Using JSON in Oracle 12c

Presented by: John Jay King

Download this paper from: http://www.kingtraining.com

Session Objectives

- Understand how Oracle 12c supports JSON
- Store JSON data in the database
- Become familiar with JSON conditionals and functions
- Query JSON data using JSON conditionals and functions
- Query JSON data using SQL via JSON_TABLE

Who Am I?

- John King – Partner, King Training Resources
- Oracle Ace Director
- Member Oak Table Network
- Providing training to Oracle and IT community for over 25 years – http://www.kingtraining.com
- "Techie" who knows Oracle, ADF, SQL, Java, and PL/SQL pretty well (along with many other topics)
- Member of AZORA, ODTUG, IOUG, and RMOUG

What is JSON

- JSON (JavaScript Object Notation) is a language-independent open data format (www.json.org)
- Stores text in name-value pairs
- Originally in JavaScript, now also in: Java, R, .NET, PHP, Python, Node.js, and Oracle
- Most-often used for data interchange
- Frequently used to pass data to and from REST-style web services
- Becoming favored for big-data persistence

12c (12.1.0.2) and JSON

- 12c patch-set 2 (12.1.0.2) added JSON
- JSON documents are stored as VARCHAR2, CLOB, or BLOB data type
- JSON data works with all existing Oracle features including SQL and Analytics
- 12c supports path-based queries of JSON data stored in the database, JSON Path Language, and JSON Path Expressions
- JSON is used in SQL via SQL/JSON views
- JSON documents may be indexed

JSON-XML Similarities

- JSON is text only, just like XML and thus is an excellent vehicle for data interchange
- JSON and XML are “human readable” and "self-describing” (sort of…)
- JSON and XML are hierarchical (data sets nested within data sets)
- JSON and XML offer validation capability; XML’s is more mature and capable today

JSON-XML Dissimilarities

- XML is verbose, JSON is shorter
- JSON has no end tags, required in XML
- JSON is quicker to read and write
- Reading XML documents requires “walking the DOM” – JSON does not
- JSON works more easily and is faster than XML when working with AJAX
- XML documents must be tested for “well-formed”-ness before processing

JSON Syntax

- Data is stored using comma-delimited name/value pairs
  - Field name/key (surrounded by double quotes)
  - Colon ":",
  - Value (double quotes surround strings, numeric values unquoted, boolean true/false/null
- Objects included inside curly braces "{" & "}"

```
{"lastName":"King"}
```

- Arrays use brackets "[" & "]" and commas

```
[ {"lastName": "King"},
  {"lastName": "Manzo"} ]
```

XML File

```
<?xml version="1.0"?>
<myBooks>
  <book>
    <name>Learning XML</name>
    <author>Eric T. Ray</author>
    <publisher>O'Reilly</publisher>
  </book>
  <book>
    <name>XML Bible</name>
    <author>Elliotte Rusty Harold</author>
    <publisher>IDG Books</publisher>
  </book>
  <book>
    <name>XML by Example</name>
    <author>Sean McGrath</author>
  </book>
</myBooks>
```

JSON File

```
{"myBooks":
  [ {"book": {
            "name":"Learning XML",
            "author":"Eric T. Ray",
            "publisher":"O'Reilly" }
    },
    {"book": {
            "name":"XML Bible",
            "author":"Elliotte Rusty Harold",
            "publisher":"IDG Books" }
    },
    {"book": {
            "name":"XML by Example",
            "author":"Sean McGrath",
            "publisher":"Prentice-Hall" }
    }
  ]}
```

Oracle as JSON Data Store

- JSON documents are stored in the database using existing data types
  - VARCHAR2, CLOB and BLOB for character mode JSON
  - External JSON data sources (including those in HDFS file system) are accessible through external tables
- JSON data is available to SQL via relational views based upon JSON_TABLE
- Oracle JSON documents may be indexed; JSON paths may use functional indexes

JSON SQL

- JSON content is accessible from SQL and may be treated as JSON using:
  - JSON conditional operators
  - JSON functions
- JSON operators and functions use JSON Path language to navigate JSON objects

JSON Conditionals

- JSON content is accessible from SQL via new condition operators
  - IS JSON — Validate JSON, often in CHECK constraint and WHERE (IS / IS NOT)
  - JSON_EXISTS — True if JSON path exists in document
  - JSON_TEXTCONTAINS — True if text string is found in JSON property values (uses Oracle Text)

