

ORACLE®

JSON Support Oracle Database 12c Release 2

Mark Drake
Manager, Product Management
Server Technology
October 20th 2016

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.

Agenda

- 1 Introduction to JSON
- 2 Oracle Database 12c as a Document Store
- 3 Storing and Querying JSON documents
- 4 Accelerating JSON Query performance
- 5 Understanding your JSON
- 6 Accessing relational data as JSON
- 7 Using Oracle Spatial with Geo-JSON content
- 8 Application Development with SODA
- 9 Summary

A woman with long brown hair and glasses, wearing a brown leather jacket and a blue patterned scarf, is sitting at a wooden table in a cafe. She is holding a black smartphone to her ear with her left hand and looking down at an open newspaper on the table with her right hand. The background is a blurred cafe interior with other tables and chairs, and a person is visible sitting at a table in the distance. The overall lighting is soft and natural.

Introduction to JSON

What is JSON and why is it popular ?

- JSON – JavaScript Object Notation
 - Simple, Lightweight and Easy to Use mechanism for persisting the state of an object
 - Language independent
- Default serialization for 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 Development with JSON

- Application objects are serialized as JSON and persisted as documents
- Primary access metaphor is Key/Value
 - Each document is associated with a Unique Key
 - The key is used to store, retrieve or update the entire document
- Developers gravitate towards simple key/value document stores
 - Provide simple, easy to use, document centric API's
 - Natural fit for popular RESTful development techniques
 - A number of NoSQL document databases, including MongoDB & CouchDB provide this functionality

A woman with long brown hair and glasses is sitting at a wooden table in a cafe or office setting. She is wearing a brown leather jacket over a blue patterned scarf. She is holding a black mobile phone to her ear with her left hand and looking down at a newspaper or magazine on the table with her right hand. The background is slightly blurred, showing other people and tables.

Oracle Database 12c as a Document Store

Strategy: Oracle Database as a Document Store

Core Capabilities for Document Workloads

Built on Foundation of Oracle Database

Full Support of Multi-Model and Hybrid Apps

Oracle 12c JSON document store

Core Capabilities for Document Workloads



Strategy: Oracle Database as a Document Store

Core Capabilities for Document Workloads

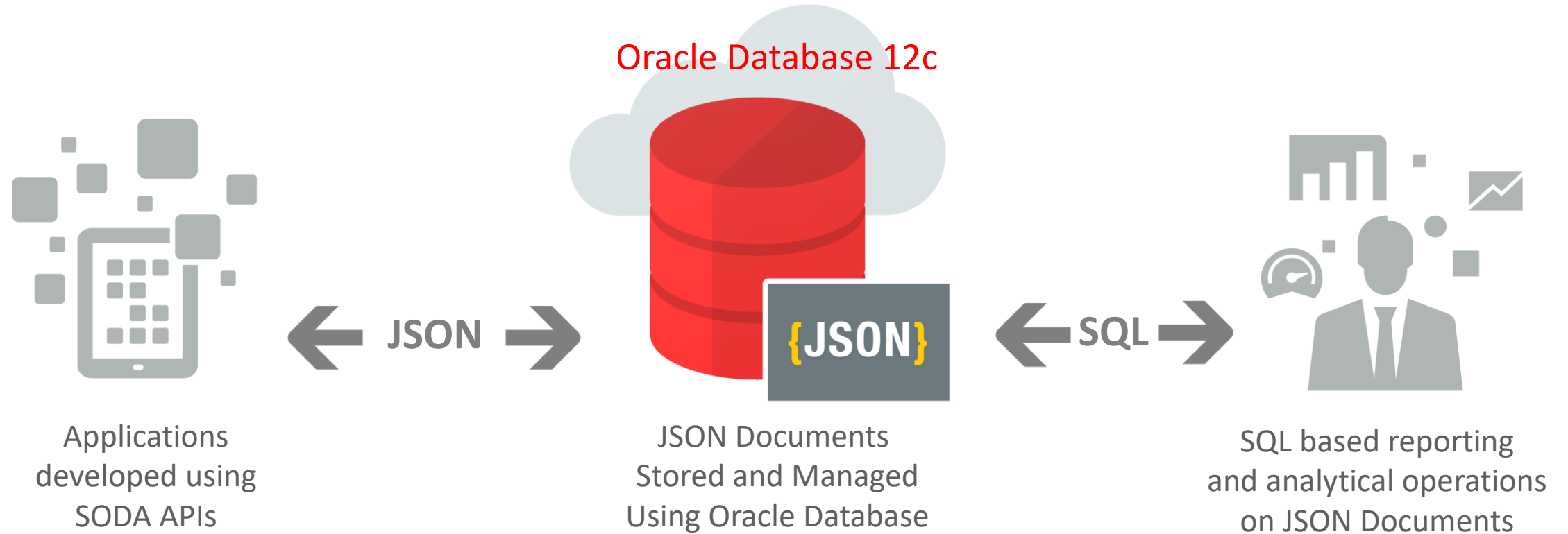
- Store and manage JSON and XML documents in Oracle Database
- Accessible via REST and all major programming languages
- Full query capabilities using JSON Path, XQuery and SQL
- Comprehensive, path-aware indexing
- No need to learn SQL or require DBA when developing applications
- Fits into the DevOPS paradigm

