OS: Datenbanksysteme - Aktuelle Trends

Anfragesprachen für Big Data

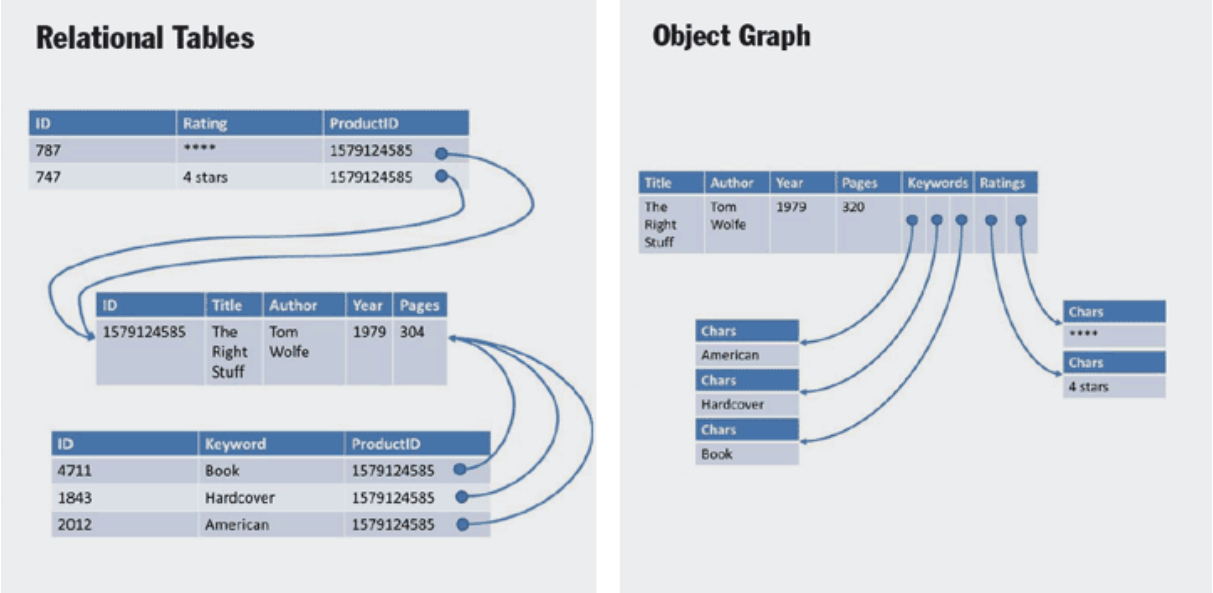
1 ”noSQL is really coSQL”

Im folgenden wird Versucht den Unterschied zwischen SQL und noSQL auf mathematischer Basis zu finden. Durch diese betrachtung auf niederer Ebene soll eine Grundlage für eine gemeinsame Query-Sprache geschaffen werden.

Diese Sektion basiert auf ”A co-Relational Model of Data for Large Shared Data Banks” von Erik Meijer und Gavin Bierman (Microsoft).

1.1 Objektgraphen gegen relationale Tabellen

Anhand eines Beispielmodells soll der Unterschied der Darstellung in einem Objektgraphen sowie in relationalen Tabellen gezeigt werden. Als Beispiel dient ein einfaches Modell eine Buch-Datenbank welche ein Produkt mit zugehörigen Autor, Bewertungen und Schlüsselwörtern verbindet.



Der Unterschied lässt sich auf die Definition einer Zeile bzw. Objektes zurückführen. So ist in relationalen Tabellen eine Zeile als:

```
row ::= new { ..., scalar, ... }
```

definiert. Es sind ausschließlich Skalare erlaubt wodurch Verschachtlungen oder direkte Verweise nicht möglich sind. In einem Objektgraphen ist ein Wert als:

```
value ::= new { ..., value, ... } | scalar
```

definiert. Jeder Wert kann rekursiv weitere Werte enthalten wodurch eine direkte Verlinkung von weiteren komplexen Werten oder das halten von Mengen ermöglicht wird.