JSON Functions

- JSON content is also accessed via new functions
  - JSON_VALUE — Used to query a scalar value from a JSON document
  - JSON_QUERY — Used to query all or part of a JSON document
  - JSON_TABLE — Used to query JSON document and create relational-style columns

IS JSON – IS NOT JSON

- IS JSON returns TRUE if specified expression is a valid JSON document

```
**expr**    IS JSON | IS NOT JSON
            FORMAT JSON
            STRICT|LAX
            WITH|WITHOUT UNIQUE KEYS
```

  - IS true if valid JSON, IS NOT true if not JSON
  - FORMAT JSON (only required for BLOB data)
  - STRICT / LAX strictness level required for true
  - WITH / WITHOUT if unique keys are required

IS JSON Check Constraint

```
create table deptj
(id raw(16) not null,
 dept_info clob constraint deptjson
                  check (dept_info is json)
);
```

```
create table deptj
(id raw(16) not null,
  dept_info clob constraint deptjson
     check (dept_info is json strict)
);
```

IS JSON in WHERE

```
select id,dept_info
from deptj
where dept_info is json
```

```
select id,dept_info
from deptj
where dept_info is json strict;
```

```
select id,dept_info
from deptj
where dept_info is json format json strict;
```

JSON and DML

```
insert into deptj values
(sys_guid(),
 '{"departments":{
   "DEPTNO": 10, "DNAME": "ACCOUNTING", "LOC": "NEW YORK",
    "deptemps": [
                 { "EMPNO": 7782,
                   "ENAME": "CLARK",
                   "JOB": "MANAGER",
                   "MGR": 7839,
                   "HIREDATE": "09-JUN-81",
                   "pay":{
                          "SAL": 2450,
                          "COMM": null},
                   "DEPTNO": "10"
                  },
                  /* more */
```

Simple JSON Query

```
select dept_info
from deptj;

DEPT_INFO
-----------------------------------------
{"departments":{
 "DEPTNO": 10,
 "DNAME": "ACCOUNTING",
 "LOC": "NEW YORK",
 "deptemps": [
                {
                 "EMPNO": 7782,
                 "ENAME": "CLARK",
  **** more ****
```

JSON Path

- $.jsonpath Identifies path in JSON document
  - $. (required)
  - jsonpath (describes part of JSON document to be searched, if omitted entire document returned
- JSON path used by JSON_QUERY, JSON_VALUE, JSON_TABLE, JSON_EXISTS, JSON_TEXTCONTAINS

JSON_VALUE

- JSON_VALUE finds a value in a JSON document and returns it as VARCHAR2 or NUMBER to SQL

```
JSON_VALUE (expr | expr FORMAT JSON,
      '$.jsonpath'
      RETURNING VARCHAR2(n) | NUMBER(n,n)
      ERROR | NULL | DEFAULT xxx ON ERROR
)
```

Query with JSON_VALUE

```
select json_value(dept_info, '$.departments.DNAME')
from deptj;
DNAME
----------------
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

JSON_VALUE

- JSON_VALUE returns scalar values from JSON data

```
select json_value(dept_info
      , '$.departments.DNAME')
      from deptj;

select json_value(dept_info
      , '$.departments.deptemps[0].ENAME')
      from deptj;

select dept_info
      from deptj
      where json_value(dept_info,
'$[0].departments.DEPTNO') = '10'
```

JSON_QUERY

- JSON_QUERY finds values in a JSON document and returns a character string

```
JSON_QUERY(expr | expr FORMAT JSON,
 '$.jsonpath'
    RETURNING VARCHAR2(n)
    PRETTY
    ASCII
  WITH | WITH CONDITIONAL | WITH UNCONDITIONAL
   WRAPPER | ARRAY WRAPPER
  ERROR | NULL | DEFAULT xxx ON ERROR )
```

  - Use PRETTY to pretty-print output
  - Use ASCII to escape non-ASCII characters
  - WRAPPER needed if multiple values or scalar returned

Query with JSON_QUERY

```
select json_query(dept_info,
     '$.departments.DEPTEMPS.ENAME'
          with unconditional wrapper)
   from deptj dj;
```

```
select json_query(dept_info,
     '$.departments.DEPTEMPS.ENAME'
          pretty with wrapper)
   from deptj dj;
```

```
select json_query(dept_info,
     '$.departments.DEPTEMPS[0].ENAME'
          pretty with conditional wrapper)
   from deptj dj;
```

JSON_TABLE

- JSON_TABLE maps JSON data into relational rows and columns

