
ORACLE®

SODA 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.

Program Agenda

1. Introduction to JSON
2. Oracle Database 12c as a Document Store
3. Modern Application Development Architecture
4. SODA for REST
5. SODA for Java
6. Application Development in the Cloud
7. JSON Support in Oracle Database 12c
8. Summary

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

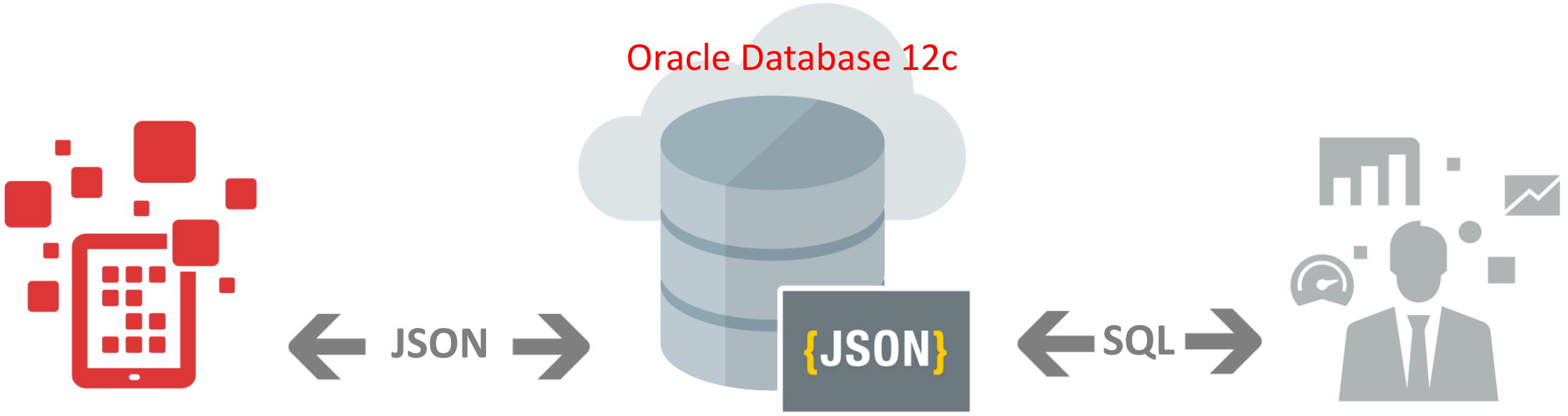
Oracle Database 12c as a Document Store

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



Applications developed using SODA APIs

JSON Documents Stored and Managed Using Oracle Database

SQL based reporting and analytical operations on JSON Documents

Strategy: Oracle Database as a Document Store

Core Capabilities for Document Workloads

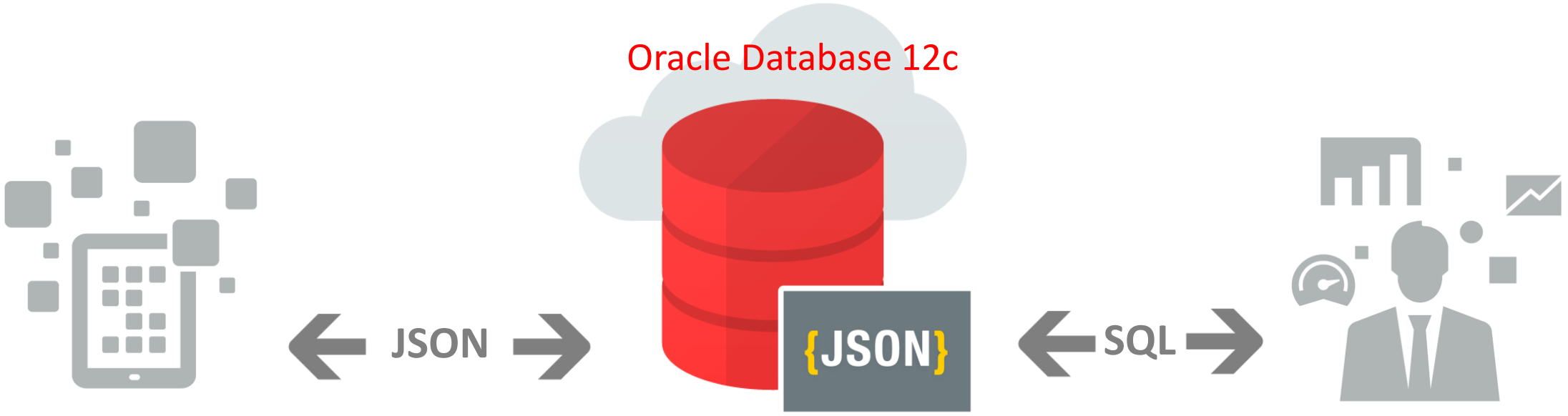
- Store and manage JSON and XML documents in Oracle Database
- Accessible via REST and all major programming languages
- Full query capabilities using JSON Path, XQuery and SQL
- Comprehensive, path-aware indexing
- No need to learn SQL or require DBA when developing applications
- Fits into the DevOPS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



