Native JSON Unterstützung in Oracle12*c*

Autor: Carsten Czarski, ORACLE Deutschland B.V. & Co KG

Mit der neuesten Version 12.1.0.2 der Oracle-Datenbank wird erstmals die native Unterstützung für das JSON-Format eingeführt. JSON kann in die Oracle-Datenbank gespeichert und mit Hilfe von SQL/JSON-Funktionen ausgewertet werden.

Neben dieser 'SQL and JSON duality' bietet die ebenfalls mitgelieferte REST-Webservice-Schnittstelle die Möglichkeit an, die Oracle-Datenbank als JSON Document Store *zu betreiben – völlig ohne SQL.*

JSON: Das flexible Datenaustauschformat

Das JSON-Format (*JavaScript Object Notation*) setzt sich vor allem bei Anwendungsentwicklern mehr und mehr für den Datenaustausch und oft auch als Datenablageformat durch. JSON ist, im Grunde genommen, JavaScript-Syntax. Kodiert man mit JavaScript-Code ein Objekt-Literal, wie in dem folgenden Listing gezeigt, so liegt de-facto JSON vor.

```
{
  "artikel": {
    "titel": "JSON in Oracle12c",
    "seiten": 3,
    "themen": ["Datenbank", "JSON", "SQL", "Tabellen"],
    "publikation": "DOAG Online"
  }
}
```

JSON bildet, wie XML, die Daten in hierarchischer Struktur ab. Allerdings gibt es für den Umgang mit JSON weit weniger Standards als für XML – und die bestehenden sind weniger strikt.

In der Praxis finden sich zwei typische Anwendungsgebiete für JSON. Zum einen wird es besonders im Umfeld von Webanwendungen als *Datenaustauschformat* zwischen Server und Browser eingesetzt. Das ist einleuchtend, denn der Browser kann JSON, da es JavaScript-Syntax ist, problemlos und ohne weiteres verarbeiten.

Das zweite Anwendungsgebiet ist die *Datenablage im JSON-Format* – diese kommt vor allem dann zum Einsatz, wenn es nicht oder nur schwer möglich ist, ein festes

Datenmodell und damit ein relationales Schema zu definieren. So kommen immer mehr Anwendungen in die Situation, dass flexible, sich häufig ändernde Attribute gespeichert werden müssen. Jedesmal das Datenbankschema anzupassen, ist zu aufwändig – gefragt ist die einfache Ablage – das Interpretieren der Daten geschieht dann später, wenn sie ausgewertet werden.

JSON bietet sich hierfür an - und gerade in einem RDBMS wie Oracle kann es die perfekte Ergänzung zum relationalen Modell sein. Die statischen Teile des Datenmodells werden nach wie vor "klassisch" mit Tabellen und Spalten modelliert; die flexiblen Teile als JSON abgespeichert.

Nun ist es natürlich nötig, dass man die als JSON gspeicherten Daten mit Mitteln der Datenbank, also SQL, bearbeiten und abfragen kann – und genau das ist der Schwerpunkt der JSON-Unterstützung in Oracle12*c*.

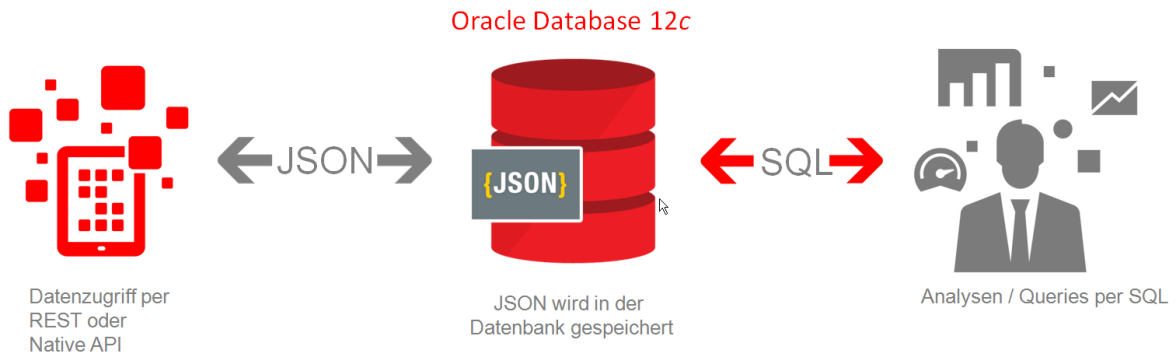

Abbildung 1: JSON and SQL Duality in Oracle12c

SQL ist allerdings nicht die einzige Schnittstelle für JSON in der Oracle-Datenbank. Darüber hinaus bringt die Datenbank eine REST-Webservice-Schnittstelle mit, mit der Anwendungen – völlig ohne SQL – JSON Dokumente speichern, bearbeiten oder abfragen können. Dies ist insbesondere für Webapplikationen ohne dedizierte JDBC-Datenbankschnittstelle interessant. Die Oracle-Datenbank kann so als *JSON Document Store* – ganz ohne SQL – verwendet werden.

