Abstract Oberseminar Datenbanksysteme
Anfragesprachen für Big Data

Ronny Zingler

25. Juni 2018

Inhaltsverzeichnis

Verzeichnisse

Abkürzungsverzeichnis

JSON JavaScript Object Notation

SQL Structured Query Language

ADM Asterix Data Model

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

„There were 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days“. Dieses Zitat des Google Mitarbeiters Eric Schmidt aus dem Jahre 2010 verdeutlicht die Relevanz des Themas Big Data.
Durch die ständige Generierung immenser Datenmengen, findet ein Umdenken in Sachen Persistierung statt. Neben den gebräuchlichen relationalen Datenbanksystemen halten sogenannte NOSQL Systeme Einzug. Diese lassen sich in folgende vier Kategorien unterteilen:

- Key-Value Stores (Redis, Oracle Coherence, memcached, Sanssouci)
- Dokumentenbasierte Datenbanken (Lotus Note, CouchDb, MongoDB)
- Wide-Column Stores (Cassandra, Hypertable, SimpleDB, BigTable)
- Graph Datenbanken (Neo4j)

Generell zeichnen sich NOSQL Datenbanksysteme durch die Möglichkeit der Speicherung von schwach- bis gar nicht strukturierten Daten und der hohen horizontalen Skalierbarkeit aus. Jedoch bringen diese Vorteile auch Nachteile mit sich. Durch die Schemafreiheit wird eine strukturierte Abfrage der Daten erschwert. Während relationale Datenbanksysteme mittels Structured Query Language (SQL) komfortabel angefragt werden können, vermisst man eine einheitliche Anfragesprache für NOSQL Datenbanken.[9] [8]
Diese Ausarbeitung soll verschiedene Möglichkeiten zur Anfrage von Big Data zeigen, welches meist auf JavaScript Object Notation (JSON) basiert. In einem ersten Abschnitt soll ein Vorschlag zur Integration von JSON in SQL behandelt werden, gefolgt von einer Erläuterung der JSON Abfragesprache JSONiq. Im dritten Abschnitt wird die Abfragesprache von AsterixDB SQL++ vorgestellt. Bevor versucht wird die Frage „gibt es ein SQL für NOSQL“im Fazit zu beantworten, wird Oracle Big Data SQL erläutert.

2 SQL/JSON

Am 06.03.2014 wurde von Jim Melton, Fred Zemke, Beda Hammerschmidt, Krishna Kulkarni, Zhen Hua Liu, Jan-Eike Michels, Doug McMahon, Fatma Özcan und Hamid Pirahesh ein Änderungsvorschlag zu SQL unterbreitet. Dieser hat das Ziel, dass JSON Format in SQL zu integrieren. Als Gründe zu diesem Vorschlag wurde aufgeführt das JSON ein weitverbreitetes Datenaustauschformat sei und im Unternehmensumfeld eine wichtige Rolle spielt. Aufgrund dessen sollte die Notwendigkeit der Nutzung mehrerer Anwendungen zur Verarbeitung von Unternehmensdaten vermieden werden. Konkret beinhaltet der Vorschlag die Unterstützung folgender Anwendungsszenarien:

- JSON Daten sollen in relationalen SQL Datenbanken speicherbar sein
- Es soll die Möglichkeit geschaaffen werden JSON Daten aus relationalen Daten zu erzeugen
- strukturierte JSON Daten sollen in relationale Daten umwandelbar sein.
- JSON Daten in SQL Datenbanken sollen filter- und suchbar sein

Explizit ausgeschlossen werden in dem Änderungsvorschlag, das Aktualisieren von JSON Daten, die Speicherung von atomaren Werten als JSON und ein Zugriff auf externe JSON Daten.[3]
Um SQL um die Anforderungen erweitern zu können, muss die Sprache um neue Operatoren und Funktionen erweitert werden. Folgende neue Operationen und Funktionen sind vorgesehen:

- IS JSON - prüft ob eine Zelle der Tabelle JSON Daten enthält
- JSON_VALUE - extrahiert einen atomaren Wert aus JSON Daten
- JSON_TABLE - konvertiert einen JSON Datensatz zu einer SQL Tabelle
- JSON_EXISTS - prüft ob JSON Wert vorhanden ist
- JSON_QUERY - extrahiert einen JSON Wert aus JSON Daten
- JSON_OBJECT, JSON_OBJECTAGG, JSON_ARRAY, JSON_ARRAYAGG - dienen der Erzeugung von JSON Daten aus relationalen Daten

