ORACLE

JSON-based Application Development with Oracle Database (and MongoDB compatibility)

Releases 19c and 21c, On-Premise and Cloud,
Autonomous JSON Database and
Oracle Database API for MongoDB

February 2022, Version 1.1
Copyright © 2022, Oracle and/or its affiliates
Public

1 **Business / Technical Brief** / JSON-based
Application Development
with Oracle Database
(and MongoDB compatibility) / Version 2.1

Copyright © 2022, Oracle and/or its affiliates / Public

ORACLE

Table of contents

ORACLE

Purpose

This document provides an overview of features and enhancements included in Oracle Database releases 19c and 21c and related Oracle technologies. It is intended to help you understand why modern application development often uses JSON as data persistency format, and why the JSON capabilities in Oracle Database are perfectly suited to solve the requirements of today's developers looking for a document store to persist, query and process application data.

Schema-flexible Application Development

Modern application development takes place in a fluid environment. Users expect applications to adapt to rapidly changing business requirements and for updates to be delivered on the fly. All of this means that developers need a flexible data persistency mechanism with minimal downtime, or DBA involvement, when the application evolves. **The relational model lacks this flexibility**: tables have a static 'shape' and application changes require modifying them (for example to add new column) which typically involves a database administrator (DBA). Also, existing data may need to be modified to fit the new schema. Even more important: the relational approach requires an **upfront schema design**: Objects of the application (for example 'customer order') are *normalized* to the tables and columns that store the objects' values. One application object often gets normalized to multiple tables. This means that simple *put* or *get* operations now require *inserts* and *selects* involving all participating tables with the right join conditions. The developer has to understand this mapping and express it using SQL.

This approach- although proven to work for decades - is often considered too rigid, formal and slow for modern application development. Also, as application and database changes often have to be synchronized, so there is a higher chance of downtime and increased operational cost.

Document stores (also called document databases) work differently and do not require an upfront schema definition. Instead, application data is modelled as documents, typically in JSON format. Each document is self-describing (consists of named key/value pairs) and hence no external schema is needed to understand the values. Also, different documents can have different key/value pairs making it easy to evolve an application by adding new key/value pairs on the fly without having to modify existing data/documents. The use of documents for data persistency therefore delivers the flexible storage mechanism that developers ask for.

An additional requirement to handle JSON arises from the ubiquity of JSON-based APIs: REST services operate with JSON input and outputs. Mapping these JSON values to tables could cause the application to break if the third-party API changes and no longer matches the table. Instead, JSON data is better stored "as is" in a database that supports queries over JSON data

Limitations of NoSQL Document Stores

Developers often gravitate toward NoSQL products because they are perceived as easier to use than relational databases. A typical NoSQL document store organizes JSON documents inside collections. Because the data model is simple, consisting solely of collections and documents, the functionality provided by these systems is also simple, and particularly limited when it comes to reporting or analytical use

ORACLE

cases. If such requirements arise, developers often deploy a second (relational) database and store the data twice; typically requiring an ETL process (*E*xtract, *T*ransform, *L*oad) to convert the data to the relational format. Also, NoSQL document stores typically do not support complex transactions and referential integrity constraints, so that data consistency now becomes the developer's problem. Required 'work arounds' increase system complexity, reduce security, allow inconsistencies and creates new problems like a point -in-time recovery across different databases. Because of this added complexity the total cost of ownership tends to be high and no longer delivers on the promise of simple NoSQL products.

Using Oracle Database as a JSON Document Store

Oracle Database provides the same application-development experience as a special-purpose NoSQL document store: It can store, manage, and index JSON documents and it provides NoSQL-like document-store APIs similar to those of common NoSQL-products. It even supports an API that is compatible to MongoDB - one of the most popular document stores. Additionally (and unlike NoSQL products), Oracle Database provides sophisticated SQL querying, reporting, analytics and machine learning over JSON documents. This lets you integrate JSON and relational data, joining them in the same query. And because JSON features are integrated into Oracle Database, all of its enterprise features for availability, security, scalability, performance, and manageability are fully supported for JSON data.