Applications developed using SODA APIs

JSON Documents Stored and Managed Using Oracle Database

SQL based reporting and analytical operations on JSON Documents

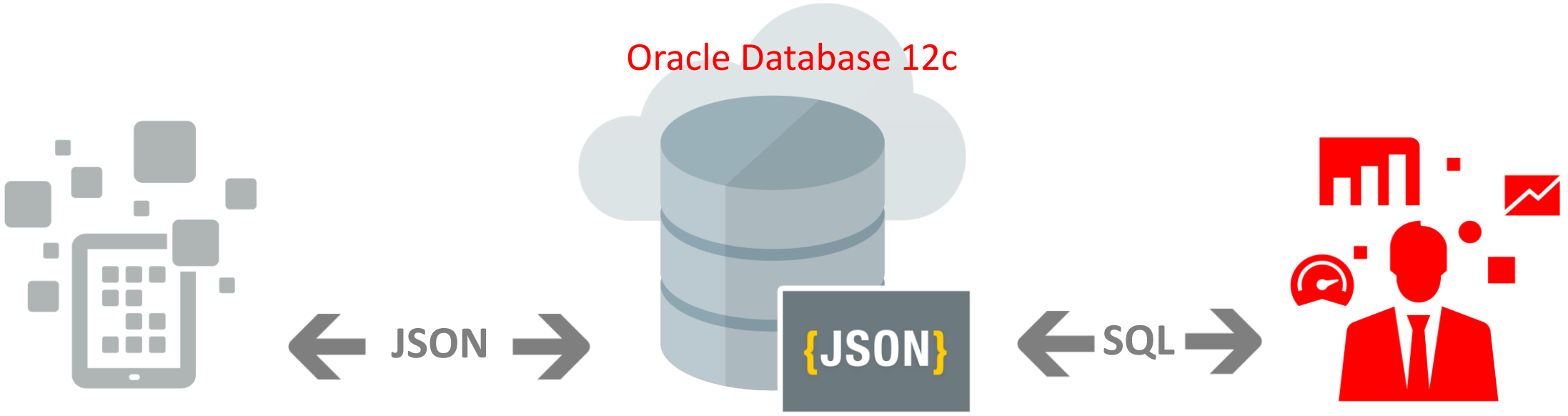
Strategy: Oracle Database as a Document Store

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



Applications developed using SODA APIs

JSON Documents Stored and Managed Using Oracle Database

SQL based reporting and analytical operations on JSON Documents

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

Modern Application Development Architecture

Application Development Architecture

Client Tier

- Browser
  - HTML 5, CSS3
  - JavaScript 6.0
- JavaScript Frameworks
  - JQuery, Angular.js,
  - Twitter Bootstrap, Oracle JET
- JSON Data Model with Rest Services

