

ORACLE®

Oracle Database 12c JSON Document Store

New Features in 12.1.0.2 for Application Developers

Mark D Drake
Manager, Product Management

Safe Harbor Statement

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

- 1 **Schema-less Development and Data Management**
- 2 Introduction to JSON
- 3 Oracle 12c as JSON Document Store
- 4 Oracle Rest Services for JSON
- 5 Simple Oracle Document API (SODA)
- 6 Query, Reporting and Analysis on JSON documents

Schema-less Development

A.KA. “Schema-on-Read”, “Schema-Later” or “Schema-Never”

- Extremely customizable applications
 - Changes to the application data model do not require changes to the database schema
 - Different versions of an application can share a common data store
 - Zero-downtime application upgrades
 - Data structures that are not (completely) known until run-time
- Application data stored as documents rather than in tables
 - Using JSON or XML documents shields the database from knowledge of the data model
- All about flexibility
 - Application data model is not baked into the storage model

The drivers for No-SQL document stores

- Schema-less Development requires a document based persistence model.
- JSON has become the preferred mechanism for document persistence
- JSON document stores are perceived as being easy to install
- JSON document stores are perceived as being easy to use
- JSON document stores are perceived as having a low TCO

Schema-less data management requirements

- Ability to store data without requiring a Schema
 - Store semi-structured data in its native (aggregated) form
- Ability to query data without knowledge of Schema
 - Query Semi-Structured data using a semi-structured query language
- Ability to index data with knowledge of Schema

Impact on traditional RDBMS systems

- Increased adoption of document-centric development means
 - Managing large volumes of semi-structured data, e.g. JSON & XML.
 - Semi-structured data becoming mission critical
- Relational model is not going to disappear
 - Relational data and semi-structured data need to coexist
 - Need to avoid stove-piped data management at all costs

Schema-less data management and the Oracle RDBMS

- Organizations need a single infrastructure that provides
 - Declarative techniques for persisting, querying and indexing any kind of data
 - A platform equally optimized for any kind of data
 - A query language that works with any kind of data
 - Support for common data management tasks
 - Transactions, Partitioning, Replication, Security, Scalability, Availability

Program Agenda

- 1 Schema-less Development and Data Management
- 2 **Introduction to JSON**
- 3 Oracle 12c as JSON Document Store
- 4 Oracle Rest Services for JSON
- 5 Simple Oracle Document API (SODA)
- 6 Query, Reporting and Analysis on JSON documents

What is JSON and why is it popular ?

- JSON – JavaScript Object Notation
 - A method of representing the state of an application object
- Default serialization for preserving browser state
 - Browser based applications use JavaScript and JavaScript objects
- Supported by many public Rest interfaces
 - Facebook API, Google Geocoder, Twitter API
- Growing influence on server side coding (Node.js)
- Easier to use than XML, particularly when working with JavaScript
 - Perception that is more efficient / Lightweight

Example JSON document

```
{
  "PONumber" : 1600,
  "Reference" : "ABULL-20140421",
  "Requestor" : "Alexis Bull",
  "User" : "ABULL",
  "CostCenter" : "A50",
  "ShippingInstructions" : {
    "name" : "Alexis Bull",
    "Address" : { ... },
    "Phone" : [ ... ]
  },
  "Special Instructions" : null,
  "AllowPartialShipment" : true,
  "LineItems" : [{
    "ItemNumber" : 1,
    "Part" : {
      "Description" : "One Magic Christmas",
      "UnitPrice" : 19.95,
      "UPCCode" : 13131092899
    },
    "Quantity" : 9
  },
  { ... }
  ]
}
```

Application Persistence

- Developers want to preserve application state as JSON
 - Store and retrieve JSON objects using simple key/value techniques
 - Leverage easy to use document level API's
 - Little or no thought given to the impact of JSON data model on reporting and analysis
- Common objections to using an RDBMS
 - Avoiding the overhead of mapping between JSON and traditional storage models
 - Need to involve DBA/IT when the data model changes
- JSON is the data model for a number of No-SQL data stores
 - couchDB, MongoDB

