Oracle Big Data SQL bringt Data Warehouse und Big Data zusammen

Alfred Schlaucher, ORACLE Deutschland B.V. & Co. KG

Die Big-Data-Diskussion ist gereift – auch bei uns in Deutschland, wo wir meistens etwas Zeit benötigen, um neue Entwicklungen aus den USA zu adaptieren. Diese Phase der letzten vier Jahre war eine Mischung von skeptischen Bewertungen.

Die einen sagten: „Das, was da kommt, ist ja nur eine Erweiterung für das Data Warehouse und wird von uns schon längst gemacht". Andere starteten unüberlegte „Sturzprojekte", in denen plötzlich nur noch „alles anders gemacht werden" muss, „keine relationalen Datenbanken mehr", „auch kein Warehouse mehr", nur noch „Hadoop auf selbst montierten Cluster-Rechnern" und aus sich selbst heraus erklärende Analysen. Wieder andere suchen (zum Teil noch heute) die „Use Cases" für die neue Technik in ihrem Unternehmen.

In manchen Unternehmen hat sich der Nebel gelichtet, rationale Überlegungen sind eingetreten. Nüchtern stellt man fest, dass das Wesentliche in der Diskussion nicht die neue Technik ist, sondern dass ein innovatives Denken einsetzen muss, wenn man sich fragt, was man alles mit gesammelten Daten anfangen kann. Viele entdecken jetzt erst – angeleitet durch die Big-Data-Diskussion –, dass sie zwar seit vielen Jahren operative Daten in ihrem Data Warehouse sammeln, aber nicht das Optimale aus diesen Daten herausgeholt haben. Viele betreiben ein „rückwärts gerichtetes Analysieren", letztlich ein operativ ausgerichtetes Berichtswesen über Geschäftserfolg und Trends in den gesammelten und internen Unternehmensdaten.

Charakteristisch ist diese kleine Geschichte eines Handelsunternehmens aus dem Oracle-Warehouse-Umfeld, wie sie so in diesen Tagen häufig erlebt werden kann: Bis vor einem Jahr präsentierten Hersteller ihre Big-Data-Lösungen reihenweise in unterschiedlichen Abteilungen des Unternehmens. Aus Fachabteilung und Management kamen die Rufe: „Wir müssen was tun, alle machen was". Die IT-Leitung reagierte prompt, budgetierte und plante Hardware-Komponenten für einen Big-Data-Cluster. „Bitte, liebe Fachanwender, demnächst kann es mit dem neuen Cluster losgehen. Dürfen wir fragen, was Ihr damit machen wollt?" Und prompt der Rückzieher aus der Fachabteilung: „Halt, halt … wir wissen doch noch gar nicht, was wir damit wollen, wir müssen erst unsere Use Cases entwickeln". Das Beschaffungsprojekt war gestoppt.

Heute, ein Jahr später, ist man in dem Unternehmen einen Schritt weiter. Die Fachabteilung ist dabei, ihre Hausaufgaben zu machen. die Mitarbeiter haben sich die bestehenden Kundendaten genauer angeschaut und beginnen über Data-Mining-Verfahren, Segmentierungsvarianten zu erkunden. Sie durchforsten damit das bestehende Kundendaten-Material und das Kaufverhalten der Kunden in den letzten Jahren. Sie erweitern dies sukzessiv um zusätzliche externe Daten aus Medien, in denen sich ihre Kunden umtreiben, und reichern die Analyse noch mit Weblog-Daten über das Click-Verhalten im Webauftritt und auch (man höre) mit Wetterdaten an.

