ORACLE

JSON in Oracle Database: common use cases and best practices

Presentation at hroug, October, 17th 2019

Beda Hammerschmidt

Twitter: bch_t

Consulting (Coding) Member of Technical Staff
Oracle Database, JSON support

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

What's JSON?

```
{
  "firstName":"John",
  "lastName":"Smith",
  "age":25,
  "address":{
    "street":"21 2nd Street“,"city":"New York",
    "state":"NY","postalCode":"10021",
    "isBusiness":false
  },
  "phoneNumbers":[
    {"type":"home","number":"212 555-1234"},
    {"type":"mobile","number":"646 555-4567"}
  ],
  "bankruptcies":null,
  ”lastUpdated":"2019-05-13T13:03:35+0000"
}
```


What's JSON?


```
{
  "firstName":"John",
  "lastName":"Smith",
  "age":25,
  "address":{
    "street":"21 2nd Street“,"city":"New York",
    "state":"NY","postalCode":"10021",
    "isBusiness":false
  },
  "phoneNumbers":[
    {"type":"home","number":"212 555-1234"},
    {"type":"mobile","number":"646 555-4567"}
  ],
  "bankruptcies":null,
  ”lastUpdated":"2019-05-13T13:03:35+0000"
}
```

JSON Support in Oracle Database

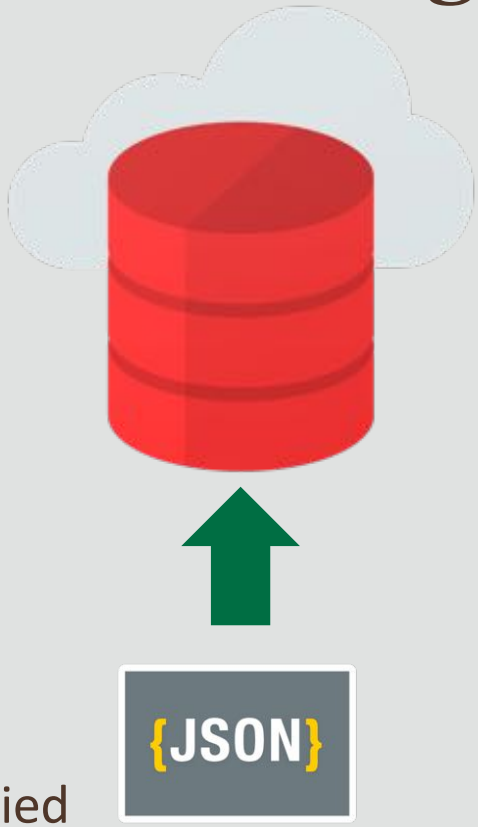
JSON storage

```
create table emp(
        empno   number,
        first   varchar2(50),
        last    varchar2(50),
        salary  number,
        flex    CLOB,
        check (flex IS JSON));

insert into emp values(1, 'John', 'Smith',
  2000, '{skills:["Java", "C", "C++"]}');

insert into emp values(2, 'Amanda', 'Jones',
  2500, '{skills:["JavaScript", "nodeJS"],
          numPatents:5}');
```


Common use cases for JSON in Oracle Database

Use Case: JSON Ingestion and Processing

- JSON import to relational processing stack
  - Feed existing application with JSON data
  - Examples: Google Clickstream, travel bookings, tax-filing, policy data, social media feeds,…
  - Use-cases: Analytical queries, Reporting
  - Often no control over JSON structure/schema. Sometimes originating from XML
  - Often JSON is not only imported but also stored and queried (schema-less storage)



JSON Support in the Oracle database

```
insert into emp values(1, 'John', 'Smith',
  2000, '{skills:["Java", "C", "C++"]}');

insert into emp values(2, 'Amanda', 'Jones',
  2500, '{skills:["JavaScript", "nodeJS"],
          numPatents:5}');
```

```
select e.flex.skills,
       e.flex.numPatents.number()
from emp e
where e.flex.skills like '%Java%';

SKILLS                    NUMPATENTS
--------------------------------
["Java","C","C++"]          (null)
["JavaScript","nodeJS"]          5
```

JSON Support in the Oracle database

```
insert into emp values(1, 'John', 'Smith',
  2000, '{skills:["Java", "C", "C++"]}');

insert into emp values(2, 'Amanda
  2500, '{skills:["JavaScript", "
          numPatents:5}');
```

```
select JSON_QUERY(e.flex, '$.skills[0,1]'
                                WITH ARRAY WRAPPER)
from emp e where
JSON_EXISTS(e.flex,'$.skills?(@ == "Java")');

JSON_QUERY(SKILLS..
--------------------
["Java","C"]
```

```
select e.flex.skills,
       e.flex.numPatents.number()
from emp e
where e.flex.skills like '%Java%';

SKILLS                      NUMPATENTS
--------------------------------------
["Java","C","C++"]             (null)
["JavaScript",”nodeJS"]             5
```