Program Agenda

- 1 Schema-less Development and Data Management
- 2 Introduction to JSON
- 3 **Oracle 12c as JSON Document Store**
- 4 Oracle Rest Services for JSON
- 5 Simple Oracle Document API (SODA)
- 6 Query, Reporting and Analysis on JSON documents

Supporting 'schema-less' development on Oracle Database

- Store and manage JSON documents in Oracle Database
- Deliver NO-SQL application development experience
 - Simple document centric API's for storing and accessing JSON
 - QBE Query capability
 - No need to learn SQL
- Allow “next generation” application development to leverage the Oracle Database
- Provide Consistent, standard database environment for IT
- Delivered as part of 12.1.0.2.0 patch set

Oracle RDBMS as a JSON document Store

- JSON documents stored in the database using existing data types
 - VARCHAR2, CLOB and BLOB for character mode JSON
- External JSON data sources accessible through external tables
 - JSON in file system (also HDFS) can be accessed via external tables

Value Added capabilities for Application Developers










- All existing Oracle features work with JSON Data
- Enables path based queries on JSON documents stored in database
 - Support JSON Path language and JSON Path expressions.
- Enable SQL based reporting and Analytics
 - SQL/JSON and JSON path language provide relational access to JSON.
 - Relational views based on SQL/JSON enable SQL operations on JSON content.
- Powerful and Flexible indexing of JSON documents
 - JSON path based Functional Indexes
 - Materialized Views; Full document indexing with inverted JSON index

Oracle as JSON document store

- Oracle Multitenant makes Oracle easy to install (For Developers).
 - Multitenant enables the deployment of self-service applications that can spin up a dedicated pluggable database on demand
- New, document centric APIs, allow developers to work with JSON content without having to learn SQL
 - RestAPI accessible from all common programming and scripting environments
 - Dedicated document centric API's for Java and other environments are forthcoming
- Simple intuitive API enables operations on collections and documents
 - Eliminates need for DBA support to create and deploy applications

Oracle Database 12c for the Developer

Supporting all major development environments and API's

		Native (SQL) API	Document Store API
Java		✓	H2 2014
.NET		✓	Use REST API
Node.js		✓	H2 2014
REST (ORDS)		✓	DB 12.1.0.2
Ruby		✓	Use REST API
Python		✓	Use REST API
PHP		✓	Use REST API
R		✓	Use REST API
Perl		✓	Use REST API

Use new document-store API's to build applications without writing SQL

Existing SQL API's can access JSON documents as well

Oracle as JSON document store

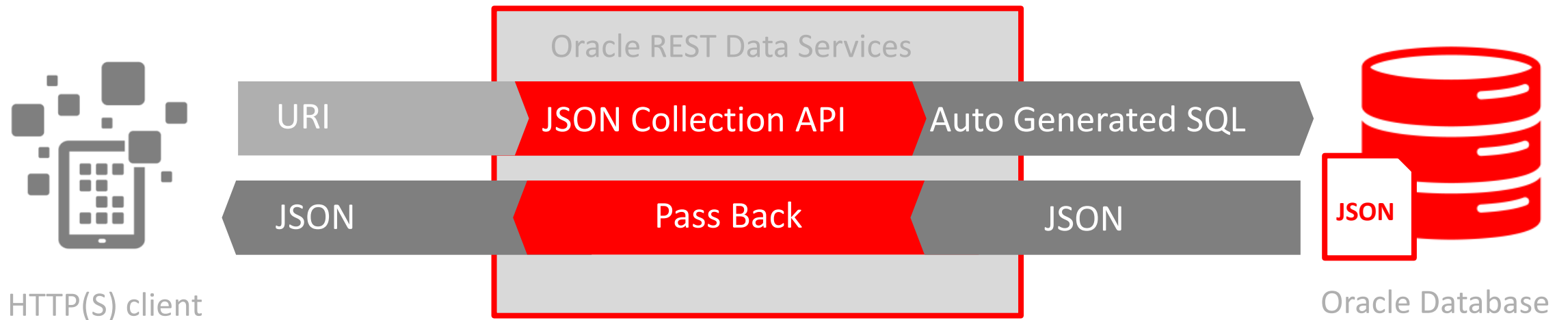
- Oracle delivers a lower TCO than stove-piped, heterogeneous data stores
 - Multitenant significantly reduces the cost of deploying Oracle Database
 - Using Oracle as a JSON document store eliminates cost and complexity.