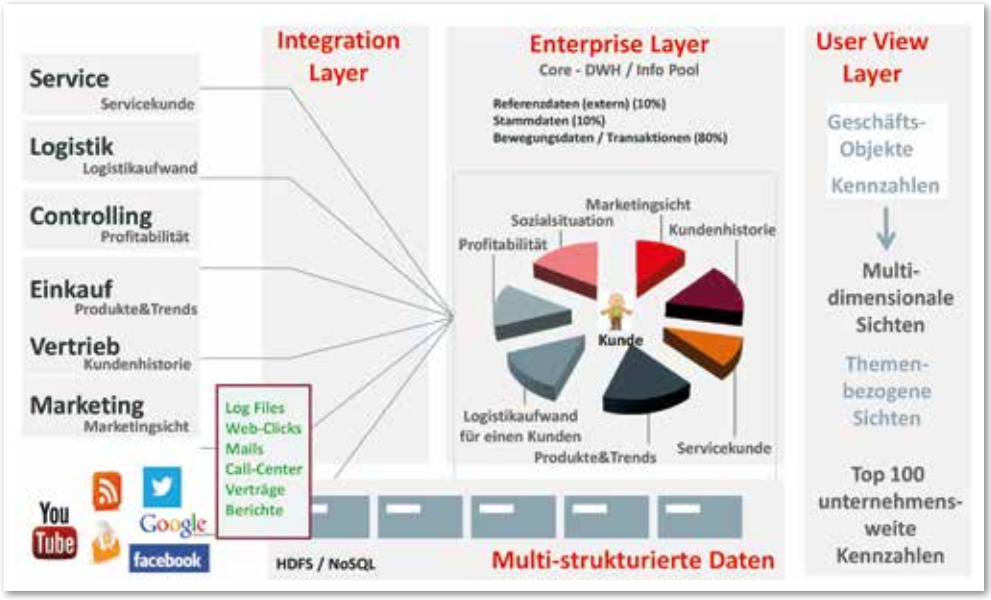

Abbildung 1: Der große Überblick

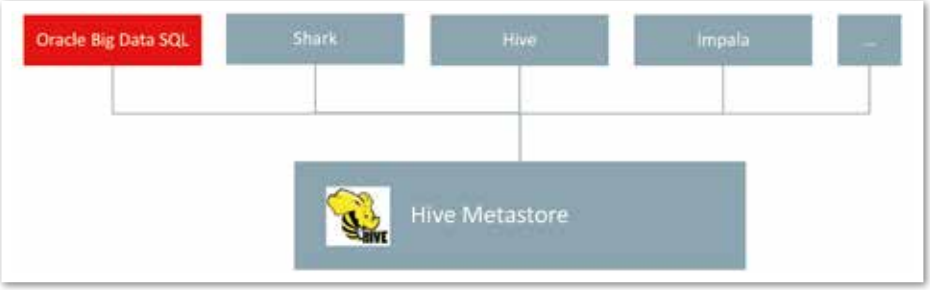

Abbildung 2: Hive Metastore

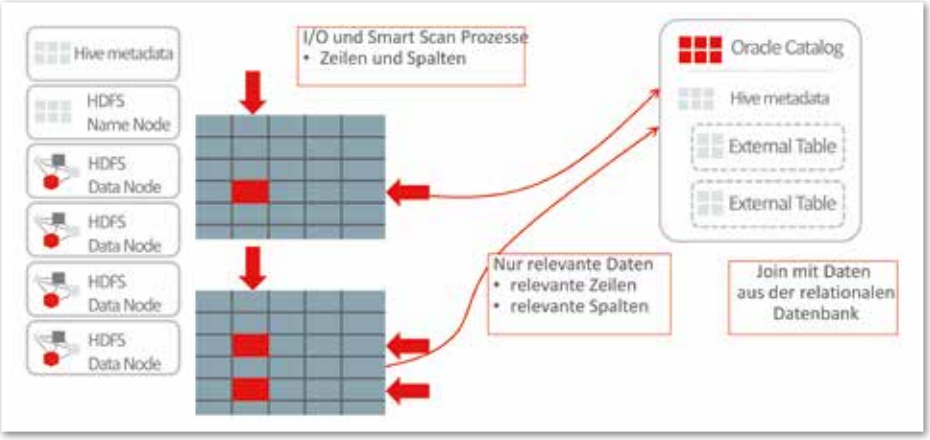

Abbildung 3: Die Analyse erfolgt in der Oracle-Datenbank

Eigentlich ganz klassisch. Nichts anderes haben wir vor 15 Jahren bereits gemacht, indem wir alle bekannten Kundenmerkmale in eine breite Tabelle gepackt und Data-Mining-Algorithmen angewendet haben. Nur – wer hat das damals wirklich gemacht? Es waren wenige Firmen mit bestimmten Geschäftsmodellen, die beispielsweise über Loyalty-Karten verfügten, oder spezialisierte Firmen mit besonderen Verfahren im Umgang mit Unternehmensdaten. Unsere Beispielfirma jedenfalls nicht. Das ist das neue Positive an der Entwicklung: Aufgeweckt durch die Big-Data-Diskussion beschäftigt man sich jetzt in einer bewussten Weise mit den eigenen Daten und ergänzt sie punktuell um externe Informationen. Technik und Tools kommen erst in zweiter Linie, nachdem man erkannt hat, was man will. Erst in dieser Phase lohnt die Beschäftigung mit Technologie. Jetzt kann man bewerten, was man wirklich benötigt *(siehe Abbildung 1)*.

Klassische Kunden- und Transaktionsdaten müssen bei dem oben beschriebenen Szenario um unspezifische, lückenhafte Daten ergänzt werden, die von außerhalb des Unternehmens beziehungsweise aus technischen Prozessen wie der Internet-Kommunikation kommen. Daraus lassen sich Anforderungen an eine technische Infrastruktur ableiten:

- Big-Data-Daten und klassische Daten müssen verknüpfbar sein
- Die physikalische Speicherung sollte technisch zusammenhängend funktionieren, also über eine zusammenhängende Infrastruktur bedienbar sein
- Big-Data-Daten und klassische Warehouse-Daten sollten zusammenhängend abfragbar beziehungsweise analysierbar sein
- Die Benutzer sollten sich mit den ihnen zur Verfügung stehenden Mitteln helfen können, also die ihnen bekannten Tools und ihr bestehendes Know-how weiter nutzen können
- Es sollte ein durchgängiges Security-Verfahren nutzbar sein
