Non-Blocking-Eigenschaft von NodeJS ist es möglich, neben dem ständig aktiven Streaming-Endpunkt noch weitere Funktionalitäten zu realisieren. Dazu gehört unter anderem auch die manuelle Abfrage nach historischen Daten, die für die grafische Aufbereitung verwendet werden.

Graphen-Theorie

Um große Mengen an Daten aggregiert und verständlich darzustellen, eignen sich Graphen besonders gut. Im Vergleich zu tabellarisch komprimierten Daten, die sich meist auf die Darstellung von Zahlen beschränken, bieten Graphen mehrere Dimensionen zur Visualisierung. Selbst in der einfachsten Form des Graphen, bestehend aus Kanten und Knoten, ist eine Verbindung von zwei Punkten selbsterklärend als Beziehung zu erkennen. Nimmt man nun die Möglichkeit einer gerichteten Kante hinzu, kann man dadurch auch die Flussrichtung der Information erkennen. Weitere Eigenschaften können in der Kantenlänge beziehungsweise im Abstand zwischen den Knoten, in der Größe der Knoten sowie in unterschiedlichen Farben von Knoten abgebildet werden.

Die Transformation

Der maßgebliche Faktor in einer Visualisierung durch Graphen liegt in der Transformation der Grund-Daten. Diese stammen aus Twitter und sind in Form einer Liste mit Metadaten vorhanden, die zum Teil in Beziehung zueinander stehen. Um diese in einer Graphen-Struktur darzustellen, werden die Tweets zu einem bestimmten Thema (in Twitter als „Hashtag“ bezeichnet) gefiltert. Im Beispiel werden die Postings zur nächsten Apex-Connect-Konferenz mit dem Hashtag „#APEXConn16“ verwendet.

Neben dem Hashtag selber stellen die Eigenschaften des Benutzernamens, der Posting-ID und des Profilbilds die relevanten Daten dar, die für eine spätere Visualisierung benötigt werden. Dazu werden alle JSON-Objekte, die durch den WebSocket empfangen werden, hinsichtlich des Benutzernamens überprüft. Falls dieser nicht vorhanden ist, wird er als neues Objekt wiederum im JSON-Format für den Graphen gespeichert, wobei das Profilbild beziehungsweise die URL des Bilds später für die Darstellung des Knoten verwendet werden soll. Die Posting-ID wird dabei genutzt, um zu gewährleisten, dass ein bereits verarbeiteter Post nicht nochmals benutzt wird. *Listing 2* zeigt, wie das im JSON-Format aussieht.

Die Anforderungen

Für die Darstellung eines Graphen kommt ein auf JavaScript basierendes Framework zum Einsatz, um einen nahtlosen Ansatz im Browser zu gewährleisten. Es soll die Möglichkeit bieten, einen vorhandenen Graphen zur Laufzeit zu erweitern, ohne dass man die Bezugspunkte verliert. Im Vergleich zu einer Visualisierung von statischen Daten wird ein Graph nicht direkt mit allen Daten dargestellt, sondern es existieren anfangs keine oder nur eine kleine Menge an Daten.

Über die Zeit werden mithilfe des Streaming-API immer neue Postings übermittelt und der vorhandene Graph wächst so Stück für Stück. Bei diesen Anpassungen soll sich der Graph möglichst nur gering in der gesamten Darstellung verändern, damit ein Benutzer schnell erkennen kann, wo sich eine Veränderung ergeben hat, ohne den Rest der schon vorhandenen Darstellung aus den Augen zu verlieren.

```
{
  "nodes": [{
      "user_id": 689815827
  "image_url": "https://pbs.twimg.com/profile/images/..."
  "tweets": [{
      "tweet_id": 689815827479203800
  },{
      "tweet_id": 689815827479203801
  }]
  ]}
}
```

Listing 2

```
// if user is running mozilla then use it's built-in WebSocket
window.WebSocket = window.WebSocket || window.MozWebSocket;
var connection = new WebSocket('wss://mywebsocketserver.de:11111');
connection.onopen = function() {
  console.log('Connection established');
};
```