Application Development Architecture

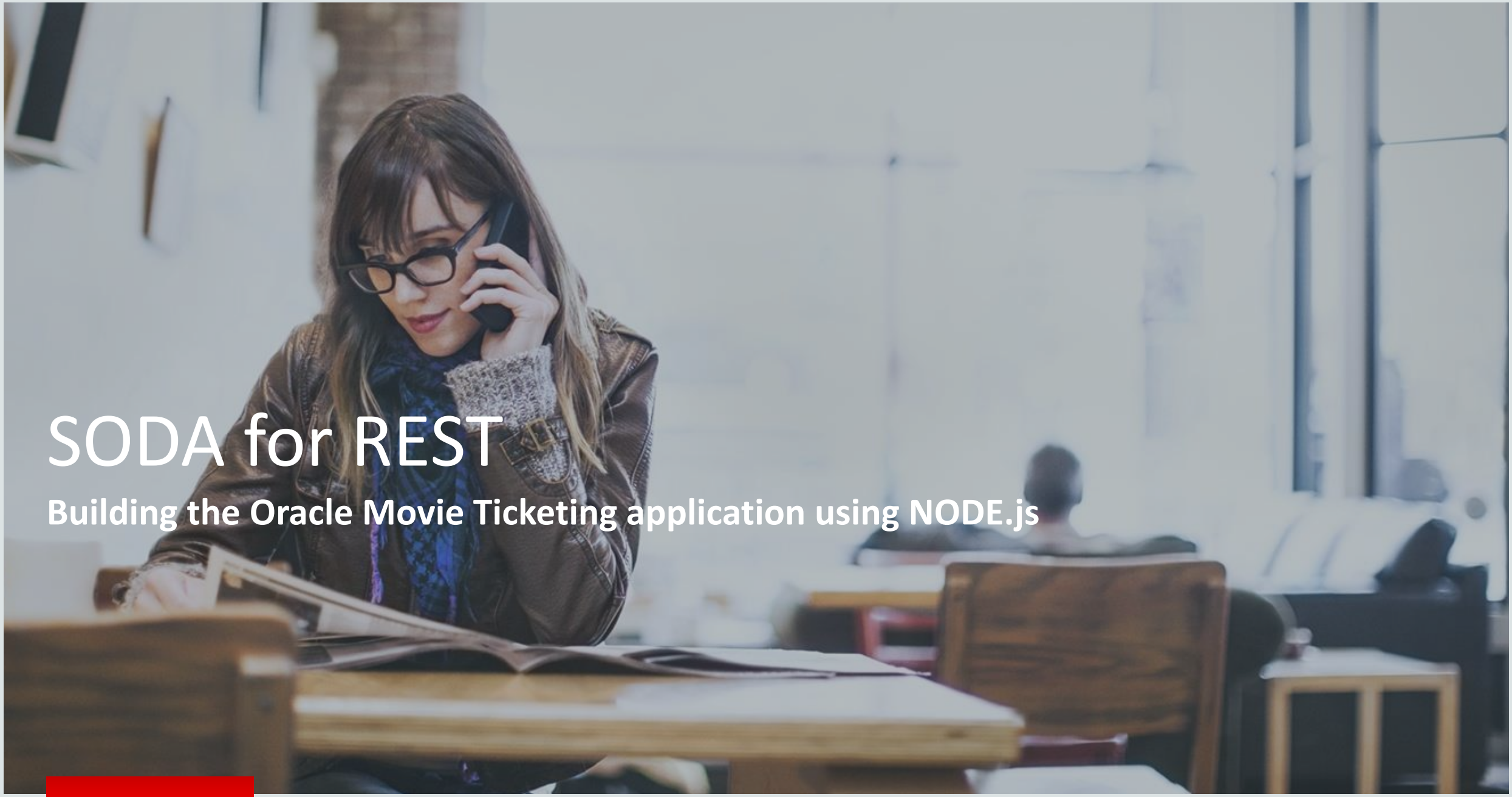
MidTier

- Node.js
  - JavaScript 6.0
  - Node Plugins: Express.js, Request.js, Morgan, etc.
- JSON Data Model with Rest Services