- Bekannte und bewährte Tools sollten weiter genutzt werden können und es sollte überhaupt eine ausreichende Zahl spezialisierter und ausgereifter Tools zur Verfügung stehen

SQL – ein bevorzugtes Zugriffsmittel

Man könnte die Liste noch um den Punkt „SQL-Fähigkeit" erweitern. Doch damit hätte man schon alle anderen Punkte abgedeckt. SQL ist das unangefochtene und unumstrittene eine Mittel, mit dem nahezu alle Forderungen erfüllbar sind. Daher verwundert es auch nicht, dass sich SQL auch in der Hadoop-Welt als Abfragemedium durchgesetzt hat.

Die meisten neu entwickelten Abfrage-Werkzeuge setzen auf SQL. SQL ersetzt an dieser Stelle auch die Verfahren der ersten Stunde wie MapReduce. Zu groß waren die Hürden der Java-Entwicklung von Abfragen und Algorithmen auf Hadoop-Daten. Die Software-Anbieter haben erkannt, wie hinderlich dieser Programmier- und Java-Aspekt bei der breiten Einführung von Hadoop-Technik in den IT-Betrieb vieler Firmen ist. Sie liefern mittlerweile SQL-basierte Werkzeuge, mit denen sich die archaische Cluster-Physik der neuen Datenhaltungen von HDFS bis NoSQL im Hintergrund kaschieren lässt. Sie machen also das, was Datenbankhersteller wie Oracle schon immer gemacht haben. In dieser Situation kommt Big Data SQL von Oracle gerade richtig. Die neue Funktionalität des Datenbank-Release 12.1.0.2 setzt auf der Argumentation von oben auf:

- Sie macht alle Datenarten (HDFS und DB-Tabellen) für alle Datenbank-Anwendungen verfügbar
- Sie nutzt die volle Bandbreite der Oracle SQL Query Language
- Sie stellt alle bewährten Security-Features von Oracle 12*c* auch für Hadoop-Daten bereit
- Daten müssen nicht zwischen dem Hadoop-Cluster und der Oracle-Datenbank hin und her kopiert werden, Daten verbleiben also in ihrer jeweiligen Physik und sind dennoch zusammen analysierbar
- Abfragen mit sehr hoher Performance
- Das bestehende (SQL-) Wissen der Anwender wird wiederverwendet
- Zum Einsatz kommen dennoch die aktuellsten Hadoop-Neuerungen