Im SQL ist die Identität eines Wertes Teil des Wertes während im noSQL diese Identität erst durch einen referenzierenden Schlüssel gegeben wird. Ein Keyword besitzt im relationalen Beispiel eine Identität da es durch seine ProductID auf ein Produkt Verweist. Im Objekt-Graphen sind Keywords als reine Zeichenketten gespeichert, erst durch die Referenz ausgehend von einem Produkt wird dieser Zeichenkette eine spezielle Bedeutung zugewiesen.

Wie im Bild sichtbar ist die Repräsentation des Beispiels in beiden Varianten dennoch sehr ähnlich, nur die Pfeile sind umgekehrt.

1.2 Kategorientheorie

Beide Modelle bestehen aus Elementen und Pfeilen, diese Gebilde sind auch in der Kategorientheorie zu finden. In diese besteht eine Kategorie C aus einer Klasse von Objekten $Ob(C)$ sowie einer Klasse von Morphismen (sogenannten Pfeilen): $Mor_C(X, Y)$.

Ein tiefgreifender Aspekt besteht in den dualen Kategorien. Für eine duale Kategorie $co(C)$ zu einer Kategorie C gilt $Ob(C) = Ob(co(C))$ (sie enthält die gleichen Objekte) sowie $Mor_C(X, Y) = Mor_{co(C)}(Y, X)$ (alle Pfeile sind umgekehrt). Außerdem besitzt jede Aussage, Satz oder Definition T ein Dual $co(T)$: "Wenn T in C gilt, so gilt auch $co(T)$ in $co(C)$".

Im SQL zeigen Child-Elemente auf Parent-Elemente wenn ein Fremdschlüssel identisch zum Primärschlüssel des Parent-Elementes ist wobei im noSQL Parent-Elemente auf Child-Elemente zeigen wenn ein Pointer im Parent-Element identisch zur Adresse des Child-Elementes ist.

$\hookrightarrow$ noSQL ist dual zu SQL $\rightarrow$ "noSQL is really coSQL"

1.3 Queries

Es wird versucht die Grundlage für eine vereinigte Query-Language für SQL und coSQL zu schaffen. Dazu soll ein einheitliches Interface für Collections von Zeilen definiert werden sowie generalisierte Queries auf diesem Interface möglich sein. Somit werden identische Operatoren für eine generalisierte Collection `M<T>` mit Werten vom beliebigen Typ `T` definiert.

Interface für eine abstrakten Collection `M<T>`:

$$\begin{aligned} \varnothing &\in M<T> \\ \{\_\} &\in T \Rightarrow M<T> \\ \cup &\in (M<T>, M<T>) \Rightarrow M<T> \end{aligned}$$

Generalisierung der traditionellen relationalen Operatoren:

$$\begin{aligned} \sigma_P &\in (M<T>, (T \Rightarrow bool)) \Rightarrow M<T> \\ \pi_F &\in (M<T>, (T \Rightarrow S)) \Rightarrow M<S> \\ \times &\in (M<T>, M<S>) \Rightarrow M<T, S> \end{aligned}$$

Die Implementierung dieser Operatoren ist allgemein mit dem `SelectMany` Operator (im SQL als CROSS APPLY bezeichnet) möglich:

$$SelectMany \in (M<T>, (T \Rightarrow M<S>)) \Rightarrow M<S>$$

Somit können die traditionellen relationalen Operatoren auf einer einheitlichen Basis definiert werden:

$$\begin{aligned} \sigma_P(as) &= SelectMany(as, a \Rightarrow P(a)?\{a\} : \varnothing) \\ \pi_F(as) &= SelectMany(as, a \Rightarrow \{F(a)\}) \\ as \times bs &= SelectMany(as, a \Rightarrow \pi(bs, b \Rightarrow (a, b))) \end{aligned}$$

LINQ Queries besitzen einen SQL ähnlicher Syntax für bekannte Operatoren (`xs.Where(P)` für $\sigma_P(xs)$ und `xs.Select(F)` für $\pi_F(xs)$) sowie weitere Operatoren für Aggregation und Gruppierung:

$$Aggregate \in (M<T>, ((T,T) \Rightarrow T)) \Rightarrow T$$
$$GroupBy \in (M<T>, (T \Rightarrow K)) \Rightarrow M<K, M<T>>$$

Es gilt generell dass jede Datenquelle welche LINQ Operatoren implementiert mit gewohnter Syntax abgefragt werden, dies gilt unabhängig davon ob es sich um eine SQL oder coSQL Quelle handelt.