Storing and Managing JSON Documents in Oracle Database

Oracle Database Release 21c adds a new SQL datatype 'JSON' that uses a binary format optimized for fast queries and piecewise updates. Earlier releases like 19c allow to store JSON documents using VARCHAR2, CLOB, or BLOB columns. An "IS JSON" SQL check constraint ensures that the column contains only valid JSON documents, allowing the database to understand that the column is being used as a container for JSON documents.

Oracle's JSON capabilities are focused on providing full support for schema-flexible development and document-based storage. Consequently, although Oracle Database knows that a given column contains JSON documents; those documents are stored, indexed and queried without the database having any knowledge of their internal structure (the key/value pairs). Developers are free to change the structure of JSON documents as necessary.

Oracle Database provides full JSON support for all of its advanced features, including disaster recovery, replication, compression, and encryption. Additionally, products that support Oracle Database, such as Oracle Golden Gate and Oracle Data Integrator, (as well as third party tools) seamlessly support JSON documents stored in the database.

"Independent benchmarking of JSON databases based on the Yahoo! Cloud System benchmark revealed that Oracle is by far the leader in the space, outperforming all competitors [...]"

Accenture technical report: Increase agility and cut development time with JSON and Oracle, 2021 *https://accntu.re/3Iezy00*

ORACLE

Autonomous JSON Database

Oracle Database has supported JSON since release 12.1.0.2 and many JSON features have been added since. A managed database cloud service called 'Autonomous JSON Database' (AJD) provides the functionalities outlined in this technical report, at a price point that is significantly lower than the other members of the Autonomous Database family. AJD, on top of supporting document store APIs is fully capable of running arbitrary SQL and storing non-JSON data in relational tables. As AJD caters towards JSON developer there is a 20GB limit on non-JSON data; if more is needed then an upgrade to the Autonomous Transaction Processing (ATP) service can be done with a single mouse click. AJD is therefore not a separate development environment requiring different skills or APIs.

As part of the Autonomous Database platform, AJD users fully benefit from the self-driving, self-securing and self-repairing capabilities of the Autonomous Database. Database uptime is maximized and auto-scaling (up to three times configured CPU limits) provides maximum performance at minimum cost.

More information on the Autonomous JSON Database service can be found here: https://www.oracle.com/autonomous-database/autonomous-json-database/

Oracle Database API for MongoDB for Autonomous Databases

All Oracle Autonomous Database - including the Autonomous JSON database - are MongoDB compatible: tools, drivers and applications written for MongoDB can connect to an Oracle Autonomous Database using a native API for MongoDB which translates MongoDB database operations transparently into equivalent SQL/JSON operations that are then executed on the Oracle Database. MongoDB applications communicate through the MongoDB API if it they were still connected to a MongoDB server. Developers can continue to use their MongoDB skills and tools whilst now being able to also run SQL statements over the JSON data in the MongoDB collections. This enables real-time SQL analytics and machine learning over JSON data. It is also possible to generate JSON from relational data and expose the result as a MongoDB-compatible collection such that query results or relational data can easily be made accessible to MongoDB applications.

The Oracle Database API for MongoDB also supports MongoDB tools like Compass, mongo shell and mongoimport/mongorestore, therefore simplifying migrations to Oracle.

As of today (Feb 2022) the Oracle Database API for MongoDB is initially only available on shared Autonomous Databases. Details can be found here: http://docs.oracle.com/en/database/oracle/mongodb-api/mgapi

Simple Oracle Document Access API (SODA)

As the 'Oracle Database API for MongoDB' is currently limited to shared Autonomous Databases, Oracle provides another document-store API that is available universally: in the cloud (all Oracle cloud databases) as well as on-premise: The *Simple Oracle Document Access* (SODA) API. This API was designed from the ground up to support schema-flexible application development and is very similar to common No-Sql document store APIs like MongoDB's.

"Autonomous JSON is the product that MongoDB hopes to have in the next decade:
With Full ACID, full parallel analytics, fast updates, open standards-based JSON, full indexing, and support for everything from blockchain to spatial to graph, Oracle is helping MongoDB developers leap into the future."