JSON_TABLE: 'flatten' the JSON data

```
{
 "firstName":"John",
 "lastName":"Smith",
 "age":25,
 "address":{
   "street":"21 2nd Street",
    "city":"New York",
   "state":"NY",
  "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptcies":null,
 "lastUpdated":"2019-05-13T13:03:35"
}
```

```
select id, jt.*
from custData, JSON_TABLE(jcol, '$' columns (
  "first"              path '$.firstName',
  "age"   number       path '$.age',
  "state" varchar2(2) path '$.address.state',
   NESTED PATH '$.phoneNumbers[*] columns(
      "phone" path '$.number')
 )) jt;

ID first age state  phone
-------------------------
1  John  25  NY     212-555-1234
1  John  25  NY     646-555-4567
```

JSON_TABLE: 'flatten' the JSON data

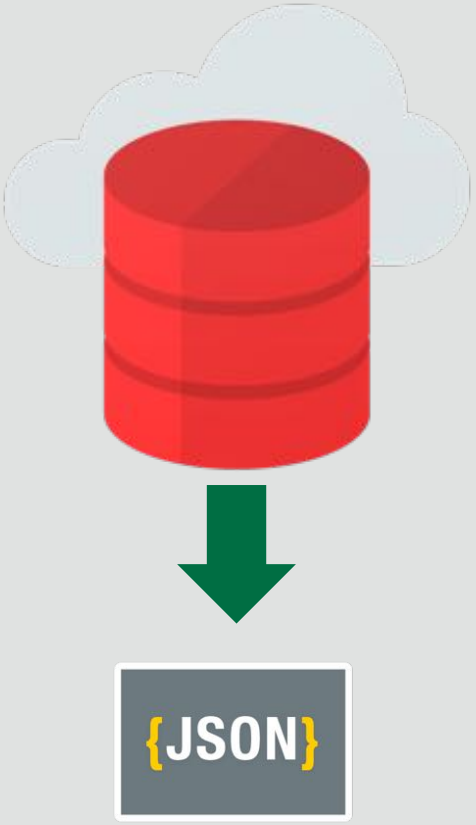
```
{
 "firstName":"John",
 "lastName":"Smith",
 "age":25,
 "address":{
   "street":"21 2nd Street“,
    "city":"New York",
   "state":"NY",
  "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptcies":null,
 "lastUpdated":"2019-05-13T13:03:35"
}
```

NEW

```
select id, jt.*
from custData NESTED jcol columns (
   firstName,
   age NUMBER,
   ”state" PATH '$.address.state',
   NESTED phoneNumbers[*] columns(
       "number"
) jt;

ID firstName age state  number
-------------------------
1  John      25  NY     212-555-1234
1  John      25  NY     646-555-4567
```

Use Case: JSON Generation

- Export of (relational) data as JSON
  - Feed data to other systems
  - modern JavaScript UI on legacy app
- Message generation
  - Example: RAC autonomous health framework generate node status information as JSON
  - Sometimes PL/SQL is used if message generation requires procedural logic



JSON Generation

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
select JSON_OBJECT(
      'id'   VALUE empno,
      'name' VALUE ename )
from emp;
```

```
JSON_OBJECT(…
-------------------------
{"id":7669, "name":"SMITH"}
{"id":7499, "name":"ALLEN"}
{"id":7521, "name":"WARD"}
{"id":7566, "name":"JONES"}
```


JSON Generation

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD						20
7934	MILLER						10

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
select
JSON_OBJECTAGG(dname VALUE
  select
  JSON_ARRAYAGG(JSON {ename})
  from emp e
  where e.deptno = d.deptno
) from dept d;

JSON_OBJECTAGG(…
------------------------
{"ACCOUNTING":[
      {"eName":"CLARK"},
      {"eName":"KING"},
      {"eName": "MILLER"}
    ],
 "RESEARCH":[
      {"eName": "SMITH"},
      {"eName": "JONES"},
```

Use Case: Procedural Logic on JSON data

```
create or replace function score(skills CLOB) return number is
  skillsArr JSON_ARRAY_T;
  skill     varchar2(50);
  score     number := 0;
begin
  skillsArr := JSON_ARRAY_T(skills);
  for i in 0 .. skillsArr.get_size loop
    skill := skillsArr.get_string(i);
    if (skill like '%JSON%') then
      score := score + 2;
    else
      score := score + 1;
    end if;
  end loop;
  return score;
end;
/
```