Listing 3

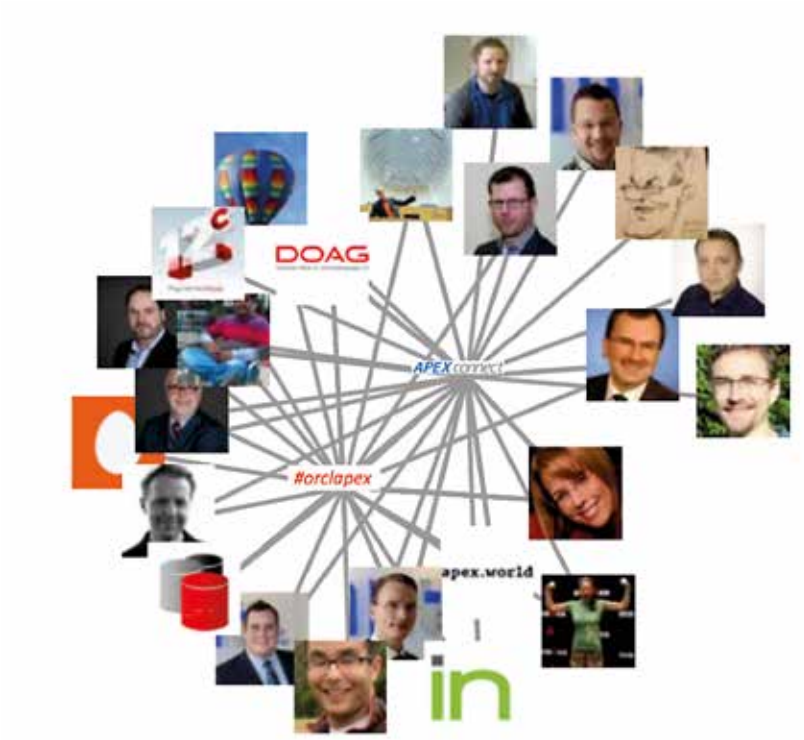

Abbildung 3: Darstellung des Graphen

Die Frameworks

Als ein mögliches Framework zur Graphen-Visualisierung bot sich D3 an, das eine Vielzahl von verschiedenen Graphen-Darstellungen unterstützt. Dabei stehen aber meist statisch zu visualisierende Daten zur Verfügung, die nicht noch zur Laufzeit angepasst werden müssen.

Nach weiteren Recherchen ist man auf das Visualisierungsframework VivaGraphJS gestoßen. Der technische Ansatz ist dem D3-Framework sehr ähnlich, da beide SVGs im Browser zur Darstellung der Graphen-Strukturen nutzen. VivaGraphJS wurde dabei aber speziell zur Visualisierung von Netzstrukturen entwickelt und bietet neben dem rein grafischen Ansatz auch Unterstützung für Gravitations-Effekte. Dies ist vor allem deshalb interessant, da ein Hinzufügen eines Knotens nicht eine komplett andere Struktur mit sich bringt, sondern der Graph sich beim Hinzufügen leicht anpasst und diese Anpassung durch den Gravitations-Effekt sichtbar ist. Eine Auflistung verschiedener Graph Libraries, die unter GitHub zur Verfügung stehen, kann man unterhalb des VivaGraphJS-Projekts auf GitHub einsehen.

Die Umsetzung

Für die Umsetzung wurde eine Apex-5-Anwendung erstellt, in der das VivaGraphJS-Framework als ZIP-Datei in die statischen Anwendungsdateien importiert wurde. In der Anwendung wurde neben der Login-Seite eine einzelne Seite erstellt, die die komplette Logik enthält. Durch eine Dynamic Action entsteht beim Seitenladen eine Verbindung zum NodeJS-Server per WebSocket-Verbindung *(siehe Listing 3)*.

