Speicherung von und Zugriff auf JSON Objekte in NoSQL Datenbanken

Betrachtung von Oracle, Cassandra, Redis

H. Thierbach / 30.01.2018 Version 0.3

NoSQL (Not only SQL) Datenbanken

Quelle https://www.bigdata-insider.de/was-ist-nosql-a-615718/

- Während relationale Datenbanken Tabellen mit Spalten und Zeilen für die Datenspeicherung verwenden, nutzen NoSQL-Datenbanken zum Beispiel Wertepaare, Objekte, Dokumente oder Listen und Reihen für die Organisation der Daten. NoSQL-Systeme haben gemeinsam, dass sie für Anwendungen optimiert sind, bei denen die SQL-basierten relationalen Datenbanken an ihre Grenzen stoßen. Sie stellen eine Ergänzung und Bereicherung der Datenbankwelt dar und werden oft als strukturierte Datenspeicher bezeichnet.
- Aufgrund ihres Aufbaus skalieren die NoSQL-Datenbanken horizontal. Dank der fehlendenstarren Schemata der relationalen Datenbanken, sind NoSQL-Systeme sehr flexibel einsetzbar und eignen sich für große Datenmengen. Ihre komplette Architektur ist auf Skalierbarkeit und Performance ausgelegt.

Was NoSQL-Datenbanken auszeichnet

Quelle https://www.bigdata-insider.de/was-ist-nosql-a-615718/

In NoSQL-Datenbanken sind die Forderungen umgesetzt, große, exponentiell wachsende Datenmengen performant zu verarbeiten sowie skalierbare, leistungsfähige und verteilte Architekturen zu unterstützen. Folgende typischen Eigenschaften zeichnen die NoSQL-Systeme aus:

- Die horizontale Skalierbarkeit,
- das Vermeiden von unnötiger Komplexität,
- eine hohe Performance und ein hoher Durchsatz,
- die Nutzung von allgemein gebräuchlicher Hardware,
- die Vermeidung von relationalen Ansätzen des Datenmappings,
- die Einfachheit in der Installation und Konfiguration von verteilten Datenbank-Clustern sowie
- die Unterstützung der aktuellen Hardwaregenerationen.

Mögliche NoSQL-Datenbankmodelle

Quelle https://www.bigdata-insider.de/was-ist-nosql-a-615718/

NoSQL-Datenbanken verwenden keine relationalen Datenbankmodelle. Es kommen je nach System und Produkt andere Modelle zum Einsatz. Dies können beispielsweisefolgende sein:

- Key-Value-Datenbankmodelle,
- spaltenorientierte Datenbankmodelle,
- dokumentenorientierte Datenbankmodelle und
- Graphendatenbankmodelle.
- Das Key-Value-Datenbankmodell nutzt einfache Schlüssel und Werteschemata. Einzelne Werte sind immer bestimmten Schlüsseln zugeordnet und können aus willkürlichen Zeichenketten oder strukturierten Zeichen bestehen. Die Datenbanklösung Redis nutzt das Key-Value-Datenbankmodell.

Fragestellung

- Welche DB unterstützt JSON?
- Wie ist die Unterstützung implementiert?
- Welche (defaut) DB-Features sind bei JSON Objekten möglich?
- Werden CSV Import unterstützt?
- Läuft die Datenbank in einem Docker Container?
- Steht die Datenbank zur Verfügung?
- Wird die Datenbank von P betrieben?

Oracle 12c

JSON Support ab **Oracle database 12c release 2 (12.1.0.2)**

Reference User siehe DOAG

→ https://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6246

- Schema required
- JSON stored as CLOB
- Simple Dot.Notation Access to JSON Data
  - Direkter Zugriff auf JSON-Attribute
- Indizierung von JSON Data
- Full text search in JSON Data
- Transaktionsicherheit
- Konsistenz Eventual consistancy* not Immediate Consistency (!)
- Server side Scripting via Stored Procedures
- csv import via sql loader script

* JSON data has often been stored in NoSQL databases such as OracleNoSQL Database and Oracle Berkeley DB. These allow for storage andretrieval of data that is not based on any schema, but they do not offerthe rigorous consistency models of relational databases.