JSON in der Datenbank speichern

Im Gegensatz zu XML wurde für JSON *kein* eigener Datentyp in der Oracle-Datenbank eingeführt. JSON wird als VARCHAR2, CLOB oder BLOB in Tabellenspalten abgelegt. Damit ist es sogar möglich, JSON und "Nicht-JSON" in ein- und derselben Tabellenspalte zu mischen (ob das aber sinnvoll ist, ist eine andere Frage).

Möchte man feststellen, ob der Inhalt einer Tabellenspalte syntaktisch korrektes JSON ist, leisten die neuen SQL-Operatoren *IS JSON* und *IS NOT JSON* wertvolle Dienste. Wie das folgende Beispiel zeigt, kann man diesen auch als Check-Constraint verwenden.

```
create table PO_JSON (
  ID        number(10) primary key,
  FILE_NAME varchar2(500),
  JSON      clob,
  constraint CK_JSON_IS_JSON check (JSON is json)
)
```

JSON-Dokumente können in die Tabelle nun mit gewöhnlichen SQL INSERT-Anweisungen abgelegt werden. Möchte man in dieser Tabelle einen normalen Text speichern, der kein JSON repräsentiert, so wird der Check-Constraint eine Fehlermeldung auslösen.

```
insert into PO_JSON (id, file_name, json) values (
  3, 'third-file.js', 'Ein Text'
)
*
FEHLER in Zeile 1:
ORA-02290: CHECK-Constraint (JSONTEST.CK_JSON_IS_JSON) verletzt
```

Gültige JSON-Dokumente warden dagegen anstandslos entgegengenommen; die folgende Abbildung zeigt ein etwas komplexeres JSON-Beispiel.

Abbildung 2: Ein JSON Beispiel: PurchaseOrder

SQL/JSON-Funktionen: JSON_VALUE

Mit der neuen SQL/JSON Funktion *JSON_VALUE* werden einzelne, skalare Werte aus den JSON-Dokumenten extrahiert. Per "Punktnotation" gelangt man zum gewünschten Knoten in der Hierarchie.

```
select json_value(
 json,
 '$.PurchaseOrder.Reference[0]' returning VARCHAR2
) from po_json where rownum < 10;

JSON_VALUE
------------------------------------------------------------
FORD-20021009123336872PDT
JONES-20011127121050471PST
:

9 Zeilen ausgewählt.
```

Die Funktion JSON_VALUE nimmt (wie auch die anderen SQL/JSON-Funktionen) zunächst das JSON-Dokument (oder die Tabellenspalte mit den JSON-Dokumenten) und dann einen *JSON-Pfadausdruck* entgegen. Im Pfadausdruck wird das JSON-Dokument selbst mit dem Dollarzeichen repräsentiert; von dort aus navigiert man per Punktnotation durch die Hierarchie. Auch in Arrays kann navigiert werden, wie das obige Beispiel zeigt: *"[0]"* repräsentiert das erste Objekt eines Arrays.

SQL/JSON Funktionen: JSON_QUERY

JSON_QUERY ist das "Gegenstück" zu JSON_VALUE: Diese Funktion dient dazu, JSON-Fragmente aus dem Dokument auszuschneiden. Der JSON-Pfadausdruck darf nun nicht mehr auf einen skalaren Wert zeigen - er muss auf ein JSON-Array oder Objekt verweisen. Die folgende Beispielabfrage zeigt dies.

```
select json_query(
 json, '$.PurchaseOrder.ShippingInstructions[0]'
) from po_json where rownum=1;

JSON_QUERY
----------------------------------------------------------
{"name":["Gerry B. Ford"],"address":["100 Oracle Parkway\r\nRedwood
Shores\r\nCA\r\n94065\r\nUSA"],"telephone":["650 506 7100"]}
```

Im Fehlerfall liefert JSON_QUERY genauso wie JSON_VALUE standardmäßig SQL NULL zurück. Dieses Verhalten kann aber durch Parameter verändert werden.

```
select json_query(
 json, '$.invalid.json.path
) from po_json where rownum=1;

JSON_QUERY
----------------------------------------------------
- SQL NULL -
```

SQL "Punktnotation" als Alternative

Anstelle der Funktionen JSON_QUERY oder JSON_VALUE kann man auch mit der "SQL-Punktnotation" arbeiten. Die Abfrage sieht dann wie folgt aus.

```
select e.json.PurchaseOrder.Reference
from po_json e where rownum <= 10;

PURCHASEORDER
------------------------------------------------
["FORD-20021009123336872PDT"]
["JONES-20011127121050471PST"]
["MARTIN-20011127121050401PST"]
:

10 Zeilen ausgewählt.
```

Wie man sieht, braucht es dann keine JSON_QUERY oder JSON_VALUE-Funktion mehr. Voraussetzung hierfür ist allerdings, dass, wie im Artikel oben dargestellt, ein Check-Constraint mit IS JSON auf der Tabellenspalte eingerichtet ist. Ohne Check-Constraint funktioniert die vereinfachte Punktnotation nicht.