Program Agenda

- 1 Schema-less Development and Data Management
- 2 Introduction to JSON
- 3 Oracle 12c as JSON Document Store
- 4 **Oracle Rest Services for JSON**
- 5 Simple Oracle Document API (SODA)
- 6 Query, Reporting and Analysis on JSON documents

Oracle REST Data Services

Schema-less development using JSON Collection API (12.1.0.2)



- Data stored in Oracle Database as JSON documents
- App Developer make standard HTTP(S) calls to JSON Collection APIs

REST based API for JSON documents

- Rest based API for managing collections and operations on documents
 - Standard REST based model
 - URI patterns mapped to operations on collections managed by the database
- Based on HTTP(s)
- Can be invoked from almost any programming language
- Implemented in JAVA
 - Can run in any container supported by Oracle REST Data Services, or as a JAVA servlet under the XMLDB HTTP Listener

REST based API for JSON documents

- Simple well understood model
- CRUD operations are mapped to HTTP Verbs
 - Create / Update : PUT / POST
 - Retrieve : GET
 - Delete : DELETE
 - QBE, Bulk Update, Utility functions : POST
- Stateless

Oracle REST API

GET /rest/schema	List all collections in a schema
GET /rest/schema/collection	Get all objects in collection
GET /rest/schema/collection/id	Get specific object in collection
PUT /rest/schema/collection	Create a collection if necessary
PUT /rest/schema/collection/id	Update object with id
POST /rest/schema/collection	Insert object into collection
POST /rest/schema/coll?action=query	Find objects matching filter in body

JSON Rest Services: Using PUT to Insert a new record

```
PUT /my_database/my_schema/customers HTTP/1.0
Content-Type: application/json
Body:
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021",
    "isBusiness": false },
  "phoneNumbers": [
    {"type": "home",
     "number": "212 555-1234" },
    {"type": "fax",
     "number": "646 555-4567" } ]
}
```

JSON Rest Services : Invoking Query by Example using POST

```
POST
/my_database/my_schema/customers?command=query HTTP/1.0
Content-Type: application/json
Body:
{
  "company" : "IBM",
  "$startsWith" : {"department": "S"},
  "$or" : [
    {"$startsWith" : {"name": "Melissa"}},
    {"$gte" : {"salary" : 10000}}
  ]
}
```



Generated SQL

```
select "JKEY","JVALUE",
       to_char("LAST_MODIFIED",'YYYY-MM-DD"T"HH24:MI:SS.FF'),
       "VERSION"
from "SCOTT"."EMPLOYEES"
where JSON_EXISTS(
       "JVALUE",
       '$?( $.company == $B0 && $.department starts with $B1
       && ( $.name starts with $B2 || $.salary >= $B3 ) )'
       PASSING 'IBM' AS "B0" ,
              'S' AS "B1" ,
              'Melissa' AS "B2" ,
              10000 AS "B3"
       )
```

Using REST API from Javascript

XMLHttpRequest

- Javascript routines embedded in an HTML Page can use the XMLHttpRequest object to interact with the REST Service
- Based on the W3C XMLHttpRequest specification
 - <http://www.w3.org/TR/XMLHttpRequest/>
- Allows browsers to interact with remote services without refreshing an entire page
- Asynchronous operations prevent browser from blocking while waiting for a response

Inserting a Document (Server assigned ID's)

POST : /servlet/schema/collectionName

```
function postDocument(URL,payload,callback) {
    var serializedPayload = JSON.stringify(payload);
    var XHR = new XMLHttpRequest();
    XHR.open ("POST", URL, true);
    XHR.setRequestHeader ("Content-type", "application/json");
    XHR.onreadystatechange = function() {
        if (XHR.readyState==4) {
            callback (XHR,URL);
        }
    };
    XHR.send(serializedPayload);
}
```

Inserting a Document (Server assigned ID's)