Bei jeder Information (Tweet), die man über die Verbindung empfängt, wird eine Funktion „add_node_to_graph“ aufgerufen, wodurch sich der Graph jeweils um einen Knoten erweitert oder ein bestehender Knoten in der Größe wächst. Neben den Knoten, die eine Person darstellen, werden zentrale Inhalte, die als Hashtags referenziert sind, auch als Knoten dargestellt. In unserem Fall „#apexconn16“ oder „#orclapex“.

Um neben der Anpassung des Graphen einen neuen Tweet und die Person hervorzuheben, kommt zusätzlich eine kleine Bibliothek namens „PNotify“ zum Einsatz, die am oberen rechten Bildschirmrand eine Benachrichtigung einblendet, sobald neue Daten eintreffen *(siehe Abbildung 3)*.

Best Practices

Als eines der ersten Probleme kristallisierte sich die Verbindung zwischen Apex und NodeJS heraus, da der Apex-Server per HTTPS betrieben wird und die Kommunikation mit dem NodeJS-Server anfangs nur unverschlüsselt lief. Der ausschlaggebende Punkt war, dass die Applikation funktionierte, wenn alle Komponenten ohne SSL verbunden sind. Wenn der Benutzer die Applikation im Browser per HTTP aufruft und die Verbindung des WebSocket ebenfalls ohne SSL aufgenommen wurde, gab es keine Probleme. Erst wenn eine der genannten Verbindungen auf SSL-Verschlüsselung umgestellt wurde, verweigerte der Browser die Verbindung aufgrund von Sicherheitsrichtlinien. Um dieses Verhalten und somit auch die Sicherheit der gesamten Applikation zu gewährleisten, wurden der NodeJS-Server und somit auch die WebSocket-Verbindungen mit validen SSL-Zertifikaten versehen.

Bei Testläufen bestand ein weiteres Problem darin, dass oft genug Daten benötigt wurden, es aber aktuell keine Tweets gab. Hier wird man also fleißig in Twitter aktiv, um dies voranzutreiben. Um auch ohne Livedaten eine Visualisierung zu testen, wurde neben dem Streaming-API auch eine Abfrage über das REST-API implementiert.

Nach einigen Testläufen stellte man fest, dass hin und wieder einige Knoten dem Graphen hinzugefügt wurden, die mit keinem zentralen Knoten verknüpft waren und sich dabei vom Graph entfernten. Der Grund war, dass in den übermittelten JSON-Daten des Streaming-API die Groß- und Kleinschreibung mit übernommen wurde, aber im Graph die Ausdrücke „#APEXCONN16“ oder „#ORCLAPEX“ nur in Großbuchstaben als Knoten existierten. Dieses Problem wurde durch eine Anpassung des Verarbeitungsschritts auf Seiten des NodeJS-Servers behoben. Alle Hashtags werden nur noch in Großbuchstaben verarbeitet.

Ein weiteres Problem bei solchen dynamischen Graphen ist die Größe beziehungsweise die Skalierung des Graphen.

Da anfangs nicht bekannt ist, wie groß der Graph insgesamt wird, und ein automatisches Anpassen der Größe nicht vorhanden ist, sind die Graphen nicht ohne händisches Anpassen perfekt zu nutzen.

Alleinstellungsmerkmal

Das Thema bietet eine Menge interessanter Technologien, aber somit auch eine Menge von Problemen, die dabei auftreten können. Vor allem der Einstieg in die Visualisierung von Echtzeitdaten ist mit wesentlich mehr Anforderungen verbunden als eine Visualisierung von statischen Daten. Der Aufwand lohnt sich jedoch, gerade bei großen Datenmengen bietet sich eine Graphen-Struktur durch die Aggregation der Daten an, um den Überblick zu behalten. Zusätzlich wird durch die Echtzeit-Verarbeitung der Twitter-Daten und die Aktualisierung des Graphen durch das performante VivaGraphJS-Framework eine Livedatenvisualisierung ohne Polling-Ansatz erreicht.

Vergleichbare Ansätze basierend auf dem Twitter-Streaming-API, die eine Graphen-Struktur ohne Beschränkungen in der Anzahl der verknüpften Personen darstellen, sind bisher nicht vorhanden. Alles Weitere zu dem Thema wird es auf der diesjährigen Apex Connect in Berlin geben.