Application Development Architecture

Data Layer

- JSON based data persistence
  - Flexible



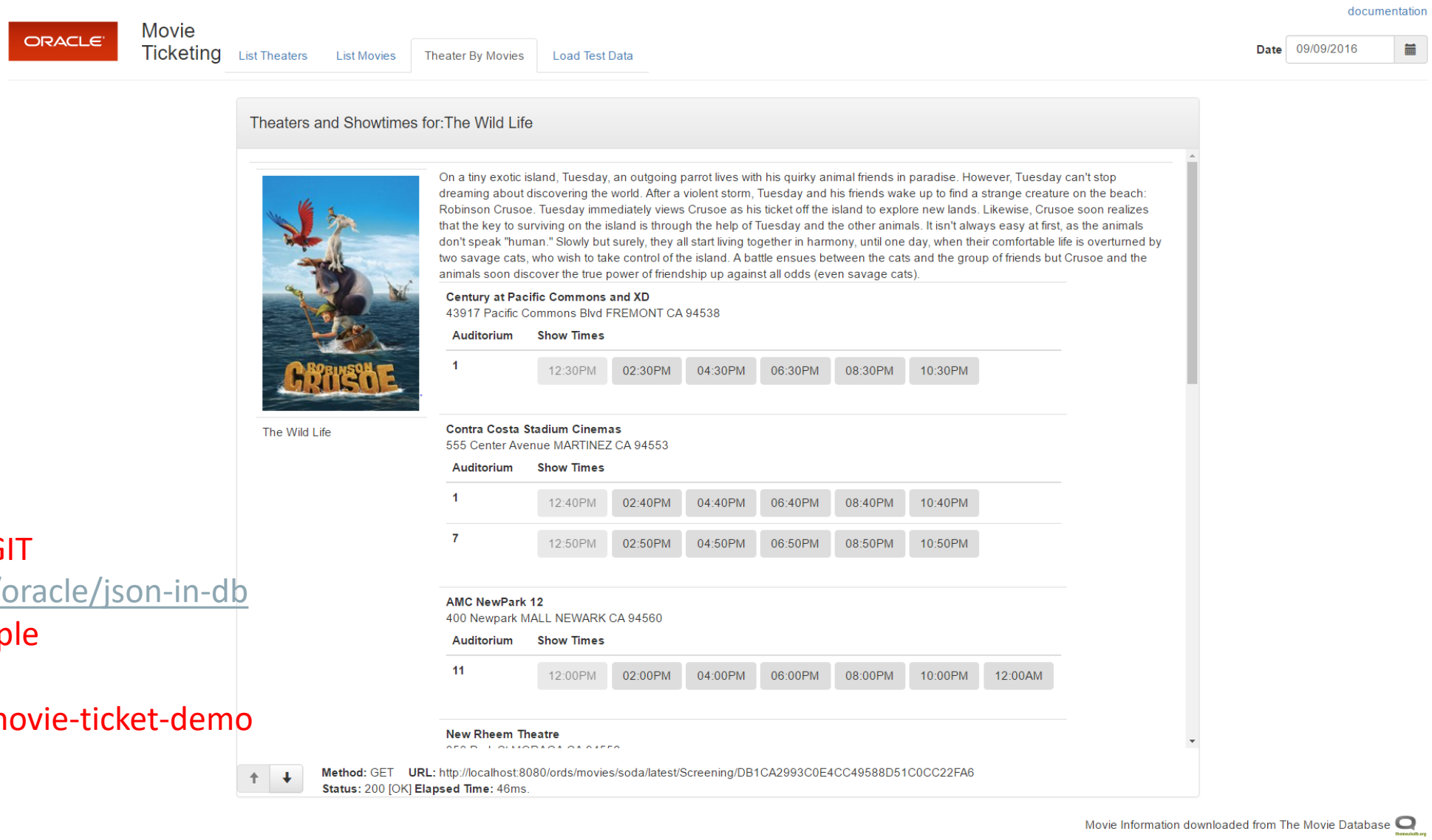
SODA for REST

Building the Oracle Movie Ticketing application using NODE.js

ORACLE

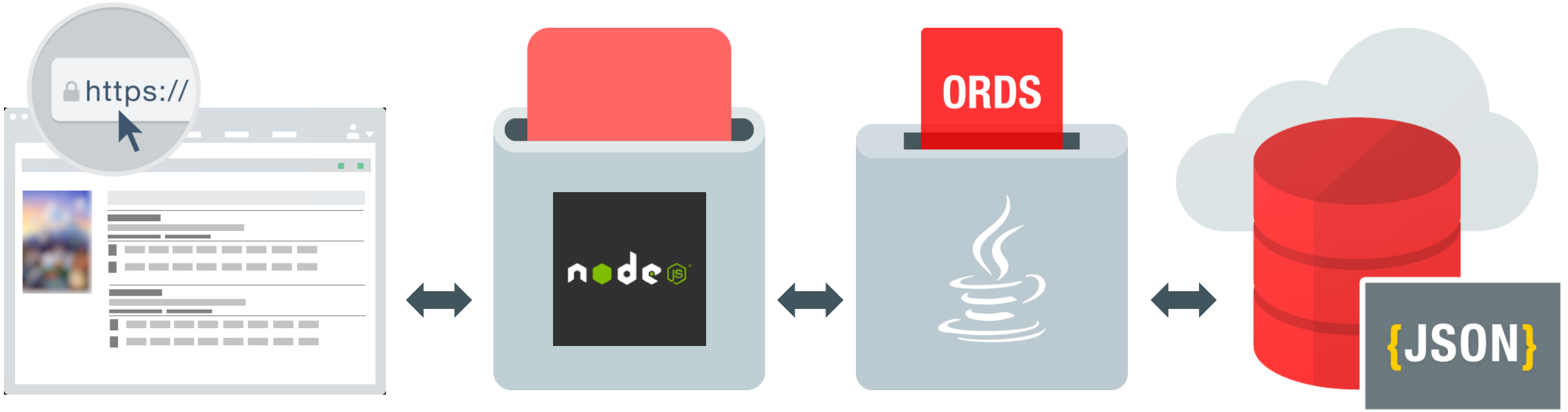
Demo

DOWNLOAD links: GIT
https://github.com/oracle/json-in-db
Folder : Node Example
or
npm install oracle-movie-ticket-demo



Oracle Movie Ticketing Application

Architecture



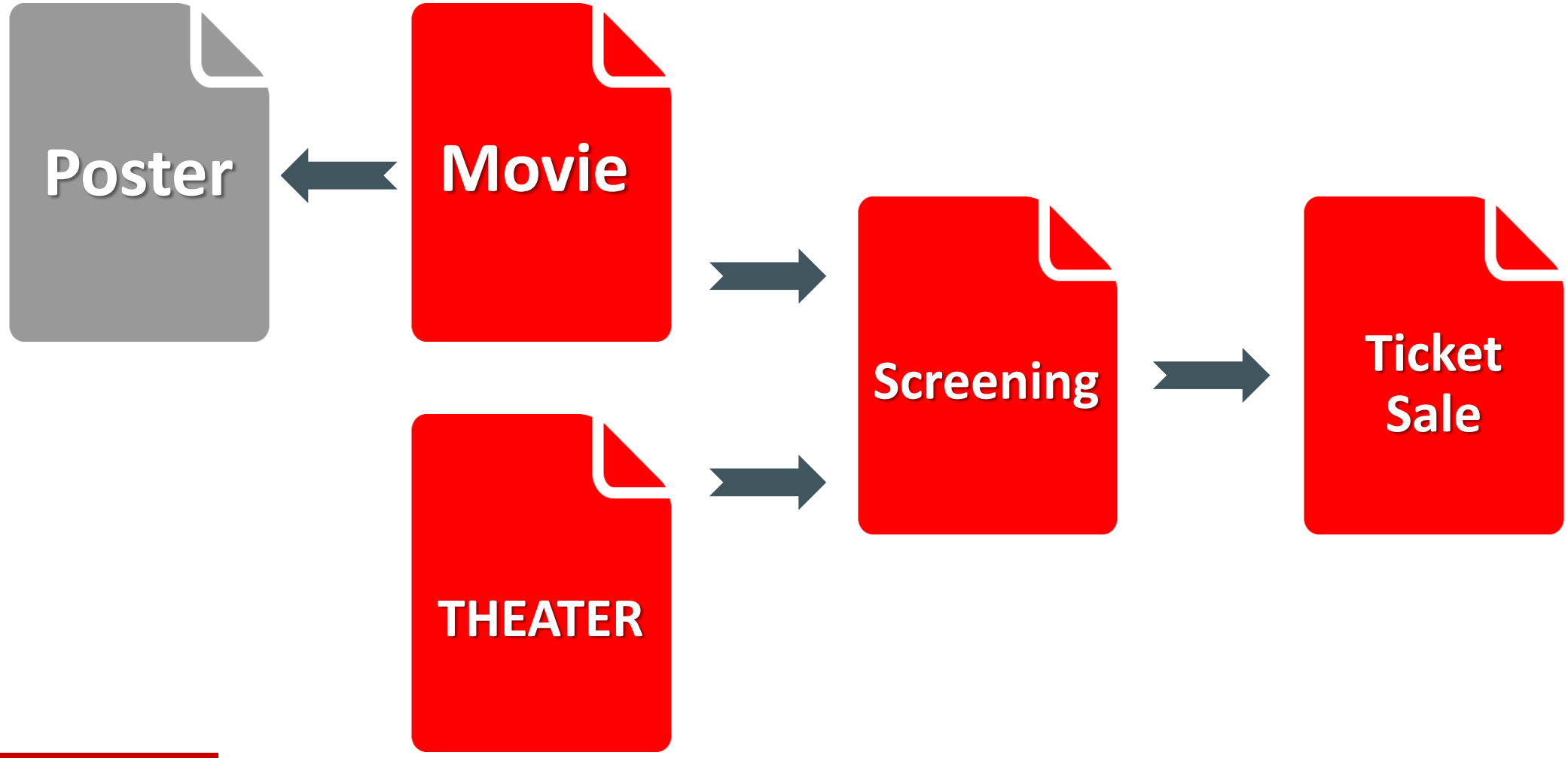
HTML5, JavaScript Bootstrap and Angular.js