SODA: Simple Oracle Document Access

- A simple NoSQL-style API for Oracle
 - Collection Management: Create and drop collections
 - Document Management: CRUD (Create, Retrieve, Update and Delete) operations
 - List and Search: (Query-by-Example) operations on collections
 - Utility and Control: Bulk Insert, index management
- Developers can work with Oracle without learning SQL or requiring DBA support
 - Same development experience as pure-play document stores
- Currently available for Java and REST. Other versions are planned

Oracle 12c JSON document store

Built on Foundation of Oracle Database



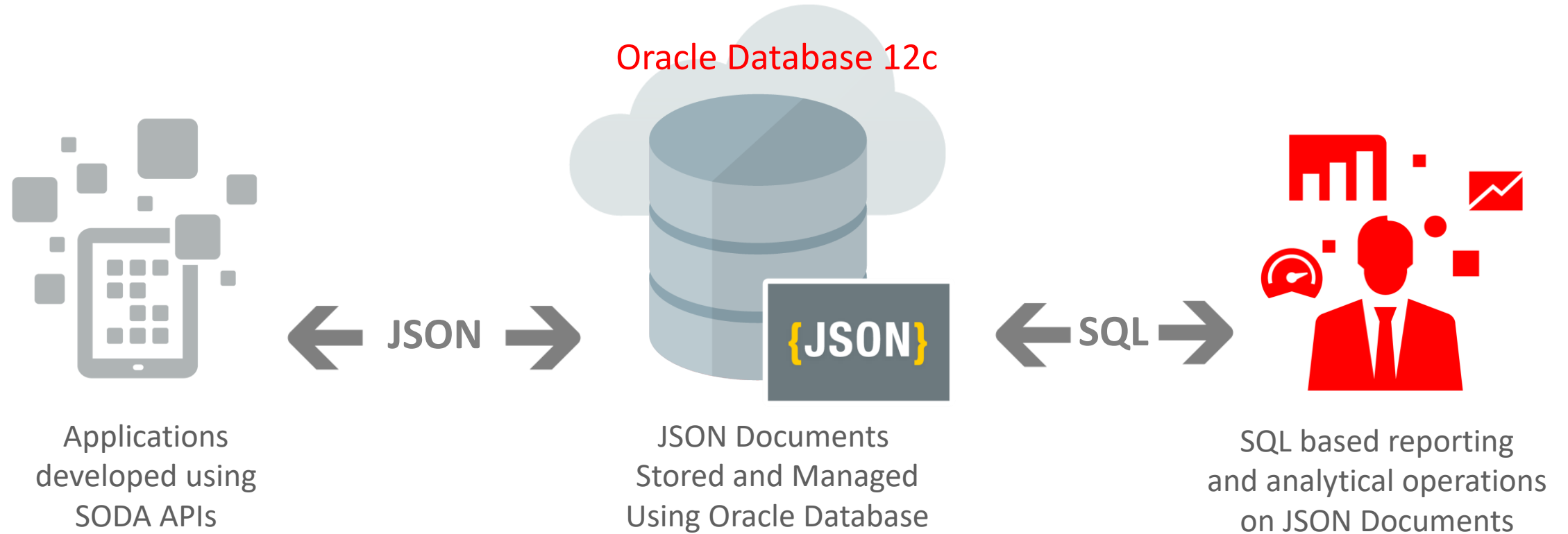
Strategy: Oracle Database as a Document Store

Built on Foundation of Oracle Database

- Transactions and consistency
- Advanced SQL engine
- Enterprise-Grade High Availability
- Enterprise-Grade Security
- Scalability and Performance: Exadata and Real Application Clusters
- Oracle Public Cloud Infrastructure

Oracle 12c JSON document store

All the power of SQL when needed



Strategy: Oracle Database as a Document Store

Full Support of Multi-Model and Hybrid Apps

- Store Relational, XML, JSON, Spatial, Graph data in same database
- Access all data via SQL
 - Trivial joins between different domains
- Hybrid relational-document schemas:
 - Relational columns and document in same table

Querying JSON using SQL

- Simple Queries

```
select j.PO_DOCUMENT  
from J_PURCHASEORDER j  
where j.PO_DOCUMENT.PONumber = 1600
```

- Advanced queries using JSON path expressions

```
select JSON_VALUE(PO_DOCUMENT, '$.LineItems[0].Part.UnitPrice' returning NUMBER(5,3))  
from J_PURCHASEORDER p  
where JSON_VALUE(PO_DOCUMENT, '$.PONumber' returning NUMBER(10)) = 1600
```

– Complies with proposed SQL2017 syntax

A woman with long brown hair and glasses, wearing a brown leather jacket and a blue patterned scarf, is sitting at a wooden table in a cafe. She is talking on a black mobile phone held to her left ear and looking down at a document or newspaper on the table. The background is a blurred cafe interior with other tables and chairs.