Use Case: Schema Discovery

- JSON data typically has no formal schema definition
  - Schema implicitly defined by applications
  - Hard to identify schema by looking at instances
  - Schema changes hard to track
- JSON-Dataguide
  - derive JSON schema from JSON data (sampling)
  - Use schema to auto-build views and virtual columns
  - Track schema schanges
- Relational: Schema first – data later
- JSON: Data first - schema later

JSON Support in the Oracle database

```
{
 "firstName":"John",
 "lastName":"Smith”,
 "age":25,
 "address":{
   "street":"21 2nd Street“,
    "city":"New York",
   "state":"NY",
  "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptcies":null,
 "lastUpdated":"2019-05-13T13:03:35"
}
```

```
select JSON_DATAGUIDE(jcol) from custData;

[{
        "o:path": "$",
        "type": "object",
        "o:length": 1
}, {
        "o:path": "$.age",
        "type": "number",
        "o:length": 2
}, {
        "o:path": "$.address",
        "type": "object",
        "o:length": 1
}, {
        "o:path": "$.address.city",
        "type": "string",
        "o:length": 8
},
```

JSON Support in the Oracle database

```
{
 "firstName":"John",
 "lastName":"Smith”,
 "age":25,
 "address":{
   "street":"21 2nd Street“,
    "city":"New York",
   "state":"NY",
  "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptcies":null,
 "lastUpdated":"2019-05-13T13:03:35"
}
```

```
select JSON_DATAGUIDE(jcol,
  dbms_json.FORMAT_HIERARCHICAL) from custData;
{
  "type": "object",
  "o:length": 1,
  "properties": {
       "age": {
           "type": "number",
           "o:length": 2,
           "o:preferred_column_name": "age"
         },
         "address": {
              "type": "object",
              "o:length": 1,
              "o:preferred_column_name": "address",
              "properties": {
                 "city": {
                 "type": "string",
                 "o:length": 8,
```

JSON Support in the Oracle database

```
{
 "firstName":"John",
 "lastName":"Smith”,
 "age":25,
 "address":{
   "street":"21 2nd Street“,
    "city":"New York",
   "state":"NY",
   "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptcies":null,
 "lastUpdated":"2019-05-13T13:03:35"
}
```

```
declare
  dg clob;
begin
  select json_dataguide(jcol,
         dbms_json.FORMAT_HIERARCHICAL)
  into dg from t;
  dbms_json.create_view('auto_view', 't', 'jcol', dg);
end;
/

PL/SQL procedure successfully completed

desc auto_view;
```


JSON Support in the Oracle database

```
{
 "firstName":"John",
 "lastName":"Smith”,
 "age":25,
 "address":{
   "street":"21 2nd Street“,
    "city":"New York",
   "state":"NY",
   "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptcies":null,
 "lastUpdated":"2019-05-13T13:03:35"
}
```

```
declare
  dg clob;
begin
  select json_dataguide(jcol,
         dbms_json.FOR
  into dg from t;
  dbms_json.create_vie
end;
/

PL/SQL procedure succe

desc auto_view;
```

```
Name          Null? Type
------------- ----- ------------
age                 NUMBER
city                VARCHAR2(8)
state               VARCHAR2(2)
street              VARCHAR2(16)
isBusiness          VARCHAR2(8)
postalCode          VARCHAR2(8)
lastName            VARCHAR2(8)
firstName           VARCHAR2(4)
lastUpdated         VARCHAR2(32)
type                VARCHAR2(8)
number              VARCHAR2(16)
```

JSON Support in the Oracle database

```
{
 "firstName":"John",
 "lastName":"Smith”,
 "age":25,
 "address":{
   "street":"21 2nd Street“,
    "city":"New York",
   "state":"NY",
   "postalCode":"10021",
   "isBusiness":false
 },
 "phoneNumbers":[
   {"type":"home",
     "number":"212-555-1234"},
   {"type":"mobile",
     "number":"646-555-4567"}
 ],
 "bankruptci
 "lastUpdate
}
```

```
declare
  dg clob;
begin
  select json_dataguide(jcol,
         dbms_json.FOR
  into dg from t;
  dbms_json.create_vie
end;
/

PL/SQL procedure succe

desc auto_view;
```

```
Name          Null? Type
------------- ----- ------------
age                 NUMBER
city                VARCHAR2(8)
state               VARCHAR2(2)
street              VARCHAR2(16)
isBusiness          VARCHAR2(8)
postalCode          VARCHAR2(8)
lastName            VARCHAR2(8)
firstName           VARCHAR2(4)
lastUpdated         VARCHAR2(32)
type                VARCHAR2(8)
number              VARCHAR2(16)
```

```
select "postalCode", "city"
from auto_view
order by "state";
```