JavaScript

SODA for REST

Oracle JSON Document Store

Movie Ticketing Data Model : Document Collections



Movie Ticket Data Model : Theater Document

```
{
    "id": 12,
    "name": "Orinda Theatre",
    "location": {
        "street": "4 Orinda Way",
        "city": "ORINDA",
        "zipCode": "94563",
        "state": "CA",
        "phoneNumber": null,
        "geoCoding": {
            "type": "Point",
            "coordinates": [ -122.19335, 37.886116 ]
        }
     },
     "screens": [...]
}
```

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

SODA for REST

- 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

Node.js: Invoking SODA for REST with Request.js

- Request.js makes it easy to invoke REST services using Node
- The 'options' object defines the request
  - Method : HTTP Method such as GET, POST, PUT, DELETE, etc.
  - URI: The endpoint to be accessed
  - qs: Any Query String arguments
  - json: JSON payload for PUT & POST operations.
- The Callback function processes the result of the HTTP Request
  - Provides access to errors, the HTTP response object and any body returned by the HTTP operation
- Request.js simplifies authentication, redirects and proxies

Node.js : Using Request.js to invoke SODA for REST

```
function generateRequest(requestOptions) {
    return new Promise(function(resolve, reject) {
        request(requestOptions, function(error, response, body) {
            if (error) {
                reject(getSodaError(requestOptions,err));
            }
            else {
                processSodaResponse(response, body, resolve, reject);
            };
        }).auth(getUsername(), getUsername(), true);
    });
}
```

Node.js : Creating a Collection using SODA for REST

```
function createCollection(collectionName) {

    var requestOptions = {
      method    : 'PUT'
    , uri       : getDocumentStoreURI() + collectionName
    , json      : getCollectionProperties(collectionName)
    };

   return generateRequest(requestOptions);
}
```

- PUT on the Collection URI
- Use optional CollectionProperties object to define the collection.

Node.js : Inserting JSON using SODA for REST

```
function createCollection(collectionName, document) {

    var requestOptions = {
      method    : 'POST'
    , uri       : getDocumentStoreURI() + collectionName
    , headers   : setContentType('application/json')
    , json      : document
    };

   return generateRequest(requestOptions);
}
```

- POST the content to the Collection URI
- Supply the content using the key "json"

SODA for REST: Response for Insert Document

```
{ "items": [ {
      "id": "20F7D7197FB5476F9C9AFD9B2D37FA20",
      "etag": "9995B1786B73B2BF6E6574D5B0506E405D37A6A1A631FE2A0C511B0E57480C14",
      "lastModified": "2016-09-17T20:35:01.045977Z",
      "created": "2016-09-17T20:35:01.045977Z"
  } ],
  "hasMore": false,
  "count": 1
}
```

- Successful insert returns document metadata
  - "id" : unique id that can be used to fetch document
  - "etag" : value that can be used to ensure conflicting updates do not occur

Node.js : Listing a Collection using SODA for REST

```
function getCollection(collectionName, limit, fields) {

    var requestOptions = {
      method    : 'GET'
    , uri       : getDocumentStoreURI() + collectionName
    , qs        : {limit: limit, fields: fields}
    , json      : true
    };

   return generateRequest(requestOptions);
}
```

- GET on the Collection URI returns array of matching documents
- Limit and Fields arguments control what is returned
- Setting json to true forces the response to be returned as a JSON object

SODA for REST: Response for List and QBE operations

```
{ "items": [ {
        "id": "3933BEA4CC374721BB950A916ACE8A30",
        "etag": "255D0AEFBA5B37D1C49E226EEC1DDB6D72BD7C1299C6C3A0ED5808E27011CC3F",
        "lastModified": "2016-09-17T20:35:17.054822Z",
        "created": "2016-09-17T20:35:17.054822Z",
        "value": {
          "theaterId": 34,
          "movieId": 228165,
          ...
        }
      }, ...
  ],
  "hasMore": false,
  "count": 22
}
```

- List and QBE operations return an array of objects
- Each object describes one document that matches the supplied criteria
- Object contains metadata, content or both