Storing and Querying JSON documents

Oracle Database 12c JSON capabilities

- JSON documents are stored using VARCHAR, CLOB and BLOB data types
- Query and update JSON documents using SQL and PL/SQL
- Optimize operations on JSON documents using indexing, in-memory and Exadata smart storage techniques
- Discover information about the structure and content of JSON documents
- Generate JSON documents from database content (Relational, XML, JSON)
- Integrates JSON with other type of content (Multi-Model database)

Storing JSON documents in the Oracle database

- No limitations on the kind of JSON that can be managed by the database
 - Supports the full flexibility of the JSON data model
- Documents are stored using existing datatypes
 - VARCHAR2, CLOB and BLOB
 - Choice depends on upper bounds on the size of the document
 - Choice BLOB over CLOB when dealing with large documents
 - Using standard data types ensures that existing database features work with JSON
- IS [NOT] JSON predicate tests whether or not content is valid JSON
 - Use an IS JSON check constraint to ensure contents of a column are valid JSON
 - IS JSON check constraint also activates JSON specific features

Storing JSON : DDL and DML

```
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
```

Querying and Updating JSON

- Oracle provides two mechanisms for working with JSON from SQL
 - A “Simplified Syntax” that enables simple operations directly from SQL
 - JSON operators that enable more complex operations
 - Included in the SQL 2017 standard
 - Syntax developed in conjunction with IBM
- Both techniques use JSON path expression to navigate the content of the JSON documents
 - JSON path syntax is derived from JavaScript

Simple JSON Queries

```
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 /
```


SQL/JSON operators

Operator	Description
IS [NOT] JSON	<ul style="list-style-type: none">○ test whether some data is well-formed JSON data.○ used as a check constraint.
JSON_VALUE	<ul style="list-style-type: none">○ select a scalar value from some JSON data, as a SQL value.○ used in the select list or where clause or to create a functional index
JSON_QUERY	<ul style="list-style-type: none">○ select one or more values from some JSON data as a SQL string○ used especially to retrieve fragments of a JSON document
JSON_EXISTS	<ul style="list-style-type: none">○ test for the existence of a particular value within some JSON data.
JSON_TABLE	<ul style="list-style-type: none">○ project some JSON data to a relational format as a virtual table
JSON_TEXTCONTAINS	<ul style="list-style-type: none">○ test for existence based on a text predicate

SQL/JSON operators : JSON_VALUE

- Returns exactly one scalar value from a document
 - Value identified by a JSON Path expression
 - JSON path expression must match at most one key
- Used in the select list or the where clause
- Allows you to specify the SQL type for the result
- Provides error handling options
- Use to create functional B-Tree or Bitmap indexes on a JSON document

JSON_VALUE()

```
SQL> select JSON_VALUE(PO_DOCUMENT,'$.CostCenter'), count(*)  
2   from J_PURCHASEORDER  
3   group by JSON_VALUE(PO_DOCUMENT,'$.CostCenter')  
4 /
```

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

SQL/JSON operators : JSON_QUERY

- Returns a JSON fragment (Object or Array) from JSON document
 - Fragment identified by a JSON Path expression
 - JSON data is returned in the specified SQL datatype
- Used in the select list
- Provides error handling options

JSON_QUERY()

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

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
- Used in the where clause
- JSON Path expressions used with JSON_EXISTS can contain predicates starting with Oracle Database 12c release 2
- Can be used to create functional BITMAP indexes

JSON_EXISTS()

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

JSON Path predicates with JSON_EXISTS

```
select j.PO_DOCUMENT
  from J_PURCHASEORDER j
 where JSON_EXISTS(
        PO_DOCUMENT,
        '$?(@.PONumber == $PO_NUMBER)'
        passing 1600 as "PO_NUMBER"
      )
/
```

```
select j.PO_DOCUMENT.PONumber
  from J_PURCHASEORDER j
 where JSON_EXISTS(
        PO_DOCUMENT,
        '$?(@.User == $USER && exists(
                                @.LineItems?(
                                    @.Part.UPCCode == $UPC
                                    &&
                                    @.Quantity > $QUANTITY
                                )
                            )
        )'
        passing 'AKHOO' as "USER", 43396087798 as "UPC", 8 as "QUANTITY"
      )
/
```

- Passing clause allows Bind Variables to be used to set JSON Path variables
- Exists clause used when searching for an object inside an array

SQL/JSON operators : JSON_TABLE

- Generates In-Line views of JSON content
- Used in the from clause of a SQL statement
- JSON Path expressions used to pivot values into columns
 - The ROW Pattern defines the starting point for the pivot
 - \$ represents the start of the document
 - COLUMN patterns are relative to the ROW pattern and map values to columns
- One row is output for each node identified by the Row Pattern
- Use JSON_TABLE rather than large numbers of JSON_VALUE operators

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 : Dealing with Arrays

- The NESTED PATH clause allows JSON_TABLE to process arrays
- When processing an array terminate the path expression with [*]
- Nested arrays can be processed using nested NESTED PATH clauses
- There is natural join between the rows generated by the NESTED PATH clause the parent row
- When the NESTED PATH clause is present JSON_TABLE generates one row for each member of the deepest array