Weitere Informationen

- D3, JavaScript Graph Library: *https://d3js.org*
- VivaGraphJS, JavaScript Graph Library: *https://github.com/anvaka/VivaGraphJS*
- Graph Libraries, diverse unter GitHub vorhandene JavaScript-Frameworks zur Visualisierung von Graphen: *https://github.com/anvaka/graph-drawing-libraries*
- Pnotify, JavaScript Framework zur Darstellung von Benachrichtigungen: *https://github.com/sciactive/pnotify*

Kai Donato
kai.donato@mt-ag.com

Oliver Lemm
oliver.lemm@mt-ag.com

Schließung des Oracle-Software-Produkt-Supports in Deutschland steht unmittelbar bevor

DOAG Online

Die bisherigen Gerüchte um den Stellenabbau im europäischen Support sollen sich schon bald bewahrheiten. Mario Kowalski, Customer Support Country Leader bei Oracle Deutschland, bestätigte im Gespräch mit der DOAG die bevorstehende Schließung des Oracle-Software-Produkt-Supports in Deutschland zum 31. März 2016. Er versprach: Sollten DOAG-Mitgliedern dadurch Probleme entstehen, stünde er als Ansprechpartner zur Verfügung.

In Deutschland sind rund 150 Mitarbeiter von der Stellenstreichung betroffen. Wie Oracle bestätigte, betrifft der Stellenabbau im Oracle-Support jedoch ganz Europa. Neben Deutschland trifft es auch die Supportzentren in Großbritannien, den Niederlanden und Frankreich. Insgesamt sollen 450 Stellen wegfallen. Ziel der Maßnahme ist die Verlegung des Supports nach Rumänien. Dort soll er laut Oracle quantitativ größer aufgestellt werden, genaue Mitarbeiterzahlen nannte das Unternehmen jedoch bisher nicht. Die Kunden stellt diese Entwicklung vor viele offene Fragen hinsichtlich der Gestaltung der Übergangszeit und der zu erwartenden Auswirkungen auf die Support-Qualität.

Die DOAG befürchtet hier einen zumindest kurzfristigen Qualitätsverlust in der Übergangsphase und initiierte daher ein Treffen mit Oracle, um über die Entscheidung und deren Auswirkungen zu sprechen. „Wichtig für die Kunden ist vor allem, ob Oracle in der Lage sein wird, die Qualität des Supports sowohl qualitativ als auch quantitativ aufrecht zu erhalten“, so Dr. Dietmar Neugebauer, Vorstandsvorsitzender der DOAG. „Entscheidend wird dabei sein, wie der Übergang Ende März reibungslos durchgeführt werden kann. Nichtsdestotrotz bedauert die DOAG es sehr, dass Oracle Deutschland sich von vielen langjährigen Mitarbeitern verabschiedet und dadurch sehr viel Know-how in Deutschland verloren geht.

Im Gespräch sicherte der Oracle-Supportverantwortliche Kowalski der DOAG zu, bei Support-Problemen den DOAG-Mitgliedern als Ansprechpartner zur Verfügung zu stehen. Christian Trieb, Leiter des DOAG Competence Center Support und der Datenbank Community, begrüßte im Namen der DOAG den angestrebten Dialog und hofft nun auf eine möglichst komplikationsfreie Übergangszeit. „Bezüglich der Support-Thematik wird die DOAG auch weiterhin eng mit Oracle in Kontakt stehen, denn erst langfristig wird sich herausstellen, wie sich die Servicequalität entwickelt und welche Rolle mögliche Schwierigkeiten in der sprachlichen Verständigung dabei spielen“, so Trieb weiter. Wie die Kunden mit dem neuen Software-Produkt-Support in Rumänien zurechtkommen, wird auch die im Herbst dieses Jahres erneut geplante DOAG-Umfrage zur Qualität des Oracle-Supports zeigen.