SQL/JSON-Funktionen: JSON_TABLE

JSON_TABLE ist die mächtigste und mit Sicherheit auch interessanteste der SQL/JSON-Funktionen, denn diese erlaubt (ähnlich wie ihr XML-Pendant XMLTABLE), das Projizieren von JSON-Elementen zu relationalen Spalten. Es können also in einem einzigen JSON_TABLE-Aufruf mehrere JSON-Knoten selektiert und dann als relationale Ergebnisspalten projiziert werden. Die folgende Abfrage zeigt dies: Es werden aus jedem JSON-Dokument drei Elemente extrahiert und als Ergebnistabelle mit drei Spalten zurückgegeben.

```
select reference, requestor, costcenter
from po_json, json_table(
  json,
  '$.PurchaseOrder'
   columns (
     reference varchar2(30) path '$.Reference[0]',
     requestor varchar2(25) path '$.Requestor[0]',
     CostCenter varchar2(4) path '$.CostCenter[0]'
   )
 )

REFERENCE                      REQUESTOR                 COST
------------------------------ ------------------------- ----
FORD-20021009123336872PDT      Gerry B. Ford             R20
JONES-20011127121050471PST     Richard J Jones           R20

:                              :                         :

SCOTT-20011127121051793PST     Susan T. Scott            R20

678 Zeilen ausgewählt.
```

JSON_TABLE kann aber noch weiter gehen: So ist es möglich, auch die enthaltenen Arrays (1:n-Hierarchien) zu verarbeiten. Im Beispiel enthalten die "PurchaseOrder"-Dokumente jeweils ein Array mit den "LineItems", also den bestellten Gegenständen. Natürlich kann eine "Purchaseorder" auch mehrere "LineItems" beinhalten. Möchte man also alle "LineItems" aller JSON-Dokumente als relationale Tabelle aufbereitet haben, so nimmt man dafür wiederum JSON_TABLE und die *NESTED PATH*-Klausel.

```
select reference, requestor, num, descr, quantity
from po_json, json_table(
  json,
  '$.PurchaseOrder'
  columns (
    reference varchar2(30) path '$.Reference[0]',
    requestor varchar2(25) path '$.Requestor[0]',
    CostCenter varchar2(4) path '$.CostCenter[0]',
    nested path '$.LineItems[*].LineItem[*]' columns (
      num      number       path '$."\u0024".ItemNumber',
      descr    varchar2(40) path '$.Description[0]',
      quantity number       path '$.Part[0]."\u0024"."Quantity"'
    )
  )
)

REFERENCE     REQUESTOR            NUM DESCR                QUANTITY
------------- ------------------ ---- ------------------ --------
FORD-20021…   Gerry B. Ford         1 Ordet                     4
FORD-20021…   Gerry B. Ford         2 The Naked Kiss            3
FORD-20021…   Gerry B. Ford         3 Charade                   2
:             :                    : :                         :
ALLEN-2002... Michael L. Allen     8 Sid & Nancy               2

14913 Zeilen ausgewählt.
```

Man sieht sofort, dass JSON_TABLE eine extrem mächtige SQL-Funktion ist; man denke nur an das Weiterverarbeiten dieses Ergebnisses mit SQL-Aggregatsfunktionen wie SUM, COUNT oder AVG. Die Möglichkeiten von SQL können nun sehr einfach auf JSON-Dokumente angewendet werden.

Fazit

Mit den im Release 12.1.0.2 eingeführten SQL/JSON Funktionen geht die Oracle-Datenbank, was die Unterstützung von JSON angeht, einen großen Schritt nach vorne. JSON-Daten können nun mit nativen SQL Mitteln gelesen, interpretiert und verarbeitet werden - dazu kann dann der ganze SQL Befehlsumfang genutzt werden. Für PL/SQL Stored Procedures fehlt die JSON Unterstützung im ersten Release noch – das Development-Team arbeitet daran. Hier lohnt es sich, einen Blick auf das kommende APEX 5.0 zu werfen. Das darin enthaltene PL/SQL-Paket APEX_JSON kann man auch schon auf der Early Adopter Instanz ausprobieren

Weitere Informationen

[1] JSON auf Wikipedia
http://de.wikipedia.org/wiki/JavaScript_Object_Notation

[2] Oracle12*c* Dokumentation: JSON in der Oracle-Datenbank
http://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6246

[3] Artikel zum Thema in der APEX Community (mit Beispieldaten-Download)
http://tinyurl.com/oracle12cjson

Carsten Czarski
Carsten.Czarski@oracle.com
http://twitter.com/cczarski - http://sql-plsql-de.blogspot.com