Nested Path example

```
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

Piecewise updates of JSON documents

- Piecewise updates of JSON documents now supported in PL/SQL
- New PL/SQL objects enable fine grained manipulation of JSON content
 - JSON_OBJECT_T : for working with JSON objects
 - JSON_ARRAY_T : for working with JSON Arrays
 - JSON_OBJECT_T and JSON_ARRAY_T are subtypes of JSON_ELEMENT_T
- These objects provide a set of methods for manipulating JSON

PL/SQL API for JSON

- Similar to GSON
 - parse()
 - Converts a variable or column containing JSON into a object. Returns JSON_ELEMENT_T.
 - isArray(), isObject(), isString() ,etc.
 - Determine the type of the value portion of a key:value pair
 - get, put
 - Access the value portion of a key:value pair as an object or array
 - get_String, get_Number:
 - Access the value portion of a key:value pair as scalar
 - stringify, to_string
 - Converts a PL/SQL JSON data type back into textual JSON

JSON integration with PL/SQL

```

WITH FUNCTION updateTax(JSON_DOC in VARCHAR2 ) RETURN VARCHAR2 IS
  jo JSON_OBJECT_T;
  price NUMBER;
  taxRate NUMBER;
BEGIN
  jo := JSON_OBJECT_T(JSON_DOC);
  taxRate := jo.get_Number('taxRate');
  price := jo.get_Number('total');
  jo.put('totalIncludingTax', price * (1+taxRate));
  RETURN jo.to_string();
END;
ORDERS as (
  select '{"taxRate":0.175,"total":10.00}' JSON_DOCUMENT
  from dual
)
select JSON_DOCUMENT, updateTax(JSON_DOCUMENT)
from ORDERS;

JSON_DOCUMENT          UPDATETAX(JSON_DOCUMENT)
-----
{"taxRate":0.175,"total":10.00} {"taxRate":0.175,"total":10.00,"totalIncludingTax":11.75}

```

A woman with long brown hair and glasses, wearing a brown leather jacket over a blue patterned scarf, is sitting at a wooden table. She is holding a black smartphone to her ear with her left hand and looking down at a large open book or document on the table with her right hand. The background is a blurred office or study environment with other people and desks.

Accelerating JSON Query performance

Indexing, Exadata and Database In-Memory Support

JSON Search Index : A universal index for JSON content

```
create search index JSON_SEARCH_INDEX on J_PURCHASEORDER (PO_DOCUMENT) for json
```

- Supports searching on JSON using key, path and value
- Supports range searches on numeric values
- Supports full text searches:
 - Full boolean search capabilities (and, or, and not)
 - Phrase search, proximity search and "within field" searches.
 - Inexact queries: fuzzy match, soundex and name search.
 - Automatic linguistic stemming for 32 languages
 - A full, integrated ISO thesaurus framework

Query Optimizations for JSON

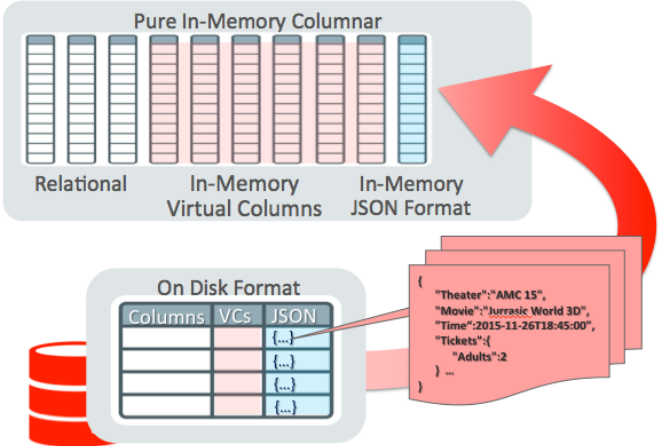
Exadata Smart Scans

- Exadata Smart Scans execute portions of SQL queries on Exadata storage cells
- JSON query operations ‘pushed down’ to Exadata storage cells
 - Massively parallel processing of JSON documents



In-Memory Columnstore

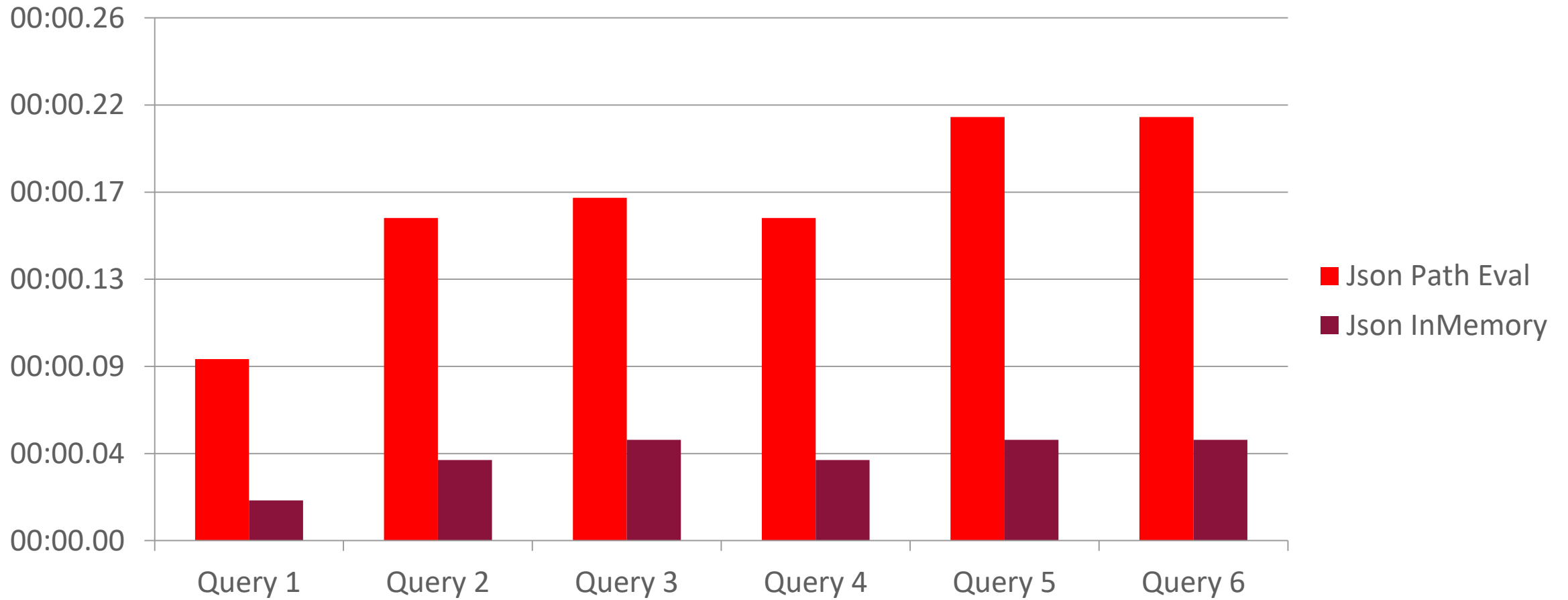
- Virtual columns, included those generated using JSON Data Guide loaded into In-Memory Virtual Columns
- JSON documents loaded using a highly optimized In-Memory binary format
- Query operations on JSON content automatically directed to In-Memory



Optimizing JSON using database In-Memory

- JSON documents stored in memory using a binary representation
- JSON path expressions evaluated without parsing
- In Memory format encodes documents using a transient dictionary to tokenize keys
- In Memory format reduces memory footprint for JSON
 - Space savings dependant on a number of factors
 - Size of Key name (ratio of Key:Value in bytes)
 - Size of Object Arrays (Number of times Key name is repeated)

JSON Database In-Memory Performance



A woman with long brown hair, wearing glasses and a brown leather jacket, is sitting at a wooden table in a cafe. She is talking on a black smartphone held to her ear with her left hand. On the table in front of her is an open book or magazine. The background is a bright, modern cafe with large windows and other people sitting at tables.

Understanding your JSON

The Oracle Data Guide for JSON

Data Guide : Understanding your JSON documents



- Metadata discovery: discovers the structure of collection of JSON documents
 - Optional: deep analysis of JSON for List of Values, ranges, sizing etc.
- Automatically Generates
 - Virtual columns
 - Relational views
 - De-normalized relational views for arrays
 - Reports/Synopsis of JSON structure

JSON Data Guide : Generating the Data Guide

1. Create Table
2. Create Search Index
3. Insert Document
4. Generate Data Guide
5. Data Guide
6. Create View
7. Describe View