Mark Staimer
Wikibon

ORACLE

Using SODA, developers can work with JSON documents and collection without having to learn SQL. Instead, database operations on collections and documents can be called directly from a simple API - which is available for REST as well as the popular programming languages Java, Python, JavaScript (Node.js), C, and PL/SQL. SODA for REST is part of Oracle Rest Data Services (ORDS) and can be invoked from any language that is capable of making REST/HTTP calls. Java, Python, Node.js and C drivers are open source.

The conceptual model of SODA is very similar to MongoDB: application objects are stored as JSON *documents* in *collections*. Documents are identified with a *key*. Collections are identified with a name. Heterogenous collections allow storing non-JSON objects, for example images. Multiple collections reside in a *database* which a client program connects to.

Documents can be accessed using SODA commands, typically for simple CRUD operations (create, read + find, update, delete) but also using SQL: reporting, analytics or machine learning can easily be done one the same JSON data.

We illustrate the SODA API with examples for REST and Java. The SODA documentation with links to the drivers and tutorials can be found here:

https://www.oracle.com/database/technologies/appdev/json.html

SODA Examples

The following Java code creates a collection 'orders' and inserts a JSON document. It then retrieves the unique key (id) that SODA assigned to the document. SODA can also accept user-generated keys.

```
OracleRDBMSClient client = new OracleRDBMSClient();
OracleDatabase    db = client.getDatabase(conn);
OracleCollection  orders = db.admin().createCollection("orders");
OracleDocument    doc = orders.insertAndGet(db.createDocument('{…}'));
String id = doc.getKey();
```

As one can see the database, collections and documents map to Java classes which have functions exposing their functionalities.

In SODA for REST, HTTP verbs such as PUT, POST, GET, and DELETE map to SODA operations over documents. The URL contains the document's key or the collection's name, together with the database host name and authorization credentials. SODA for REST is an Oracle REST Data Service and relies on ORDS for authentication and authorization. The examples omit this for space reasons. The two operations, create collection and insert a document require a REST call each. The second call yields a HTTP response with the assigned key (id):

```
curl –X PUT http://<authUrlToOrds>/soda/latest/orders

curl –X POST –H "Content–type: application/json"
––upload–file document.json http://<urlToORDS>/soda/latest/orders

{
   "items": [
      {
         "id": "A450557094D04957B36346F630CDDF9A",
         "etag":"C13578001CBBC84022DCF5F7209DBF0E6DFFCC626E3B0400C3",
         "lastModified": "2021–02–09T01:03:48.291462",
         "created": "2021–02–09T01:03:48.291462"
      }
   ],
   "hasMore": false, "count": 1
}
```

"We use Autonomous Database, JSON and REST for modern fleet management solutions. JSON simplifies application changes, customization and is naturally supported by mobile devices.

Dr. Jürgen Stausberg, CEO
Satlog GmbH
www.satlog.de/en/home/

ORACLE

The examples above show how a document-store differs from a traditional SQL database: new documents are added to a collection as JSON objects. There are no restrictions imposed by the database on the keys contained in those documents. And the API calls are simpler for developers who may be accustomed to object-oriented programming environments.

Note: a difference between SODA for REST and the other language drivers (for example Java) is that REST is stateless and therefore all REST operations are immediately committed, whereas the language drivers rely on database connections which support transactions (multiple operations can be made atomic).

Now, let's use SODA to retrieve documents. SODA obviously supports fetching a document by its key, but a more interesting way of querying data is to find all documents that satisfy a search condition - expressed as a JSON document itself - we call this Query By Example (QBE). Here is a very basic QBE that selects all documents which have field "region" whose value is "north" and a second field "quantity" whose value is 10 or greater:

```
{"region":"north", "quantity":{"$gte":10}}
```
(It is possible to paginate through large results using skip and limit.)