Neben diesen Funktionen ist es weiterhin nötig JSON Daten adressieren zu können. Diesem Zweck widmet sich die SQL/JSON Path Language, eine Pfadsprache, welche andere JSON Pfadsprachen als Vorbild hat und an SQL/XML angelehnt ist.
Im Folgenden soll beispielhaft eine SQL/JSON-Anfrage demonstriert werden.

id	name	relatives
1	Bob	{“aunt“:“Tina“}
2	Alice	{“uncle“:“Jim“, “father“:“Gordon“}
3	Carl	{“grandmother“:“Elfriede“}
4	Susan	{“mother“:“Tina“, “father“:“Richy“,“son“:“Gustav“}

Tabelle 1: beispielhafter Auszug einer SQL Datenbank

Tabelle 1 zeigt einen möglichen Auszug aus einer relationalen Datenbank, welche in der dritten Spalte JSON Daten beinhaltet. Um nun beispielsweise die ids aller Zeilen, in denen die dritte Spalte ein JSON-Object mit dem Parameter *father* enthält, zu ermitteln, muss nach dem Änderungsvorschlag folgende Anfrage ausgeführt werden.[4]

```
1 SELECT T.id
2 FROM T
3 WHERE JSON_EXISTS (T.relatives, 'lax $.father')
```

Listing 1: SQL/JSON Anfrage

Da der Änderungsvorschlag noch nicht in SQL aufgenommen wurde, stellt sich die Frage inwiefern die Ansätze bereits angewandt werden. Der Standard wird von einigen Datenbanksystemen bereits ganz oder teilweise umgesetzt, dazu gehören IDM DB2, OracleDB und SQL Server.

3 JSONiq

JSONiq ist eine funktionale Abfragesprache, welche unter der Creative Commons Lizenz veröffentlicht wurde. Sie ist dynamisch typisiert und deren Dateien besitzen die Endungen .jq oder .jqy. JSONiq wurde als JSON Pendant zu XQuery entwickelt, welches ein W3C Standard ist. Während die eigenständige Abfragesprache auf einer JSON ähnlichen Syntax basiert, existiert noch eine Erweiterung zu XQuery welche einen XML ähnlichen Syntax aufweist. Allerdings wird dieser Syntax nicht mehr gepflegt und gilt als deprecated. Neben der Abfragesprache gibt es mit JSOUND eine Schemadefinitionssprache um eine Strukturierung zu erzeugen.
Eine Abfrage mit JSONiq orientiert sich an SQLs *SELECT, WHERE, FROM*, benutzt allerdings das *FOR, LET, WHERE, ORDER BY, RETURN* Konstrukt. Ähnlich zu SQL/JSON benutzt JSONiq ebenfalls eine Pfadsprache zur Adressierung einzelner JSON Daten. Die Hauptaufgaben der Sprache sind zum einen, dass dynamische Erzeugen von JSON Daten und zum Anderen, die Transformation von JSON zu JSON oder XML.
Durch die diese Hauptaufgaben lassen sich folgende Anwendungsgebiete herleiten:

- Datenextraktion aus einer Datenbank zur Nutzung in einem Webservice
- Statusberichte eines auf JSON basierenden, dokumentenorientierten Datenbanksystems erstellen
- JSON Daten auswählen und zu XHTML transformieren um sie im Web zu veröffentlichen
- Zusammenführung von Daten verschiedener Systeme
- Umwandlung von JSON Objekten in andere Schemata

JSONiq ist bereits in vielen Systemen integriert. Dazu zählen zum Beispiel Zorba, ein NOSQL Query Processor, Sparksoniq, eine JSONiq Engine für HDFS, Nitros Base, eine High Performance Universal Database und jsoniq.js, ein Projekt welches JSONiq Anfragen zu JavaScript transferiert.[5][6]

```
1 let $person := {
2   "first name" : "Sarah",
3   "age" : 27,
4   "gender" : "female",
5   "friends" : [ "Jim", "Mary", "Jennifer"]
6 }
```

Listing 2: JSON Object

Das Listing 2 zeigt, wie mittels JSONiq ein JSON Object erzeugt werden kann. Dies geschieht über den in Zeile 1 angegebenen Operator *let*.
in der folgenden Tabelle reftbl:jsoniq soll gezeigt werden, wie einfache JSONiq Anfragen vollzogen werden können. Als Datengrundlage dient das JSON Object aus dem Listing 2.