Node.js : Query-By-Example using SODA for REST

```
function getCollection(collectionName, limit, fields) {

    var requestOptions = {
      method    : 'POST'
    , uri       : getDocumentStoreURI() + collectionName
    , qs        : {limit: limit, fields: fields, action: 'query'}
    , json      : qbe
    };

   return generateRequest(requestOptions);
}
```

- POST on the Collection URI with parameter action=query
- Limit and Fields arguments control what is returned
- Pass Query By Example definition passed using json

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"
   }}}
```

Node.js : List Theaters Service

```
function theatersService(response, next) {
   sodaRest.getCollection('Theater').then(function (sodaResponse) {
     response.json(sodaResponse);
     response.end();
   }).catch(function(e){
     next(e);
   });
}
```

List Theaters : Angular Controller

```
var app = angular.module('movieTicketing', ['ngCookies']);

app.controller('theatersCtrl',
    function($scope, $http, theaterService) {

        $scope.theaterService = theaterService;

        $http({
            method: 'GET',
            url: '/movieticket/theaters/',
        }).success(function(data, status, headers) {
            $scope.theaterService.theaters = data;
        });
    }
);
```

List Theaters : Angular.js

```
<div class="tab-pane active" id="tab_TheaterList">
  <div id="TheaterList" class="panel panel-default" ng-controller="theatersCtrl">
    <div class="panel-body" style="height:65vh; overflow: auto;">
      <table class="table table-fixed">
        <tbody>
          <tr ng-repeat="theater in theaterService.theaters">
            <td>{{theater.value.name}}</td>
            <td>{{theater.value.location.street}}  {{theater.value.location.city}}
              {{theater.value.location.state}} {{theater.value.location.zipCode}}</td>
            <td>
            <button id="btn_MoviesByTheater" type="button"
              class="btn btn-default btn-success"
              ng-click="theaterService.getMoviesByTheater(theater.id)">Movies</button>
          </td>
        </tr>
      </tbody>
    </table>
    …
```

Node.js : Ticket Sale Service

```
function bookTickets(sessionState, bookingRequest) {
	var key = bookingRequest.key;
	var eTag = null;
	var screening = {};
	var seatsRequired = bookingRequest.adult + bookingRequest.senior + bookingRequest.child;
	return movieAPI.getScreening(sessionState, key).then(function(sodaResponse) {
		eTag = sodaResponse.eTag;
		screening = sodaResponse.json;
		if (screening.seatsRemaining < seatsRequired) {
			return {status : 'SoldOut', message : ' Only + screening.seatsRemaining + ' seats remain'};
		} else {
			screening.seatsRemaining = screening.seatsRemaining - seatsRequired;
			return movieAPI.updateScreening(sessionState, key, screening, Tag).then( function(sodaResponse) {
				switch (sodaResponse.statusCode) {
					case 200: // Screening Updated : Record Ticket Sale
						var ticketSale = makeTicketSale(bookingRequest, screening);
						return movieAPI.insertTicketSale(sessionState, ticketSale).then(function(sodaResponse) {
							switch (sodaResponse.statusCode) {
								case 201: return { status : 'Booked', message : 'Please enjoy your movie.' } // Ticket Sale complete
								default: throw sodaResponse;
							}
						}).catch(function (err) {
							throw err;
						})
					default: throw sodaResponse;
				}
			}).catch(function (err) {
				switch (err.statusCode) {
					case 412: return bookTickets(sessionState, bookingRequest) // Conflicting Update - Try Again
					default: throw err;
				}
			})
		}
	}) // Default Error Handler here for getScreening()
}
```

SODA for Java

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 a 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

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();
```