```
JSON_TABLE ( expr | expr FORMAT JSON,'$.jsonpath'
      ERROR | NULL | DEFAULT xxx ON ERROR
      COLUMNS (
         colname1 datatype
            EXISTS PATH '$.jsonpath'
            ERROR|NULL|DEFAULT xxx ON ERROR,
         colname2 datatype FORMAT JSON
            jsonquerywrapper PATH '$.jsonpath'
            ERROR|NULL|DEFAULT xxx ON ERROR,
         colname3 datatype PATH '$.jsonpath'
            ERROR|NULL|DEFAULT xxx ON ERROR,
            NESTED PATH '$.jsonpath' COLUMNS (...),
         colname4 FOR ORDINALITY
      )
```

Query with JSON_TABLE

```
select dname,ename,job,sal
from deptj, json_table(dept_info,'$.departments'
            columns (dname varchar2(15) path '$.DNAME'
            ,nested path '$.deptemps[*]'
              columns (ename varchar2(20) path '$.ENAME'
              ,job varchar2(20) path '$.JOB'
                ,nested path '$.pay'
                  columns (sal number path '$.SAL')
              )
));
DNAME          ENAME     JOB                SAL
-------------- -------   --------   ----------
ACCOUNTING     CLARK     MANAGER            2450
ACCOUNTING     KING      PRESIDENT          5000
**** more ****
```

JSON and Indexing

- JSON function/expression based indexes may be “b-tree” (normal) or bitmap
- JSON full-search context indexes may also be created

JSON Indexes, 1

- Function/Expression-based indexes

```
create unique index deptj_ix
     on deptj
     (json_value(dept_info,'$.DEPTNO'));
```

```
create bitmap index deptj_emp_ix
     on deptj
     (json_value(dept_info,
                 '$.DEPTEMPS.EMPNO'));
```

JSON Indexes, 2

- Full-search context index

```
create index deptj_ctx_ix
  on deptj (dept_info)
     indextype is ctxsys.context
     parameters ('section group
          CTXSYS.JSON_SECTION_GROUP
          sync (on commit)’
);
```

  - Not just for TEXTCONTAINS may also be used for JSON_VALUE and JSON_EXISTS tests

Wrapping it all Up

- Oracle 12c support for JSON is timely and useful
- JSON is coming to an application near you soon
- JSON is:
  - Most-common mechanism for interacting with AJAX
  - Becoming most-common mechanism for web service data (especially REST/HTTP API based services)
  - Cornerstone of several "big data" data stores
- Take the time to learn JSON, JSON syntax, and Oracle's implementation

www.rmoug.org

COLLABORATE 16 – IOUG Forum

April 10 – 14, 2016

Mandalay Bay
Las Vegas, NV

ODTUG Kscope16

SAVE THE DATE
CHICAGO, ILLINOIS
JUNE 26-30
www.kscope16.com

Please Complete Session Evaluations

Using JSON in Oracle 12c

To contact the author:

John King

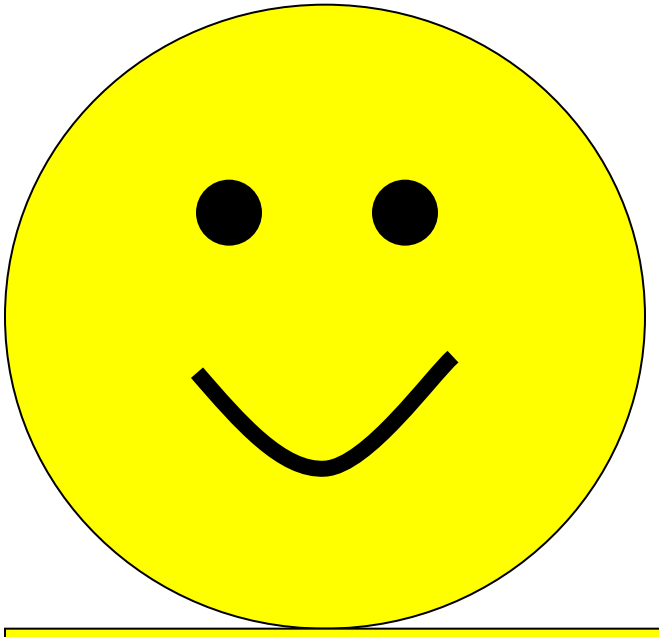
King Training Resources

P. O. Box 1780

Scottsdale, AZ 85252 USA

1.800.252.0652 - 1.303.798.5727

Email: john@kingtraining.com



Thanks for your attention!

Today's slides and examples are on the web:

http://www.kingtraining.com

- End