Die Technik

Hadoop zeichnet sich dadurch aus, dass große Datenmengen in einer zum Speicherungszeitpunkt nicht relevanten inne-

ren Struktur als Dateien in einem Dateiverzeichnis abgelegt werden, ohne dass man sich dabei an Rechnergrenzen orientieren muss. Das Hadoop-Distributed-File-System (HDFS) sorgt für die Ablage der Daten in relativ großen Datenblöcken (zum Beispiel 200 MB) über Rechnergrenzen verteilt. So gelingt es, einen Verbund von vielen Rechnern (Cluster) physikalisch über ein Netzwerk zusammenzukoppeln. Das Dateisystem kann dabei beliebig wachsen. Da es sich über viele Rechner (Server) erstreckt, können zum späteren Abfragezeitpunkt auch mehrere Rechner parallel für das Lesen der Daten herangezogen werden, was aufgrund der Menge der Rechner und der hohen Verteilung zu einer höheren Performance führt. Hadoop legt zur Sicherheit redundante Blöcke auf mehreren Servern ab.

Außer der Möglichkeit, quasi unendlich große Datenbestände auf beliebigen (eventuell kostengünstigen) Servern zu speichern und diese mit einer hohen Parallelisierung abzufragen, ist bis jetzt allerdings noch nicht viel gewonnen. Abfragen, wie wir sie aus einer relationalen Datenbank kennen, sind noch nicht möglich. In dieser rudimentären Form muss man jeden einzelnen Datenblock beispielsweise mit einem in Java geschriebenen Map-Reduce lesen und die Logik für das Verstehen der Daten individuell programmieren.

Um diese sperrige Vorgehensweise zu mildern, wurde unter anderem Hive als Hadoop-Pendant zu relationalen Datenbanken entwickelt. Damit kann man eine Struktur in den HDFS-Dateien beschreiben, sodass sie in einer gewohnten Satz- beziehungsweise Feldstruktur direkt abgreifbar ist. Bei dieser Beschreibung bedient man sich natürlich immer wiederkehrender strukturierender Zeichenfolgen in den Daten, ob Separatoren-Zeichen wie Komma oder Semikolon oder einfach nur definierter Satzende-Zeichen. Weit hergeholt ist das nicht, denn jede Log-Datei eines Webservers ist zum Beispiel nach Sätzen und Feldlängen strukturiert, auch wenn es zunächst nur Buchstabenfolgen sind. In jeder Ansammlung von E-Mails lassen sich der Beginn und das Ende jeder einzelnen E-Mail identifizieren und innerhalb derer auch Sender, Empfänger, Tag, Uhrzeit etc. feststellen. Den eigentlichen E-Mail-Text betrachtet man schließlich als Einheit (ein Feld).

Solche Metadaten speichert man in der Hadoop-Welt im sogenannten „Hive Metastore“ *(siehe Abbildung 2)*. Diese Vorgehensweise ist über Apache standardisiert. Die Hersteller von Hadoop-Distributionen wie Cloudera haben dieses Konzept in ihren Lösungen umgesetzt. Der sogenannte „Local Metastore“ wird dabei über eine zusätzliche kleine relationale Datenbank (wie MySQL) umgesetzt. Über Java Database Connectivity (JDBC) schreiben die Werkzeuge der Hersteller (etwa Cloudera oder Hortonworks) die definierten Metadaten in diese Datenbank. Damit liegen sie für jeden offen bereit.

Auch Oracle nutzt diese generischen Metadaten bei dem neuen SQL-basierten Zugriff. Um zusammenhängende Analysen zu ermöglichen, muss man Tabellen- und Spalten-Namen zusammenhängend an einem Ort vorhalten. Das ist der „Oracle-Datenbank-Katalog“, das Dictionary. Um diesen mit Beschreibungen von Hadoop-Objekten zu füllen, greift Oracle einmalig in den Hive Metastore. Das geschieht beispielsweise mit dem SQL Developer und dem darin enthaltenen ER-Modeler. Man wählt die „Reverse Engineering“-Option und liest den Hive Metastore aus. Der ER-Modeler visualisiert die gefundenen Hive-Objekte grafisch in einer Tabellendarstellung. Damit weiß man jetzt, in welcher Form die Objekte im HDFS vorliegen.