2 UnQL

UnQL steht für "Unstructured Data Query Language", ein Projekt welches in Kooperation zwischen Couchbase und SQLite gestartet wurde und versucht hat eine einheitliche Query Language für alle Datenbanktypen zu schaffen. Diese sollte einen SQL ähnlichen Syntax bereit stellen. Jedoch ist das Projekt vollkommen zum Stillstand gekommen und wird allgemein als Fehlschlag bezeichnet. Als eines der größten Probleme soll der Versuch gewesen sein absolut alle Arten von Datenbanken zu unterstützen: Key-Value-Stores, Document-Stores, Graph-Databases und weitere.

3 JSONiq

JSONiq bezeichnet eine Query- und funktionale Programmiersprache welche in der Lage ist Daten aus JSON-Dokumenten sowie allen anderen als JSON darstellbaren Datenquellen zu extrahieren und transformieren. Sie besitzt sogenannte FLWOR-Ausdrücke welche SQL-Queries sehr ähnlich sind. FLWOR steht für **F**OR, **L**ET, **W**HERE, **O**RDER BY, **R**ETURN welches die grundlegende Struktur der Queries wiedergibt.

3.1 Implementierungen

Bekannte Implementierungen von JSONiq bestehen in:

- 28.io
- Zorba
- IBM WebSphere
- BeniBela Internet Tools

Über IBM WebSphere sind sehr beschränkte Informationen zugänglich, die BeniBela Internet Tools bilden eine minimale Implementierung, Zorba und 28.io bieten weit entwickelte Systeme mit vorhandenen Anbindungen an Vorhandene Datenbanklösungen wobei 28.io eine kommerzielle Lösung darstellt und Zorba Open-Source ist.

Vorhandene Datenverbindungen:

28.io MongoDB, Couchbase, JDBC, Cloudant, CloudSearch, Cloud Services, S3, SPARQL, MarkLogic, ElasticSearch, Hadoop

Zorba Couchbase, JDBC, SQLite, Oracle NoSQL

3.2 Zorba Queries

3.2.1 SQLite

```
import module namespace s =
"http://www.zorba-xquery.com/modules/sqlite";

let $con := s:connect("small.db")
return s:execute-query($con, "select * from maintable");
```

Wie zu erkennen ist bietet Zorba jedoch keinen direkten Zugriff auf die SQLite Datenbank sondern nur eine Möglichkeit vollständige SQL-Queries auszuführen und die resultierenden Daten mit JSONiq weiter zu verarbeiten.

3.2.2 Couchbase

```
import module namespace cb = "http://www.zorba-xquery.com/modules/couchbase";
variable $db := cb:connect({ "host" : "localhost:8091", "username" : null,
"password" : null, "bucket" : "gamesim-sample" });

variable $view-name := cb:create-view($db, "testview", "testview",
{ "key" : "doc.uuid", "values" : "doc" });

let $rows := cb:view($db, $view-name)("rows")[]
for $row in $rows[$$.key eq "5ab3299d-b914-40b5-a838-a5b665f93330"]
return $row.value
```

Query ausgeführt mit Zorba:

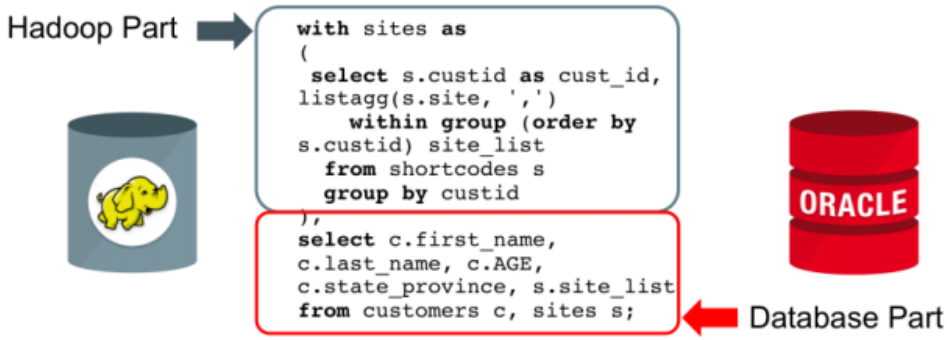
```
{
"experience" : 1,
"hitpoints" : 215,
"jsonType" : "player",
"level" : 2,
"loggedIn" : false,
"name" : "Dustin2",
"uuid" : "5ab3299d-b914-40b5-a838-a5b665f93330"
}
```

Es ist möglich Couchbase Views einfach zu definieren und Auswertungen auf den Resultaten auszuführen, jedoch können ähnlich wie bei der SQLite Anbindung nur komplette Anfragen ausgeführt werden und Ergebnismengen weiter mit JSONiq verarbeitet werden. Im Beispiel wird nach einem einzelnen Key gefiltert wobei dennoch der komplette Datenbestand der Couchbase zurückgeliefert und in das JSON Format konvertiert wird bevor eine Filterung durchgeführt wurde.

4 Oracle Big Data SQL

Oracle Big Data SQL ist Teil der Oracle Big Data Appliance welches ein ”multi-purpose engineered system for Hadoop and NoSQL processing” darstellt. Es kombiniert Oracle DB, Hadoop und NoSQL in eine gemeinsame Anfrage. Mit Hilfe von ”Smart Scan” Technologie soll die maximale Performance erreicht werden.

Ein einfacher Ansatz um dies zu Realisieren bietet die sogenannte ”Language-Level Federation”, dabei bietet jede Datenquelle einen SQL Zugang wobei SQL Queries in der Sprach-Ebene aufgeteilt werden. Ein Koordinator soll anschließend die Ergebnisse verbinden.



Dieses Vorgehen beinhaltet jedoch viele Probleme, so ist nicht klar ob alle Datenquellen einen identischen SQL Dialekt bereit stellen, die Queries werden auf einen Teil von SQL reduziert den alle unterstützen, das Ressourcenmanagement ist nicht global welches laut Oracle zu schlechterer Performance führt und eine Zugriffskontrolle ist problematisch.

Deswegen wird das sogenannte ”Query Franchising” genutzt. Dabei befinden sich auf jedem System dedizierte ”Big Data SQL agents”. Für Benutzer wird die komplette SQL Funktionalität bereit gestellt und die Daten wie normale Datenbank Tabellen dargestellt. Das System führt die Arbeitsverteilung auf Datenebene durch. Dies wird durch ”Smart Scan” optimiert welches dafür sorgt das die Verarbeitung dort stattfindet wo die Daten gehalten werden. Smart-Scan beinhaltet: ”WHERE Clause Evaluation”, ”Column Projection”, ”Bloom Filters for Better Join Performance” und ”JSON Parsing, Data Mining Model Evaluation”. Die Ergebnisse werden im Oracle-Format von allen Datenquellen zurückgegeben und anschließend verbunden und verarbeitet.

Quellen

[1] Couchbase. UnQL Query Language Unveiled by Couchbase and SQLite. www.couchbase.com/press-releases/unql-query-language.

[2] Ghislain Fourny. JSONiq: The SQL of NoSQL. www.28.io/dl/JSONiq_The_SQL_of_NoSQL-published.pdf.

[3] Erik Meijer and Gavin M. Bierman. A co-relational model of data for large shared data banks. *Commun. ACM*, 54(4):49–58, 2011.

[4] Oracle White Paper. Unified Query for Big Data Management Systems. www.oracle.com/us/products/database/big-data-sql/.

[5] Claudius Weinberger. Is UnQL Dead? www.arangodb.com/2012/04/is_unql_dead/.