To compensate for this shortcoming, a relational database issometimes used in parallel with a NoSQL database. Applications usingJSON data stored in the NoSQL database must then ensure data integritythemselves.

Oracle 12c Docker

→ https://apex.oracle.com/pls/apex/germancommunities/dbacommunity/tipp/6241/index.html

- Die Datenbank muß im Single Instance Modus betrieben werden
- Das Feature "Automatic Memory Management" steht nicht zur Verfügung
- Die Datenbank wird in einem Oracle Linux 7.3 Image betrieben

Oracle 12c

Beispiel

```
CREATE TABLE j_purchaseorder(
        idRAW (16) NOT NULL,
        date_loaded TIMESTAMP WITH TIME ZONE,
        po_document CLOB
        CONSTRAINT ensure_json CHECK(po_documentIS JSON)
);
```

Oracle 12c

Beispiel

```
INSERT INTO j_purchaseorder VALUES
       (
       SYS_GUID(),
       SYSTIMESTAMP,
       '{"PONumber": 1600,
       "Reference": "ABULL-20140421",
       "Requestor": "Alexis Bull",
       "User": "ABULL",
       "CostCenter": "A50",
       "ShippingInstructions" : {...},
       "Special Instructions" : null,
       "AllowPartialShipment" : true,"LineItems": [...]
       }'
);
```

Oracle 12c

Beispiel

Beispiele:

SELECT po.po_document.PONumberFROM j_purchaseorder po;

SELECT po.po_document.ShippingInstructions.PhoneFROM j_purchaseorder po;

SELECT po.po_document.ShippingInstructions.Phone.typeFROM j_purchaseorder po;

Oracle 12c

Beispiel

Beispiele:

```
SQL*Loader: Release 11.2.0.2.0 - Production on So Jan 28 23:39:30 2018

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

Control File:   /tmp/dummy.ctl
Data File:      /tmp/longFile.txt
  Bad File:     /tmp/longFile.bad
  Discard File:  none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:    none specified
Path used:      Conventional

Table DUMMY, loaded from every logical record.
Insert option in effect for this table: APPEND

   Column Name              Position   Len  Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ---------------------
EKID                           FIRST     *   , CHARACTER
STATUS                           NEXT     *   , CHARACTER
```

```
Table DUMMY:
  100000 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.


Space allocated for bind array:                 33024 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:        100000
Total logical records rejected:         0
Total logical records discarded:        0

Run began on So Jan 28 23:39:30 2018
Run ended on So Jan 28 23:39:39 2018

Elapsed time was:     00:00:09.35*
CPU time was:         00:00:01.00
```

* gemessen auf acer aspire one

Oracle 12c

Beispiel

Beispiele:

http://www.oracle.com/webfolder/technetwork/de/community/dbadmin/tipps/advanced_partitioning/index.html

Cassandra 3.11

Reference User
Twitter
Facebook
Walmart
Cisco
Netflix

JSON Support ab **Cassandra 2.2 (2015)**

→ https://docs.datastax.com/en/cql/3.3/cql/cql_using/useInsertJSON.html
→ http://docs.datastax.com/en/cql/3.3/cql/cql_using/useQueryJSON.html

- Schema required
- JSON stored as table
- Keine Simple Dot.Notation Access to JSON Data
  - Direkter Zugriff auf JSON-Attribute
- Indizierung von JSON Data
- Full text search in JSON Data
- Keine Transaktionsicherheit
- Kosistenz Eventual Consistency
- Kein Server side Scripting
- csv import via copy-from bulk loader

Cassandra 3.11

Beispiel

```
CREATE TABLE pdf (
        pdf_id text primary key,
        content blob,
        title text,
        type text,
        date text,
        size int )
;
```