```
SQL> create table MOVIE_TICKETS (
```

```
SQL> create search index MOVIE_TICKETS_DGUIDE
```

```
SQL> insert into MOVIE_TICKETS
```

```
SQL> select dbms_json.getDataGuide('MOVIE_TICKETS')
```

```
{"type": "object",
```

```
SQL> call dbms_json.createviewonpath(
```

```
SQL> desc MOVIE_TICKETS_VIEW
```

Name	Type
BOOKING_ID	RAW(16)
BOOKING_TIME	TIMESTAMP(6) WITH TIME ZONE
BOOKING_DETAILS\$Movie	VARCHAR2(16)
BOOKING_DETAILS\$Theater	VARCHAR2(16)
BOOKING_DETAILS\$Adults	NUMBER
BOOKING_DETAILS\$StartTime	VARCHAR2(32)

```
SQL>
```

JSON Dataguide Example : Virtual Columns

1. Create Table
2. Create Data Guide
3. Insert Document
4. Describe Table
5. Insert Document
6. Describe Table

```

SQL> create table MOVIE_TICKETS (
2
3
4
5
6
7
8 /
9
10
11
12
13
SQL> create search index MOVIE_TICKETS_DGUIDE
2
3
4
5
6
7
8 /
9
10
11
12
13
SQL> insert into MOVIE_TICKETS
2
3
4
5
6
7
8 /
9
10
11
12
13
SQL> desc MOVIE_TICKETS
2
3
4
5
6
7
8 /
9
10
11
12
13
SQL> insert into MOVIE_TICKETS
2
3
4
5
6
7
8 /
9
10
11
12
13
SQL> desc MOVIE_TICKETS
2
3
4
5
6
7
8 /
9
10
11
12
13
SQL>

```

Name	Type
BOOKING_ID	RAW(16)
BOOKING_TIME	TIMESTAMP(6) WITH TIME ZONE
BOOKING_DETAILS	VARCHAR2(4000)
BOOKING_DETAILS\$Movie	VARCHAR2(16)
BOOKING_DETAILS\$Theater	VARCHAR2(16)
BOOKING_DETAILS\$Adults	NUMBER
BOOKING_DETAILS\$StartTime	VARCHAR2(32)
BOOKING_DETAILS\$Child	NUMBER
BOOKING_DETAILS\$Senior	NUMBER



Accessing relational data as JSON

SQL/JSON Publishing operators

JSON Generation

- Operators defined by SQL Standards body
 - JSON_ARRAY, JSON_OBJECT, JSON_ARRAYAGG and JSON_OBJECTAGG
 - Nesting of operators enables generation of complex JSON documents
- Simplifies generating JSON documents from SQL Queries
 - Eliminate syntactic errors associated with string concatenation
- Improves performance
 - Eliminate multiple round trips between client and server

JSON_ARRAY: Representing rows as arrays