Response

```
{ "items": [  
  { "id"           : "94FC8B4AC80A421AB5C94BABED379C42",  
    "etag"        : "89B3A423EA7D4435BCBCF90AFD36A9A8EAAD571A3F3CFE3DBD95988665BC586D",  
    "lastModified" : "2014-09-29T18:56:34.131000",  
    "created"      : "2014-09-29T18:56:34.131000",  
    "value"        : null  
  }  
],  
  "hasMore":false,  
  "count":1  
}
```

Fetching a Document (Server assigned ID's)

GET: /servlet/schema/collectionName/id

```
function getDocument(URL, callback) {  
    var XHR = new XMLHttpRequest();  
    XHR.open ("GET", URL, true);  
    XHR.onreadystatechange = function() {  
        if (XHR.readyState==4) {  
            callback (XHR, URL);  
        }  
    };  
  
    XHR.send (null);  
}
```

Searching for Documents

POST : /servlet/schema/collectionName

```
function postDocument(URL,qbe,callback) {
    URL = URL + "?action=query"
    var serializedPayload = JSON.stringify(qbe);
    var XHR = new XMLHttpRequest();
    XHR.open ("POST", URL, true);
    XHR.setRequestHeader("Content-type","application/json");
    XHR.onreadystatechange = function() {
        if (XHR.readyState==4) {
            callback(XHR,URL);
        }
    };
    XHR.send(serializedPayload);
}
```


Searching for Documents

Response

```
{
  "items": [
    {
      "id": "16D6AC1899794194A843DD29B84CA4AA",
      "etag": "C4B7FDDEA227F023F36F8296E3657A934CE537CF42CEF66B95D8163BAB1ADA0E",
      "lastModified": "2014-09-29T16:26:57.141000",
      "created": "2014-09-29T16:26:57.141000",
      "value": {
        "PONumber": 3,...
      }
    }, ...
  ],
  "hasMore": false,
  "count": 9
}
```

Program Agenda

- 1 Schema-less Development and Data Management
- 2 Introduction to JSON
- 3 Oracle 12c as JSON Document Store
- 4 Oracle Rest Services for JSON
- 5 **Simple Oracle Document API (SODA)**
- 6 Query, Reporting and Analysis on JSON documents

Java API for JSON

- Support similar degree of functionality to Rest Services
 - Collection Management
 - CRUD operations on JSON documents
 - Query-by-Example for document searching
 - Utility and control functions
- Java implementations of JSON Path processor and JSON Parser
- Much simpler than JDBC for working with collections of JSON documents stored in Oracle Database

Sample SODA code

Creating a Collection, Inserting a Document and getting the ID and Version

```
// Create a Connection
OracleRDBMSClient client = new OracleRDBMSClient();
OracleDatabase database = client.getDatabase(conn);

// Now create a collection
OracleCollection collection = database.getDatabaseAdmin().createCollection("MyCollection");

// Create a document
OracleDocument document = database.createDocumentFromString("{ \"name\" : \"Alexander\" }");

// Next, insert it into the collection
OracleDocument insertedDocument = collection.insertAndGet(document);

// Get the key of the inserted document
String key = insertedDocument.getKey();

// Get the version of the inserted document
String version = insertedDocument.getVersion();
```

Program Agenda

- 1 Schema-less Development and Data Management
- 2 Introduction to JSON
- 3 Oracle 12c as JSON Document Store
- 4 Oracle Rest Services for JSON
- 5 Simple Oracle Document API (SODA)
- 6 **Query, Reporting and Analysis on JSON documents**

JSON Support in Oracle Database

Flexible Application Development + Powerful SQL Analytics

Oracle Database 12c



Data accessed
via RESTful
service or native
API's



Data persisted in database
In JSON



Data analyzed via SQL

Oracle JSON SQL Capabilities

- JSON content is accessible from SQL via new operators
 - JSON_VALUE, JSON_TABLE, JSON_EXISTS, IS JSON
- JSON operators use JSON Path language to navigate JSON objects
- Syntax developed in conjunction with IBM