Abfrage	Ergebnis
return $person."first name"	Sarah
return {"f": size($person.friends))}	{“f“ : 3}
return {"keys": [keys($person)]}	{“keys“ : [“first name“ , “age“ , “gender“ , “friends“]}
return $person.friends[[1+1]]	Mary
return $person.friends[]	[“Jim“ , “Mary“ , “Jennifer“]

Tabelle 2: JSONiq Anfragen

4 SQL++

Mit AsterixDB gibt es ein NOSQL Datenbanksystem der Apache Software Foundation, welches auf dem Asterix Data Model (ADM) basiert. Das ADM basiert auf JSON, angereicht durch Typen. Um dieses spezielle Datenformat abfragen zu können, bedarf es einer zu dem Format passenden Abfragesprache. Aus diesem Grund wurde SQL++ erschaffen. SQL++ weist erhebliche syntaktische Ähnlichkeiten zu SQL auf und ist komplett abwärtskompatibel zu diesem. Im AsterixDB Umfeld spricht man von Dataverses, Datatypes und Datasets. Dataverses sind das komplement zu Database in SQL. Mithilfe dieser lässt sich ein Schema definieren, welches mit Datentypen versehen wird. Um NOSQL gerecht zu werden, ist dieses Schema bei Bedarf variabel. Das bedeutet es müssen nicht alle Felder ausgefüllt sein bzw. weitere nicht im Schema vorhandene Informationen können ohne Schemaänderung hinzugefügt werden. Datasets bezeichnen nun die Daten des entsprechenden Datentyps.[1]

```
1 DROP DATAVERSE TinySocial IF EXISTS;
2     CREATE DATAVERSE TinySocial;
3     USE TinySocial;
```

```
4
5     CREATE TYPE ChirpUserType AS {
6         screenName: string,
7         lang: string,
8         friendsCount: int,
9         statusesCount: int,
10        name: string,
11        followersCount: int
12    };
13
14    CREATE DATASET ChirpUsers(ChirpUserType)
15        PRIMARY KEY screenName;
```

Listing 3: Definition Datentyp

Listing 3 zeigt wie eine Dataverse erzeugt wird (Zeile 2) und dieses um einen Datentypen ergänzt wird (Zeile 5-11). In Zeile 14-15 wird nun eine Instanz des Datentypes, also ein Dataset erzeugt. In diesem Beispiel lässt sich die Ähnlichkeit zu SQL gut erkennen. Neben der Verwendung von Typen, können auch typische relationale Datenbankmittel wie Primary Keys oder Indizes genutzt werden.

```
1 INSERT INTO ChirpUsers
2     ([
3     {"screenName":"NathanGiesen@211","lang":"en","friendsCount":18," ↘
          → statusesCount":473,"name":"Nathan Giesen","followersCount" ↘
          → :49416},
4     {"screenName":"ColineGeyer@63","lang":"en","friendsCount":121," ↘
          → statusesCount":362,"name":"Coline Geyer","followersCount" ↘
          → :17159}]);
```

Listing 4: Einfügen von Daten

Wie im Listing 4 zu sehen, lassen sich Daten mittels INSERT INTO hinzufügen. Anfrage gestalten sich im typischen SQL Syntax, wie in Listing 5 ersichtlich ist.[2]

```
1 SELECT VALUE user
2     FROM ChirpUsers user
3     WHERE user.screenName = "NathanGiesen@211";
```

Listing 5: Anfrage von Daten

5 Oracle Big Data SQL

Da Unternehmen ihre Daten aufgrund von verschiedenen Anforderungen in unterschiedlichen Datenbanksystemen verwalten, entwickelte Oracle mit ORacle Big Data SQL eine Technologie um diese Datenquellen zusammenführen zu können.