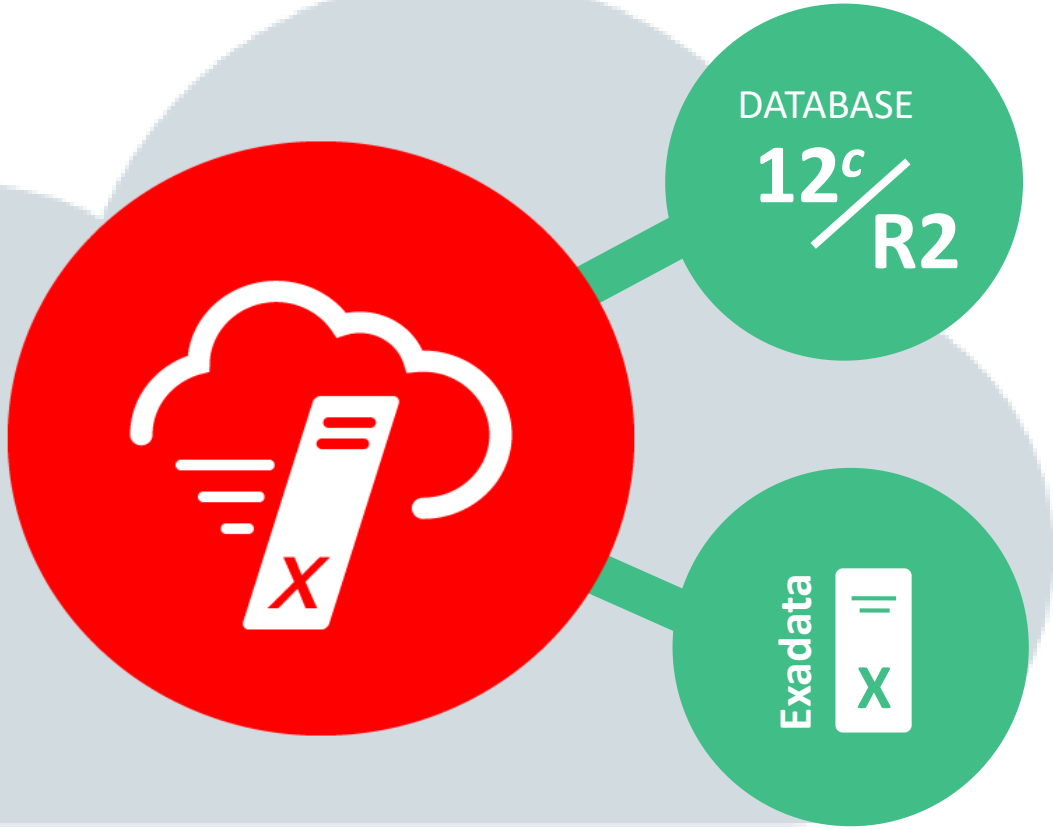
Application Development in the Cloud

Oracle Exadata Express

Oracle Database Exadata Express Cloud Service

A Fully Managed Experience for Small-to-Medium Sized Cloud Databases

- Launch platform for Oracle's newest release – Oracle Database 12*c* Release 2
- Runs on the world's #1 engineered system – Oracle Exadata
- Packed with features for modern application development
- Perfect for line-of-business apps, dev & test environments and more
- Simple all-inclusive prices starting at only $175 / month



Application Development on Exadata Express

- All popular languages supported
- Full database support for
  - JSON
  - REST
- Development tools included
  - Application Express
  - SQL Developer
  - SODA

and more …

JSON Support in Oracle Database 12c

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)

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

Filtering based on JSON Path Expressions

NEW IN 12.2

```
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

JSON Search Index : A universal index for JSON content

NEW IN 12.2

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

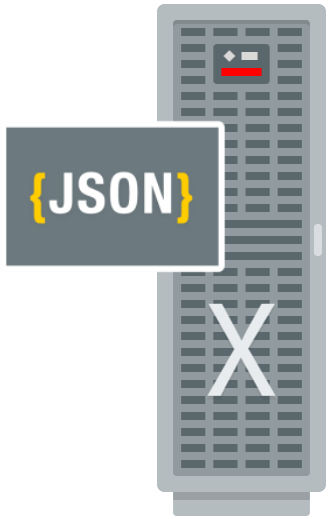
- Supports searching on JSON using key, path and value
- Supports range searches on numeric values
- Supports full text searches:
  - Full boolean search capabilities (and, or, and not)
  - Phrase search, proximity search and "within field" searches.
  - Inexact queries: fuzzy match, soundex and name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

Query Optimizations for JSON

NEW IN 12.2

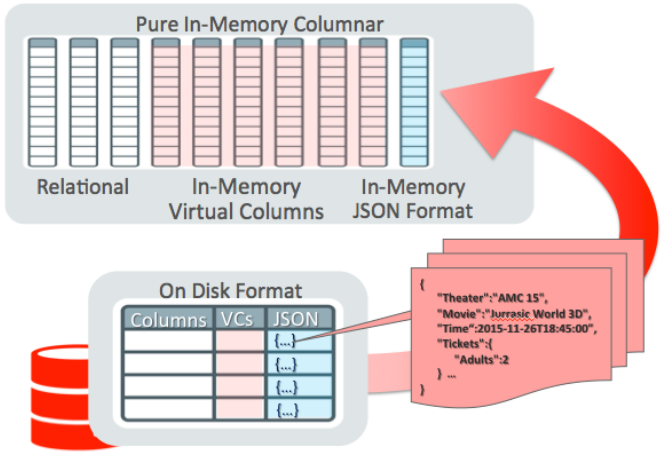
Exadata Smart Scans

- 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



ORACLE

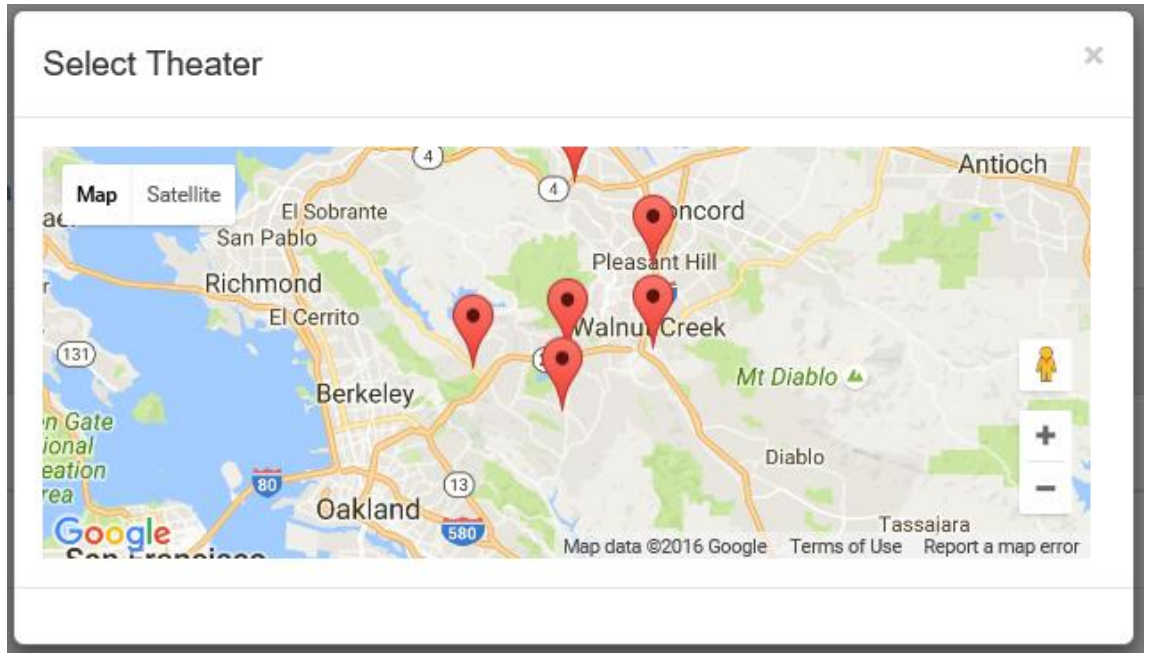
JSON Generation

NEW IN 12.2

- 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

ORACLE

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"
    }
  }
}
```