JSON Validity

IS JSON predicate

- Check constraint guarantees that values are valid JSON documents
- IS [NOT] JSON predicate
 - Returns TRUE if column value is JSON, FALSE otherwise
 - Full parse of the data while validating syntax
 - Tolerant and strict modes
 - JSON accepted by operators vs strict compliance to syntax definition
- Use to ensure that the documents stored in a column are valid JSON

JSON DDL Operations

```
create table J_PURCHASEORDER (  
  ID          RAW(16) NOT NULL,  
  DATE_LOADED  TIMESTAMP(6) WITH TIME ZONE,  
  PO_DOCUMENT CLOB CHECK (PO_DOCUMENT IS JSON)  
)
```

```
insert into J_PURCHASEORDER values('0x1','{Invalid JSON Text}');
```

ERROR at line 1:
ORA-02290: check constraint (DEMO.IS_VALID_JSON) violated

JSON : DML Operations

```
SQL> insert into J_PURCHASEORDER
 2 select SYS_GUID(), SYSTIMESTAMP, JSON_DOCUMENT
 3 from STAGING_TABLE
 4 where JSON_DOCUMENT IS JSON
 5 /
```

```
SQL> delete from STAGING_TABLE
 2 where DOCUMENT IS NOT JSON;
```

JSON : Queries – Simplified Syntax

```
SQL> select j.PO_DOCUMENT
2   from J_PURCHASEORDER j
3  where j.PO_DOCUMENT.PONumber = 1600
4 /
```

```
SQL> select j.PO_DOCUMENT.CostCenter, count(*)
2   from J_PURCHASEORDER j
3  group by j.PO_DOCUMENT.CostCenter
4  order by j.PO_DOCUMENT.CostCenter
5 /
```

```
SQL> select j.PO_DOCUMENT.ShippingInstructions.Address
2   from J_PURCHASEORDER j
3  where j.PO_DOCUMENT.PONumber = 1600
4 /
```

JSON Path Expressions

- Similar role to XPATH in XML
- Syntactically similar to Java Script (. and [])
- Predicate support forth coming
- Lax mode does not differentiate between singletons and collections
 - Similar semantics to W3C XPath expressions and MongoDB
- JSON Path expressions can be evaluated using streaming techniques
 - Does not require construction of an in-memory representation of document

JSON Path language

Path language for intra document navigation

- Compatible with Java Script
 - `$.phone[0]`
- Wildcards, Multi-Selects, Ranges
 - `$.phone[*]`, `$.phone[0,1 5 to 9]`
- Predicates
 - `$.address?($.zip > $zip)`
- SQL conversion functions usable in predicates
 - `?(to_date($.date) > $date)`

JSON Path language

Path language for intra document navigation

- Compatible with Java Script
 - `$.phone[0]`
- Wildcards, Multi-Selects, Ranges
 - `$.phone[*]`, `$.phone[0,1 5 to 9]`
- Predicates
 - `$.address?($.zip > $zip)`
- SQL conversion functions usable in predicates
 - `?(to_date($.date) > $date)`

SQL/JSON operators : JSON_VALUE

- Extract exactly one scalar value from a document
 - Value identified by a JSON Path expression
- Typically used in the select list or the where clause
- Allows value to be cast to a SQL data type
- Allows specification of error handling
 - NULL on ERROR, DEFAULT on ERROR, ERROR on ERROR
- Also used to create functional indexes on a JSON document

JSON_VALUE()

```
SQL> select JSON_VALUE(  
2         PO_DOCUMENT,  
2         '$.LineItems[0].Part.UnitPrice'  
3         returning NUMBER(5,3)  
4     )  
5     from J_PURCHASEORDER p  
6     where JSON_VALUE(  
7         PO_DOCUMENT,  
8         '$.PONumber'  
9         returning NUMBER(10)  
10    ) = 1600  
11 /
```


SQL/JSON operators : JSON_QUERY

- Extract JSON fragment from JSON document
 - Fragment identified by a JSON Path expression
 - Fragment can be an ARRAY or an OBJECT
 - JSON data is contained in the specified SQL data type and form
- Allows specification of error handling
 - NULL on ERROR, DEFAULT on ERROR, ERROR on ERROR
 - (Conditional) Array Wrapping
- Typically used in the select list