Use Case: Flex fields, sparse columns

- Flex fields (hybrid storage)
  - Core of application is relational but ‘new’ fields are in a (flat) JSON column
  - Schema flexibility, easier/faster to extend application
  - Data augmentation by third parties: fields are unknown and never defined,
    - JSON Search index (index all values)

- Sparse columns
  - Many optional attributes
  - Hitting 1000 column limit?
  - JSON_TABLE views, to select relevant values as columns

Performance, etc

- Partitioning (partition pruning during query)
- Exadata Smart Scans (predicates pushed down to storage layer)
- In-Memory JSON (SIMD scans, binary OSON format)
- Indexing
  - Index a specific ‘path’, e.g. $.customer.id
    - B-Tree Index: log(n)
  - Index entire JSON document (JSON Search Index)
    - Variant of context index
    - Posting lists for values
- Keep JSON data concise (omit null values, short key names)

Simple Oracle Document Access (SODA)

Document Store Model mapped to relational model

- A **document** is a JSON (or binary) instance
  - Document is identified by a key (get, update, delete by key)
- Store documents inside a *collection*
  - Collection stores similar documents
  - Collection is identified by name
- One or more collections reside in a *database*
  - Application connects to database

26

Document Store Model mapped to relational model

- A **document** is a JSON (or binary) instance
  - Document is identified by a key (get, update, delete by key)
- Store documents inside a *collection*
  - Collection stores similar documents
  - Collection is identified by name
- One or more collections reside in a *database*
  - Application connects to database

Row

Table

Schema

KEY	JSON_DOC	VERSION	CREATED_ON	LAST_MODIFIED
3C990A8D39…	{“name” : “Alex”}	75611B292…	2018-09-12T00:00:00Z	2018-09-12T00:00:00Z

Creating and dropping collections

Node.js

```
conn = await oracledb.getConnection(…);
db = conn.getSodaDatabase();
coll = await db.createCollection(“sweets");
await coll.drop();
```

Python

```
conn = cx_Oracle.connect(…);
db =  conn.getSodaDatabase();
coll = db.createCollection(“sweets");
coll.drop();
```

Java

```
OracleClient cl = new OracleRDBMSClient();
db = cl.getDatabase(jdbcConn);
OracleCollection coll =
db.admin().createCollection(“sweets");
coll.admin().drop();
```

PL/SQL

```
coll := dbms_soda.create_collection(‘sweets’);


select dbms_soda.drop_collection(‘sweets’) from
dual;
```


28

Inserting, finding, updating documents

Node.js (*other implementations have similar syntax*)

- `resultDoc = await coll.insertOneAndGet({name : “Alexander”});`
- `key = resultDoc.key;`

- `col.find().key(“k1”);`
- `col.find().key(“k1”). replaceOne(newDoc);`
- `col.find().filter({“name” : “alex”});`
- `col.find().filter({“name” : “alex”}).remove();`
- `col.find().filter({“name” : “chris”}).skip(10).limit(10).getCursor();`

QBEs can be sophisticated!

```
{ "$query" : {
                "empno" : { "$lte": 10041 },
                "name" : { "$startsWith" : "Melissa" },
                "salary" : { "$gt" : 200000 },
                "title" : { "$contains" : "President" },
              },
  "$orderby" : [
                 {"path" : "salary", "order" : "asc"},
                 {"path" : "department.name", "order" : "desc"}
                ]
}
```


30

QBE rewritten to SQL

```
SELECT "ID", "JSON_DOCUMENT" FROM "Employees" coll
WHERE JSON_Exists(coll."JSON_DOCUMENT",
                  '$?(@.empno <= $B0 &&
                      @.salary > $B1 &&
                      @.name starts with $B2)'
                  PASSING 10041 AS "B0", 200000 AS "B1", 'Melissa' AS
"B2")
AND JSON_TextContains(coll."JSON_DOCUMENT",
                      '$.title', 'President')
ORDER BY JSON_Value(coll."JSON_DOCUMENT", '$.salary') ASC,
         JSON_Value(coll."JSON_DOCUMENT", '$.department.name') DESC;
```

Conclusion: JSON storage and easy access Api(SODA)

Simple NoSQL Development experience

ORACLE

JSON in Oracle Database: common use cases and best practices

Presentation at hroug, October, 17th 2019

Beda Hammerschmidt

Twitter: bch_t

Consulting (Coding) Member of Technical Staff
Oracle Database, JSON support