Zugriff auf die Hadoop-Daten ohne Hive

Für den eigentlichen Lesevorgang der Daten ist Hive nicht mehr erforderlich. Dafür hat Oracle im Rahmen der Big-Data-SQL-Funktionalität die altbekannten External Tables weiterentwickelt. Konnten diese bis zum Release 12.1.0.1 nur Flat Files beziehungsweise Datapump-Objekte lesen, können sie jetzt mit 12.1.0.2 auch Hive-Tabellen und Hadoop-Datenbestände (HDFS-Files) erfassen. Die External Table lässt man sich aus dem ER-Modeler heraus als Data Definition Language (DDL) generieren *(siehe Listing 1)*.

Spannend wird es, wenn man jetzt über die External Table zugreift. Die Zeit der Map-Reduce-Programme ist vorbei. Oracle hat seit der Einführung von Exa-

data vor sechs Jahren gute Erfahrungen mit externen, also außerhalb des eigentlichen Datenbank-Servers laufenden Scan-Prozessen gemacht. Diese Technologie wurde in die Hadoop-Welt übertragen. Auf jedem Hadoop-Cluster-Knoten ist jetzt eine Oracle-Scan-Software installiert, sodass auf jedem Cluster-Knoten Scan-Prozesse laufen können.

Über den Nameserver des Hadoop-Clusters ist bekannt, welche Blöcke auf welchen Cluster-Knoten liegen. Setzt der Benutzer jetzt eine SQL-Abfrage über die oben beschriebene External Table ab, erfolgt zunächst ein Zugriff auf diesen Nameserver und schließlich auf die so gefundenen Cluster-Knoten. Die vorher initialisierten Oracle-Scan-Prozesse lesen jetzt die einzelnen Dateiblöcke und filtern bereits beim Zugriff auf die Daten die gewünschten Informationen heraus. Das Filtern der Daten findet also bereits während des Zugriffs auf die Daten statt. Nur die passenden Daten (gesuchte Tabellen beziehungsweise Spalten oder Werte innerhalb der Spalten) liefert der Leseprozess an die aufrufende Stelle zurück.

Paralleles Lesen von HDFS-Daten

Aus der vorherigen Beschreibung erkennt man bereits, dass die Abfrage parallel erfolgt. Oracle hat dazu das External-Table-Verfahren parallelisiert. Während die ursprüngliche External Table etwa CSV-Dateien nur sequenziell, also in einem einzigen Prozess lesen konnte, kann die Hive- beziehungsweise Hadoop-Variante der External Tables einen maximalen Parallelitätsgrad in der Menge der zu lesenden Blöcke erreichen. Ist beispielsweise eine Datei im HDFS in 50 Blöcke à 200 MB zerlegt, starten auf dem Hadoop-Cluster 50 parallele Leseprozesse. Entsprechend hoch ist die Verarbeitungs-Performance.

Alles andere, was jetzt noch mit den Daten geschehen kann, erfolgt innerhalb des Oracle-Datenbank-Servers, egal ob aggregiert und gruppiert wird, ob analytische Funktionen arbeiten, eine Pattern-Analyse läuft oder ein Datenbank-internes R-Skript seine Arbeit erledigt. Sobald die Scan-Prozesse die bereits gefilterten Daten an den Master-Prozess zurückliefern, wirkt die bekannte und volle Funktionalität der Oracle-Datenbank *(siehe Abbildung 3)*.