Summary

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 a 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

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" }  ]
}
```

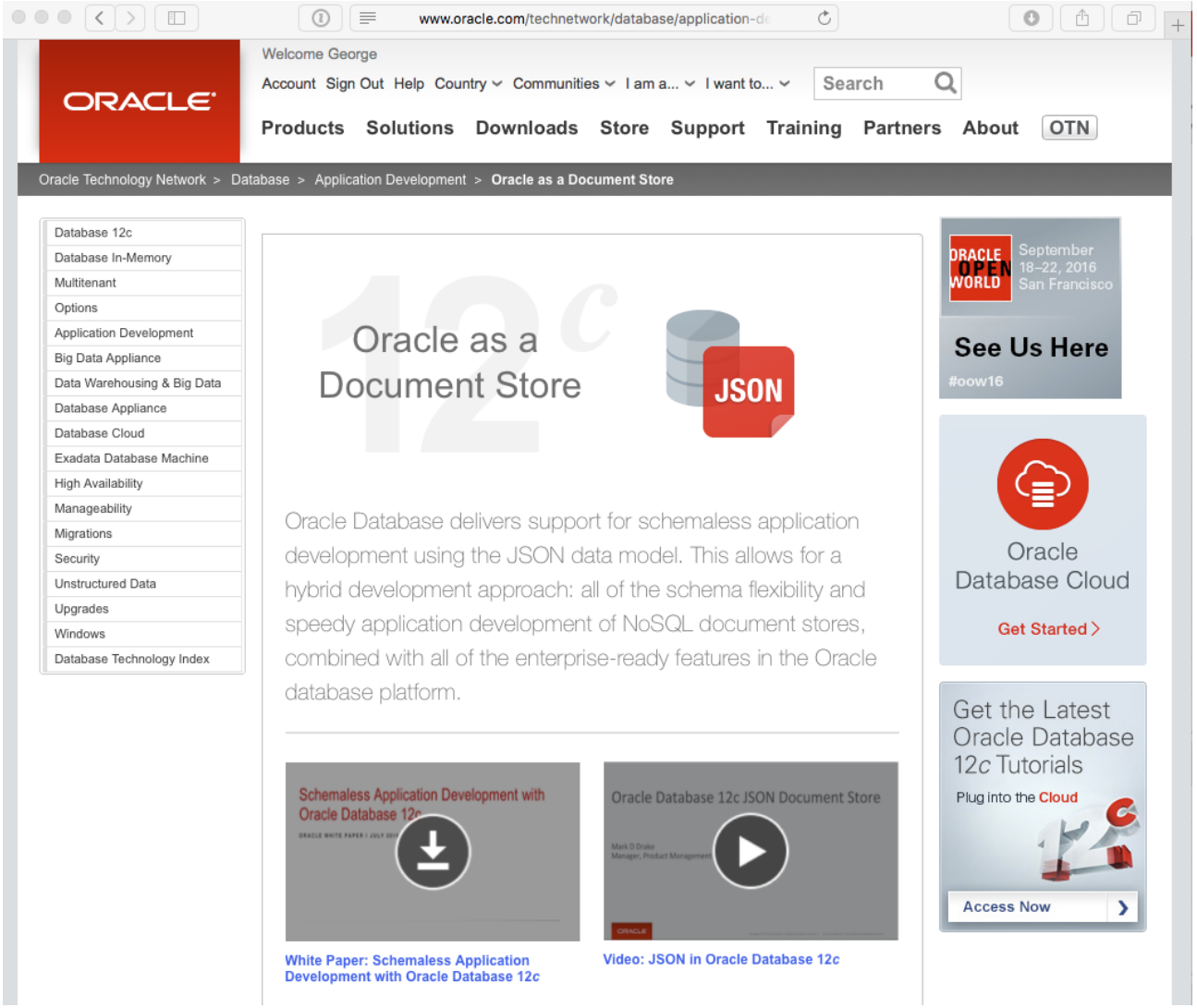
Oracle Database 12c

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?



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

Hardware and Software
Engineered to Work Together


ORACLE®