```
SQL> select JSON_ARRAY(EMPLOYEE_ID, FIRST_NAME, LAST_NAME) JSON  
2      from HR.EMPLOYEES  
3      where EMPLOYEE_ID = 100;
```

JSON

[100 , "Steven" , "King"]

SQL>

- JSON Array contains one item for each argument
- Arrays can contain heterogeneous items

JSON_OBJECT : Representing rows as objects

```
SQL> select JSON_OBJECT('Id' is EMPLOYEE_ID, 'FirstName' is FIRST_NAME,  
2                      'LastName' is LAST_NAME) JSON  
3      from HR.EMPLOYEES  
4      where EMPLOYEE_ID = 100;
```

JSON

```
{ "Id" : 100 , "FirstName" : "Steven" , "LastName" : "King" }
```

SQL>

- JSON object contains a key:value pair for each set of arguments

JSON_ARRAYAGG: Embedding arrays in documents

```
select JSON_OBJECT(  
  'departmentId' is d.DEPARTMENT_ID,  
  'name' is d. DEPARTMENT_NAME,  
  'employees' is (  
    select JSON_ARRAYAGG(  
      JSON_OBJECT(  
        'employeeId' is EMPLOYEE_ID,  
        'firstName' is FIRST_NAME,  
        'lastName' is LAST_NAME,  
        'emailAddress' is EMAIL  
      )  
    )  
    from HR.EMPLOYEES e  
    where e.DEPARTMENT_ID = d.DEPARTMENT_ID  
  )  
  ) DEPT_WITH_EMPLOYEES  
from HR.DEPARTMENTS d  
where DEPARTMENT_NAME = 'Executive';
```