Aktuell ist dieses Verfahren noch an die Big Data Appliance und Exadata gebunden. Die Oracle-Software für die Scan-Prozesse ist auf dem Hadoop-Cluster vorinstalliert und sie kann nur aus einer Datenbank heraus aufgerufen werden, die sich auf einer Exadata befindet. Ob sich diese Abhängigkeit noch ändert, bleibt abzuwarten. Zu wünschen wäre es, damit sich die Technik weiter durchsetzt. Während man die Abhängigkeit von der Exadata-Maschine nicht ganz verstehen kann, ist der Bezug auf die Big Data Appliance (BDA) durchaus sinnvoll, denn der Konfigurationsaufwand eines Hadoop-Clusters sollte nicht unterschätzt werden. Es wurden schon Projekte beobachtet, die mehr als sechs Monate mit der Bereitstel-

```
create table Web_Log
( wl_rec_id      number(10,0)
, wl_ip_Adr      char(32)
, wl_dns         char(24)
, wl_start_date  char(24)
, wl_end_date    char(24)
, wl_ses_id      number(14,0)
, wl_status      char(10)
)
organization external (
TYPE ORACLE_HIVE
DEFAULT DIRECTORY DEFAULT_DIR
ACCESS PARAMETERS
   (com.oracle.bigdata.cluster hadoop_cl_1)
LOCATION ('hive://weblog_address')
)
```

Listing 1

```
create table Web_Log
( id      raw (16) not null,
   date_loaded TIMESTAMP WITH TIME ZONE,
   Log_Record VARCHAR2(1000)
   CONSTRAINT log_ensure_json CHECK (Log_Record IS JSON))
```

Listing 2

```
INSERT INTO Web_Log
   VALUES (SYS_GUID(),
       SYSTIMESTAMP,
       ,{"wl_rec_id"         : 1600,
           "wl_ip_Adr"       : "168.192.1.10",
           "wl_dns"          : "MP-AM5643",
           "wl_start_date"   : "11-08-2014:23:21",
       "wl_end_date"         : "11-08-2014:25:18",
           "wl_ses_id"       : "77763576423",
       "wl_Ses_anz_sec"      :   10,
           "wl_status"       : "ACK"}');
```

Listing 3

```
select avg(log.Log_Record.wl_ses_anz_sec),
   log.Log_Record.wl_ip_Adr
   from Web_log log
   where substr(log.Log_Record.wl_ip_Adr,1,4) like ,"168'
   group by log.Log_Record.wl_ip_Adr;
```

Listing 4

lung lauffähiger Hardware beschäftigt waren. Anschließend hatte man immer noch kein ausfallsicheres System und keine wirksamen Security-Maßnahmen.

Multi-strukturierte Daten mit JSON

JSON hat sich in den letzten Jahren zu einem universellen Nachfolge-File-Format für XML-Daten entwickelt, wenn es um den Austausch von Daten geht beziehungsweise Systeme über ein standardisiertes Verfahren miteinander kommunizieren. Das JSON-Format benötigt weniger Speicherplatz als XML und kann damit in der Verarbeitung performanter sein. Gerade weil es nicht so extrem universell angelegt ist wie XML, eignet es sich für programmierte Zugriffe oder auch Zugriffe mit SQL besser als XML, weil weniger Overhead zu berücksichtigen ist.

Im Kontext von Big Data kommt noch ein besonderer Aspekt hinzu: Wenn man in der Big-Data-Diskussion von unstrukturierten Daten spricht, trifft dies die Problematik nicht ganz. Viel häufiger kommen komplexe Strukturen beziehungsweise uneinheitliche Strukturen vor. Aus der relationalen Welt kommend, denkt man oft in Satz-Strukturen mit der gleichen Anzahl von Feldern pro Satz-Art. Aber was ist, wenn es sich zwar um gleiche Satz-Arten handelt, aber die Anzahl der Felder von Satz zu Satz unterschiedlich ist oder innerhalb eines Feldes mehrere Werte existieren?

So beinhalten zum Beispiel die Daten für einen Haushalt Personendaten für eine n-Anzahl Erwachsener und eine n-Anzahl Kinder. In einer relationalen Struktur mit einer fixen Anzahl von Spalten pro Satz ist das nicht darstellbar. Wir würden mehrere Satzarten (beziehungsweise Tabellen) und Beziehungen definieren. Das Objekt „Haushalt" ist damit nicht zusammenhängend abgelegt. Mit JSON kann man solche Sachverhalte als zusammenhängendes Objekt speichern. Wenn die Anzahl der Haushaltsmitglieder variiert, dann verändert sich die innere Struktur des Haushalts-Objekts. Bei der Auflistung der Kinder wird etwa ein in dem Objekt eingebettetes Array mit entsprechenden Array-Plätzen angelegt.