The following Java snippet performs a QBE search, limits the result to the first 100 document and prints all document keys. The variable 'qbe' holds above QBE.

```
OracleCollection coll = database.openCollection("orders");
OracleCursor results = coll.find().filter(qbe).limit(100).getCursor();
while (results.hasNext())
{
   OracleDocument doc = results.next();
   System.out.println(doc.getKey());
}
```

The REST parameter action=query indicates that a POST contains a QBE request.

```
curl -X POST -H "Content-type: application/json" --data
'{"region":"north", "quantity":{"$gte":10}}'
http://<urlToORDS>/ords/SCOTT/soda/latest/orders?action=query
```

Analytics and Reporting on JSON Content Stored in Oracle Database

As shown Oracle Database provides all of the advantages of a NoSQL document store for application development. A major advantage of using Oracle Database is that it also provides the full power of SQL to be applied to the same JSON documents. This can be done because JSON collections are backed by regular tables which are automatically created. They contain a JSON column to store the document and additional columns for the unique key (ID) and metadata such as the creation date. The *orders* collection is backed by a table:

```
SQL> describe "orders"

NAME                      NULL?       TYPE
------------------------- -------- ---------------------------
ID               NOT NULL  VARCHAR2(255)
CREATED_ON       NOT NULL  TIMESTAMP(6)
LAST_MODIFIED    NOT NULL  TIMESTAMP(6)
VERSION          NOT NULL  VARCHAR2(255)
JSON_DOCUMENT              JSON
```

ORACLE

Oracle Database supports a wide range of SQL operators to work with JSON:

IS JSON	Tests if an expression contains JSON
JSON_Value	Extracts a scalar SQL value
JSON_Query	Extracts a JSON fragment
JSON_Exists	Tests if one or more conditions is met
JSON_TextContains	Full text search on JSON fields
JSON_Table	Projects JSON to relational model
JSON_Object[Agg]	Generates a JSON object
JSON_Array[Agg]	Generates a JSON array
JSON_Transform	Modify JSON, e.g. as part of update
JSON_Mergepatch	Merges two JSON objects
JSON_Dataguide	Samples JSON to build schema

Many operators rely on path expressions to navigate inside the JSON data and optionally perform filters using path predicates. Detailed description of operators and path expression can be found in the JSON Developer Guide:

https://docs.oracle.com/en/database/oracle/oracle–database/21/adjsn/

For the following examples we assume that this collection/table contains purchase order documents:

```
{
    "PONumber": 1600,
    "Reference": " ABULL–20140421",
    "Requestor": "Alexis Bull",
    "User": "ABULL",
    "CostCenter": "A50",
    "Instructions": {
        "name": "Alexis Bull",
        "Address": {
            "street": "200 Sporting Green",
            "city": "South San Francisco",
            "state": "CA",
            "zipCode": 99236,
            "country": "United States of America"
        },
        "Phone": [
            {
                "type": "Office",
                "number": "823–555–9969"
            }
        ]
    },
    "Special Instructions": "Counter to Counter",
    "LineItems": […]
}
```

The easiest way to use SQL to query JSON data is the *simple dot notation* which allows to navigate the JSON structure and select values.

```
select j.PO_DOCUMENT.Reference,
       j.PO_DOCUMENT.Requestor,
       j.PO_DOCUMENT.CostCenter,
       j.PO_DOCUMENT.Instructions.Address.city
from J_PURCHASEORDER j
where j.PO_DOCUMENT.PONumber = 1600;


REFERENCE        REQUESTOR       COSTCENTER      SHIPPINGINSTRUCTIONS
ABULL–20140421  Alexis Bull     A50             South San Francisco
```

> *"We heavily use JSON whenever faced with unpredictable data from external APIs or custom user extensions. We decided to use Oracle Database as a document store that also supports SQL analytics over JSON and Blockchain."*
>
> **Peter Merkert, CTO**
> Retraced
> *www.retraced.co*

ORACLE

JSON_TABLE is a table function commonly used to project JSON to the relational model, in order to access it as if it was a table. This has many benefits:

- The relational model is highly suitable for analytical queries, in particular for warehousing-style queries were dimensions and facts are stored in separate collections. With materialized views it is even possible to 'precompute' these joins.

- Tools that operate on the relational model can be used to work with JSON, for example report builders, dashboards and machine learning can be applied directly over JSON data.

- SQL language and tuning skills can be applied by data analysts, there is no need to manually map JSON data to tables or rite custom code.