Cassandra 3.11

Beispiel

```
INSERT INTO pdf_owner JSON ,
        `{
         "pdf_id" : " 829aa84a-4bba-411f-a4fb-38167a987cda",
         "content" : "...",
         "title" : “Lebenslauf",
         "type" : “Nachweis",
         "date" : “21.01.2018 12:34:21 ",
         "size" : 112231
        }`
;
```

Cassandra 3.11

Beispiel

```
select json content from pdf
      where pdf_id = '829aa84a-4bba-411f-a4fb-38167a987cda'
      allow filtering;
```

Cassandra 3.11

Beispiel

```
cqlsh:dev>copy dev.dummy (ekid, status) from 'largeFile.txt' with header = false;
Using 3 child processes

Starting copy of dev.dummy with columns [ekid, status].
Processed: 100000 rows; Rate:   12574 rows/s; Avg. rate:   13841 rows/s
100000 rows imported from 1 files in 7.225 seconds* (0 skipped).
cqlsh:dev>select count(*) from dummy;

 count
--------
 100001

(1 rows)
```

* gemessen auf Mac Book Pro

Cassandra 3.11

Beispiel

```
cqlsh:dev>insert into dummy (ekid, status) values ('dummy', 'gummy') using ttl 120;
cqlsh:dev>select * from dummy where ekid = 'dummy';

 ekid  | status
-------+--------
 dummy |  gummy
(1 rows)

after 120 sec:

cqlsh:dev>select * from dummy where ekid = 'dummy';

 ekid | status
------+--------
(0 rows)
```

Cassandra 3.11

Beispiel

```
// Instanciate a Session with Cassandra DB
Cluster cluster = null;
cluster = Cluster.builder().addContactPoints('127.0.0.1')
    .withRetryPolicy(Policies.defaultRetryPolicy())
    .withCredentials("", "")
    .build()
Session session = cluster.connect();

// Read the PDF Document in
FileInputStream fis=new FileInputStream("File.pdf");
byte[] b = new byte[fis.available() + 1];
int length=b.length;
fis.read(b);
```

```
// Convert the Input Stream into a buffered byte array
ByteBuffer buffer = ByteBuffer.wrap(b);
 …
docDate = date.format('YYY-MM-DD HH:mm:ss.SS');

// Prepare the Statement 'insert pdf document' and execute
PreparedStatement ps = session.prepare("insert into dev.pdf ( pdf_id,
content, title, type, date, size ) values(?,?,?,?,?,?)");
BoundStatement boundStatement = new BoundStatement(ps);
session.execute( boundStatement.bind(UUID,  buffer, Titel, 'Nachweis',
docDate, b.length));

System.exit(0);
```

REDIS 3.0.7 (hier REJSON)

JSON Support ab **Cassandra 2.2 (2015)**

→ http://rejson.io/
→ https://redislabs.com/blog/redis-as-a-json-store/

- Native Key:Value Store, schema free
- JSON stored as Value (JSON-Object)
- Simple Dot.Notation Access to JSON Data
  - Direkter Zugriff auf JSON-Attribute
  - Direkte Manipulation von Values
- Indizierung von JSON Data (only primary key)
- Transaktionsicherheit optimistic locking (via multi)
- Kosistenz Eventual Consistency
- Server side Scripting mit LUA
- csv Batch import via redis-cli --pipe

Reference User
Twitter
Snapchat
Github
Flickr
Pinterest

REDIS 3.0.7 (hier REJSON)

REDIS 3.0.7 (hier REJSON)

Beispiel

```
JSON.SET object. ,{"foo":"bar", "ans":"42"}`;
OK

JSON.GET object . ;
{"foo":"bar", "ans":"42"}

JSON.GET object .foo;
"bar"

JSON.NUMINCRBY object .ans 1
"43"
```

Vergleich

Cassandra vs Redis

→ https://db-engines.com/de/system/Cassandra%3BRedis

Cassandra vs Oracle

→ https://db-engines.com/de/system/Cassandra%3BOracle

Redis vs Oracle

→https://db-engines.com/en/system/Oracle%3BRedis