JSON_QUERY()

```
SQL> select JSON_QUERY(  
2         PO_DOCUMENT,  
3         '$.LineItems'  
4     ) LINEITEMS  
5     from J_PURCHASEORDER p  
6     where JSON_VALUE(  
7         PO_DOCUMENT,  
8         '$.PONumber'  
9         returning NUMBER(10)  
10    ) = 1600  
11 /
```

SQL/JSON operators : JSON_EXISTS

- Return true / false depending on whether a JSON document contains a value that corresponds to a JSON Path expression
- Allow JSON Path expression to be used as a row filter.
 - Select rows based on content of JSON documents
- JSON Path expression can be a path
 - Support for Predicates in JSON Path expressions forthcoming
- Typically used in the where clause
- Can be used to create functional BITMAP indexes to test for PATH existence

JSON_EXISTS()

```
SQL> select count(*)  
2   from J_PURCHASEORDER  
3   where JSON_EXISTS(  
4           PO_DOCUMENT,  
5           '$.ShippingInstructions.Address.state'  
6           )  
7 /
```

SQL/JSON operators : JSON_TABLE

- Generate rows from a JSON Array
- Pivot properties / key values into columns
- Use Nested Path clause to process multi-level collections with a single JSON_TABLE operator.

JSON_TABLE()

```
SQL> select M.*
 2  from J_PURCHASEORDER p,
 3      JSON_TABLE(
 4      p.PO_DOCUMENT,
 5      '$'
 6      columns
 7      PO_NUMBER          NUMBER(10)          path '$.PONumber',
 8      REFERENCE          VARCHAR2(30 CHAR)    path '$.Reference',
 9      REQUESTOR          VARCHAR2(32 CHAR)    path '$.Requestor',
10      USERID             VARCHAR2(10 CHAR)    path '$.User',
11      COSTCENTER         VARCHAR2(16)        path '$.CostCenter'
12  ) M
13  where PO_NUMBER > 1600 and PO_Number < 1605
14  /
```

JSON_TABLE output

1 row output for each row in table

PO_NUMBER	REFERENCE	REQUSTOR	USERID	COSTCENTER
1600	ABULL-20140421	Alexis Bull	ABULL	A50
1601	ABULL-20140423	Alexis Bull	ABULL	A50
1602	ABULL-20140430	Alexis Bull	ABULL	A50
1603	KCHUNG-20141022	Kelly Chung	KCHUNG	A50
1604	LBISSOT-20141009	Laura Bissot	LBISSOT	A50

JSON_TABLE() with NESTED PATH clause

```
SQL> select D.*
 2  from J_PURCHASEORDER p,
 3      JSON_TABLE(
 4      p.PO_DOCUMENT,
 5      '$'
 6      columns(
 7      PO_NUMBER          NUMBER(10)          path '$.PONumber',
 8      NESTED PATH '$.LineItems[*]'
 9      columns(
10      ITEMNO             NUMBER(16)          path '$.ItemNumber',
11      UPCCODE            VARCHAR2(14 CHAR)    path '$.Part.UPCCode' ))
12  ) D
13  where PO_NUMBER = 1600 or PO_NUMBER = 1601
14  /
```


JSON_TABLE output

1 row output for each member of Lineltems array

PO_NUMBER	ITEMNO	UPCCODE
1600	1	13131092899
1600	2	85391628927
1601	1	97366003448
1601	2	43396050839
1601	3	13131119695
1601	4	25192032325

Indexing

- Known Query Patterns : JSON Path expression
 - Functional indexes using JSON_VALUE and, JSON_EXISTS
 - Materialized View using JSON_TABLE()
- Ad-hoc Query Strategy
 - Generalized Inverted Index
 - Based on Oracle's full text index (Oracle Text)
 - Support ad-hoc path, value and keyword query search using JSON Path expressions.

Summary

Oracle Database 12.1.0.2 supports 'document store' application

- Delivers same/similar application development experience as NoSQL databases
- Provides value-add capabilities: SQL on JSON, data integration
- Consistent, standard database environment for IT

ORACLE®