Für diese veränderliche Anzahl von Kindern wäre eine kleine Zusatz-Tabelle im relationalen Modell noch akzeptabel. Haushaltsstrukturen können jedoch komplizierter werden und sie können sich über einen Zeitverlauf hinweg verändern, in dem neue Merkmale pro Haushalt erfasst werden müssen, an die man beim Design des ursprünglichen Modells noch nicht gedacht hatte: Es werden über Monate hinweg Gastkinder aufgenommen, es gibt Patchwork-Familien mit unterschiedlichen Elternteilen und nicht aus dieser Familie stammenden Kindern, es gibt reine Senioren-Wohngemeinschaften etc. Man spürt, wie klassische Datenmodellierung hier zu einem Dauerprojekt werden kann. In JSON gibt es für solche Objekte keine fest definierte Struktur. Die innere Struktur der gespeicherten Objekte kann von Satz zu Satz variieren. Mehr dazu unter *„http://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6287"*.

Seit dem Datenbank-Release 12.1.0.2 lassen sich JSON-Objekte auch in der Datenbank ablegen und mit SQL abfragen. Die Log-File-Information des oben dargestellten Beispiels könnte man in einer einfachen JSON-Tabelle ablegen *(siehe Listing 2)*. *Listing 3* zeigt den „INSERT" eines Datensatzes.

Man erkennt in dem INSERT-Statement die innere Struktur der Spalte „Log_Record". Dieses Beispiel ist bewusst einfach gewählt, um die Machart zu verstehen. Eines dieser inneren „(wl_)" -Felder könnte jetzt selbst noch einmal ein Objekt oder eine Auflistung (Array) sein. Diese Informationen sind mit SQL abfragbar, wie in diesem Beispiel, bei dem die durchschnittliche Session-Dauer für eine bestimmte Gruppe von IP-Adressen gesucht wird *(siehe Listing 4)*.

Damit können multi-strukturierte Informationen neben den traditionellen relationalen Daten in einem Datenbank-Medium liegen. Aber Achtung: Diese Option sollte nur dann gewählt werden, wenn es sich wirklich um sich stark ändernde Strukturen handelt. Ein Platz- und Performance-Vergleich des oben gezeigten „WEB_LOG"-Beispiels mit einer relationalen und fest definierten Tabellenstruktur zeigt große Ressourcen-Unterschiede *(siehe Tabelle 1)*.

An dieser Stelle finden wir wieder ein Argument für Big Data. Gerade bei unstrukturierten oder mit unterschiedlichen und wechselnden Strukturen behafteten Daten lohnt sich eine Hadoop-Infrastruktur. Denn wächst die Menge der JSON-Daten an und werden die Strukturen wesentlich komplexer als in dem oben gezeigten Beispiel, so verschlechtern sich die Ressourcen-Werte noch mehr im Vergleich zu einer strukturierten relationalen Datenhaltung. Man wird überlegen, diese Datenbestände in ein HDFS-System zu legen, und dann, wie oben beschrieben, mit Big Data SQL zugreifen.

Man kann JSON also als Übergangstechnologie betrachten, wenn man sich so lange wie möglich in der relationalen Datenbankwelt bewegen und den Zeitpunkt, an dem man einen Hadoop-Cluster aufbauen muss, möglichst weit nach hinten schieben will. Den Nutzen der Hadoop-Philosophie kann man dann auch schon ohne Hadoop-Cluster erfahren.

Kriterium	JSON-Format	Tabellen-Format
Anz. Zeilen	1.000.000	1.000.000
AVG_ROW_LEN	336	98
Größe in GB	0.337	0.121
Query-Zeit (Bsp. Oben)	3,2 Sec	0,2 Sec

Tabelle 1

Alfred Schlaucher
alfred.schlaucher@oracle.com