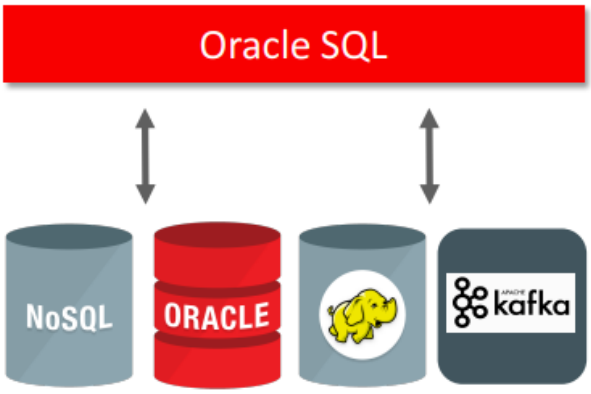

Abbildung 1: Oracle Big Data SQL

Ziel soll es sein, eine vielzahl von Datenquellen mittels Oracle SQL anzusprechen. Abbildung 1 verdeutlicht dieses Vorhaben. So lassen sich Oracles eigene Datenbanksysteme, sowie NoSQL Datenbanksysteme, HDFS Syteme und Apache Kafke mittels Oracle Big Data SQL ansprechen.
Die Grundlage bildet die Oracle Datenbank in die mittels sogenannter externer Tabellen weitere Datenquellen angehangen werden können. Diese externen Tabellen sind Oracle Database Objects und identifizieren den Ort und die Art der Daten. Um diese Daten nun mit Oracle SQL anfragen zu können, benötigen sie einen Acsess Driver, eine Art Schnittstellendefinition für Oracle SQL. Diese Treiber sind aktuell für HDFS und Apache Hive implementiert und dienen dem parsen der externen Daten.
Da die externen Daten an ihren eigentlichen Orten verbleiben kann eine Anfrage über Oracle SQL sehr ressourcenintensiv sein. Aus diesem Grund entwickelte Oracle die sogenannte Smart Scan Technologie. Diese erlaubt es dem Nutzer die Abfragen direkt auf dem betroffenen System z.B.: einem Hadoop Cluster auszuführen. Dadurch wird ein kompletter Scan der Daten vermieden. Weiterhin können die externen Daten auch lokal gefiltert und Joins mittels Bloom Filter optimiert werden.[7]

6 Fazit

Zusammenfassend lässt sich sagen, dass NoSQL Datenbanksysteme sehr unterschiedlich sein können. Aus diesem Grund gibt es für fasst jedes NoSQl System eine eigene Abfragesprache. Aktuell gibt es keinen Standard wie SQL bei relationalen Datenbanken, wobei einige Abfragesprachen für NoSQL dies von sich behaupten. Ein weiter Grund für einen fehlenden Standard ist die Eigenschaft der Schemafreiheit in NoSQL Datenbanken.
Objektiv betrachtet wäre es jedoch möglich einen Standard zu definieren, welcher eine große Anzahl an Datenbanksystemen anfragen könnte. Einen Spezialfall bilden lediglich die Graphdatenbanken, welche aufgrund ihrer Struktur, spezielle Abfragen benötigen.

Literatur

[1] AsterixDB: *The Asterix Data Model (ADM)*. https://ci.apache.org/projects/asterixdb/datamodel.html, . – (abgerufen am 25.06.2018)

[2] AsterixDB: *AsterixDB 101: An ADM and SQL++ Primer*. https://ci.apache.org/projects/asterixdb/sqlpp/primer-sqlpp.html, . – (abgerufen am 25.06.2018)

[3] Jim Melton, Fred Zemke, Beda Hammerschmidt, Krishna Kulkarni, Zhen Hua Liu, Jan-Eike Michels, Doug McMahon, Fatma Özcan, Hamid Pirahesh: *SQL/JSON Part 1*. https://www.wiscorp.com/pub/DM32.2-2014-00024R1_JSON-SQL-Proposal-1.pdf, . – (abgerufen am 25.06.2018)

[4] Jim Melton, Fred Zemke, Beda Hammerschmidt, Krishna Kulkarni, Zhen Hua Liu, Jan-Eike Michels, Doug McMahon, Fatma Özcan, Hamid Pirahesh: *SQL/JSON Part 1*. https://www.wiscorp.com/pub/DM32.2-2014-00025r1-sql-json-part-2.pdf, . – (abgerufen am 25.06.2018)

[5] Jonathan Robie, Matthias Brantner, Daniela Florescu, Ghislain Fourny, Till Westmann: *JSONiq: XQuery for JSON*. https://www.w3.org/2011/10/integration-workshop/p/Documentation-0.1-JSONiq-Article-en-US.pdf, . – (abgerufen am 25.06.2018)

[6] JSONiq: *The JSON Query Language*. http://www.jsoniq.org/#, . – (abgerufen am 25.06.2018)

[7] Oracle: *Oracle Big Data SQL Online Documentation Library Release 3.1*. https://docs.oracle.com/bigdata/bds31/index.htm, . – (abgerufen am 25.06.2018)

[8] Revell, Matthew: *Introduction to NoSQL Query*. https://www.exoscale.com/syslog/nosql-query/, . – (abgerufen am 25.06.2018)

[9] Steeman, Jan: *Query Mechanisms for NoSQL Databases*. https://de.slideshare.net/arangodb/query-mechanisms-for-nosql-databases, . – (abgerufen am 25.06.2018)