- Creates a JSON Array from the results of a nested sub-query

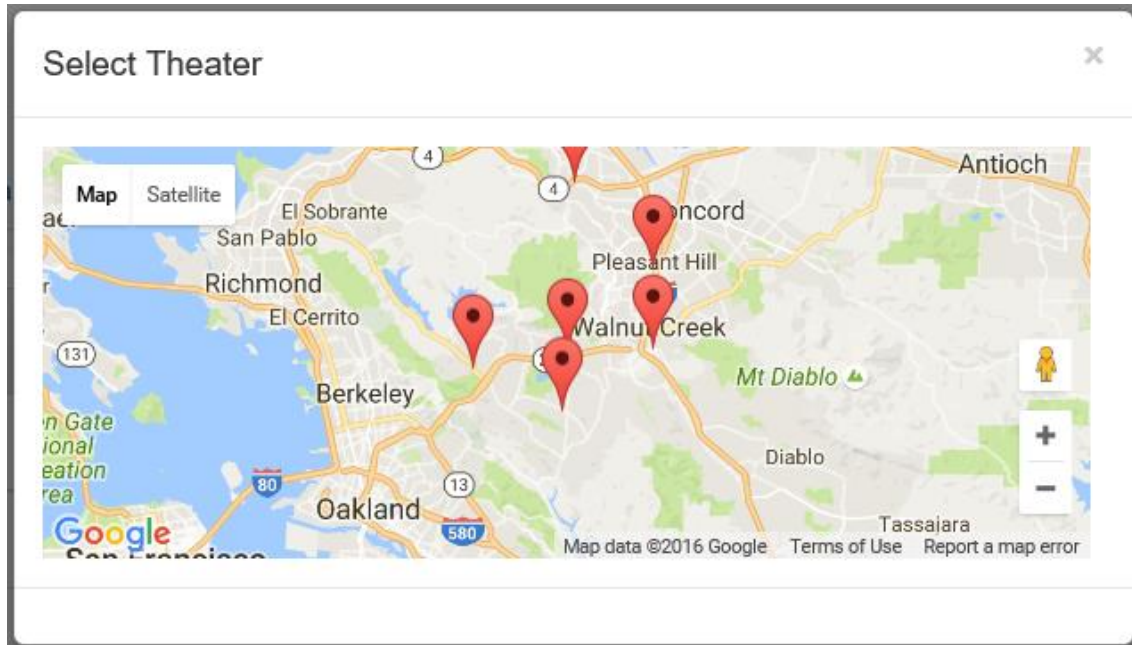
```
DEPT_WITH_EMPLOYEES  
-----  
{  
  "departmentId": 90,  
  "name": "Executive",  
  "employees": [  
    {  
      "employeeId": 100,  
      "firstName": "Steven",  
      "lastName": "King",  
      "emailAddress": "SKING"  
    }, {  
      "employeeId": 101,  
      "firstName": "Neena",  
      "lastName": "Kochhar",  
      "emailAddress": "NKOCHHAR"  
    }, {  
      "employeeId": 102,  
      "firstName": "Lex",  
      "lastName": "De Haan",  
      "emailAddress": "LDEHAAN"  
    }  
  ]  
}
```



Using Oracle Spatial with Geo-JSON content

Multi-Model Database

GeoJSON support : Location Indexing & Searching

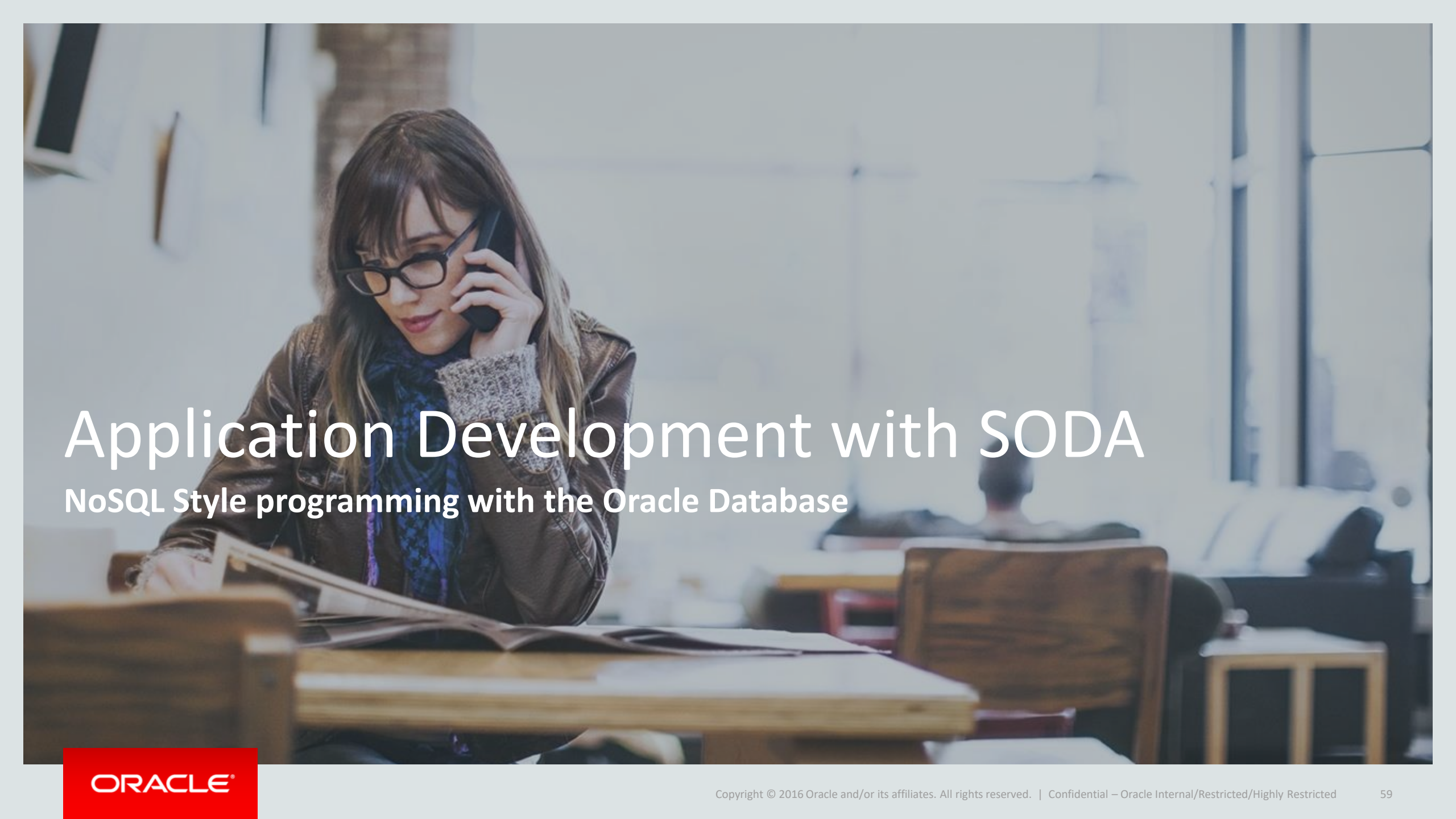


- SQL

```
select t.JSON_DOCUMENT.name
  from THEATER t
 where SDO_WITHIN_DISTANCE (
        JSON_VALUE(t.JSON_DOCUMENT, '$.location.geoCoding'
                   returning SDO_GEOMETRY NULL ON ERROR),
        sdo_geometry(2001, 8307,
                     sdo_point_type(37.8922312,-122.1306916, null),
                     null,null),
        'distance=5, units="mile"'
      ) = 'TRUE `;
```

- SODA Query-by-Example

```
{
  "location.geoCoding": {
    "$near": {
      "$geometry": {
        "type": "Point",
        "coordinates": [37.8922312,-122.1306916]
      },
      "$distance": 5,
      "$unit": "mile"
    }
  }
}
```

A woman with long brown hair and glasses, wearing a brown leather jacket over a blue patterned scarf, is sitting at a wooden table in a cafe. She is holding a black smartphone to her ear with her left hand and looking down at an open book or magazine on the table with her right hand. The background is a blurred cafe interior with other tables and chairs.

Application Development with SODA

NoSQL Style programming with the Oracle Database

SODA for REST

- Implementation of SODA for REST developers
- Collection of Micro-Services for working with JSON documents stored in Oracle Database 12c
- URI patterns mapped to operations on document collections
- Can be invoked from almost any programming language
- Distributed as part of Oracle REST Data Services (ORDS 3.0)
 - Can be installed as a JAVA servlet under the XMLDB HTTP Listener

Sample services provide by SODA for REST

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

SODA: Sample Query-By-Example documents

- Order By

```
{"$query": {}, "$orderby": {"releaseDate": -1}}
```

- Exact Match

```
{"location.city": "SAN FRANCISCO"}
```

- List of Values

```
{"id": {"$in": [245168, 299687, 177572, 76757]}}
```

- Full Text Searching

```
{"plot": {"$contains": "$ (colour) "}}
```

- Multiple Predicates with Ordering

```
{"movieId": 109410,  
  "startTime": {  
    "$gte": "2016-09-12T07:00:00.000Z",  
    "$lt": "2016-09-13T07:00:00.000Z"  
  },  
  "$orderby": {"screenId": 1, "startTime": 2}  
}
```

- Distance Search

```
{"location.geoCoding": {  
  "$near": {  
    "$geometry": {  
      "type": "Point",  
      "coordinates": [37.8953, -122.1247]  
    },  
    "$distance": 5,  
    "$unit": "mile"  
  }  
}}
```

SODA for JAVA

- Implementation of SODA for Java programmers
- SODA for Java provides classes for
 - Collection Management
 - CRUD operations on JSON documents
 - Query-by-Example for document searching
 - Utility and control functions
- 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();
```

Why choose Oracle Database 12c and SODA

- Oracle Database 12c can satisfy the data management requirements for modern application development stacks
- Using Oracle and SODA is as simple as using any other No-SQL based document store technology
- SODA allows applications to be developed and deployed without any knowledge of SQL and without DBA support.
- Applications can take full advantage of the capabilities of Oracle Database
- Using Oracle Database protects existing investment in data management software and skills

A woman with long brown hair and glasses is sitting at a wooden table in a bright, modern office or cafe. She is wearing a brown leather jacket over a blue patterned scarf. She is holding a black mobile phone to her ear with her left hand and looking down at a newspaper or magazine on the table with her right hand. The background is slightly blurred, showing other people and large windows.

Summary

JSON Support in Oracle Database

Fast Application Development + Powerful SQL Access

Application developers:

Access JSON documents using RESTful API

```
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" } ]
}
```

Oracle Database 12c



SQL Developers and Analytical tools:
Query JSON using SQL

```
select
  c.json_document.firstName,
  c.json_document.lastName,
  c.json_document.address.city
from customers c;
```

firstName	lastName	address.city
-----	-----	-----
"John"	"Smith"	"New York"

Where do Customers go to learn more?

The screenshot shows the Oracle Technology Network (OTN) website. The browser address bar displays www.oracle.com/technetwork/database/application-development/oracle-document-store/index.html. The page features the Oracle logo and navigation menus for Account, Sign Out, Help, Country, Communities, I am a..., I want to..., Search, Products, Solutions, Downloads, Store, Support, Training, Partners, and About. A breadcrumb trail indicates the current location: Oracle Technology Network > Database > Application Development > Oracle as a Document Store.

The main content area is titled "Oracle as a Document Store" and includes a large "12c" graphic and a "JSON" icon. The text describes the support for schemaless application development using the JSON data model, highlighting a hybrid development approach that combines schema flexibility and speedy application development of NoSQL document stores with enterprise-ready features.

On the left, a sidebar menu lists various database topics: Database 12c, Database In-Memory, Multitenant, Options, Application Development, Big Data Appliance, Data Warehousing & Big Data, Database Appliance, Database Cloud, Exadata Database Machine, High Availability, Manageability, Migrations, Security, Unstructured Data, Upgrades, Windows, and Database Technology Index.

On the right, there are three promotional boxes: "ORACLE OPEN WORLD" (September 18-22, 2016, San Francisco) with a "See Us Here" link and hashtag #oow16; "Oracle Database Cloud" with a "Get Started" link; and "Get the Latest Oracle Database 12c Tutorials" with a "Plug into the Cloud" link and "Access Now" button.

At the bottom, two download links are provided: "White Paper: Schemaless Application Development with Oracle Database 12c" and "Video: JSON in Oracle Database 12c".

<http://www.oracle.com/technetwork/database/application-development/oracle-document-store/index.html>

Learn More about Oracle, JSON and SODA

- Oracle JSON document store on the Oracle Technology Network
 - <http://otn.oracle.com/database/application-development/oracle-document-store/index.html>
- Downloadable Oracle XML and JSON Code samples on Github
 - <https://github.com/oracle/xml-sample-demo>
 - <https://github.com/oracle/json-in-db>

ORACLE®