JSON_TABLE uses a set of JSON path expressions to project content from a JSON document as relational columns in a virtual table. You use a JSON_TABLE expression in the FROM clause of a SQL query, in the same way you would use a relational table. The following example projects a set of columns from a collection of JSON documents. Each JSON path expression returns a single scalar value from a document, so one row of the virtual table is generated for each document.

```
select jt.*
from J_PURCHASEORDER p,
JSON_TABLE(p.PO_DOCUMENT ,'$'columns
    PO_NUMBER  NUMBER(10) path '$.PONumber',
    REFERENCE  VARCHAR2(30 CHAR) path '$.Reference',
    REQUESTOR  VARCHAR2(32 CHAR) path '$.Requestor',
    USERID     VARCHAR2(10 CHAR) path '$.User',
    COSTCENTER VARCHAR2(16 CHAR) path '$.CostCenter',
    TELEPHONE  VARCHAR2(16 CHAR) path '$.Instructions.Phone[0].number'
) jt
where PO_NUMBER > 1599 and PO_NUMBER < 1602;


PO_NUMBER  REFERENCE       REQUESTOR      USERID   COSTCENTER    TELEPHONE
-----------------------------------------------------------------------
1600      ABULL-20140421  Alexis Bull    ABULL    A50           909-555-7307
1601      ABULL-20140423  Alexis Bull    ABULL    A50           909-555-9119

2 rows selected.
```

JSON_TABLE also supports JSON documents with nested arrays.: the NESTED PATH iterates over the nested "LineItems" array: Values outside the nested array are being repeated (PO_NUMBER) as they apply for the whole nested array.

```
select jt.*
from J_PURCHASEORDER p,
    JSON_TABLE(p.PO_DOCUMENT, '$' columns(
        PO_NUMBER NUMBER(10) path '$.PONumber',
        REFERENCE VARCHAR2(30 CHAR) path '$.Reference',
        NESTED PATH '$.LineItems[*]'columns(
            ITEMNO      NUMBER(16)   path '$.ItemNumber',
            DESCRIPTION VARCHAR2(32) path '$.Part.Description',
            UPCCODE     VARCHAR2(14) path '$.Part.UPCCode',
            QUANTITY    NUMBER(5,4)  path '$.Quantity',
            UNITPRICE   NUMBER(5,2)  path '$.Part.UnitPrice')
        )
)jt
where PO_NUMBER > 1599 and PO_NUMBER < 1602;


PO_NUMBER REFERENCE ITEMNO DESCRIPTION UPCCODE QUANTITY UNITPRICE
```

ORACLE

```
     ---------------------------------------------------------------
     1600      ABULL-20140421 1 One Magic Christmas  13131092  9   19.95
     1600      ABULL-20140421 2 Lethal Weapon        8539162   5   19.95
     1601      ABULL-20140423 1 Star Trek 34         9736600   1   19.95
     1601      ABULL-20140423 2 New Blood            4339605   8   19.95
     1601      ABULL-20140423 3 The Bat              1313111   3   19.95
     1601      ABULL-20140423 4 Standard Deviants    6318650   7   27.95
     1601      ABULL-20140423 5 Darkman 2            2519203   7   19.95

     7 rows selected
```

With JSON_TABLE arbitrary complex JSON structures can be projected to the relational model. A JSON_TABLE query can be exposed as a view - for any consumer of the view the JSON data is accessed like a conventional table consisting of just rows and column with scalar value. This also enables the use or relational tools that do not support the JSON data model.

JSON Dataguide

One common use of JSON_TABLE is to create relational views which allow users and tools that have no understanding of JSON to work with documents. JSON_Dataguide can automate the view creation, by sampling all JSON documents in a collection and identifying field names and data types. The following example shows how the view 'order_view' is auto- created. The view definition contains a JSON_Table expression similar to the ones above.

```
declare
  dg CLOB;      -- this variable stores the derived JSON schema
begin
  -- JSON_Dataguide samples all documents and builds a JSON schema
  select JSON_Dataguide(json_document, dbms_json.FORMAT_HIERARCHICAL) into dg
  from orders;

  -- using this JSON schema a JSON_TABLE view can be automatically created
  dbms_json.create_view('order_view', 'orders', json_document', dg);
end;
/
```

JSON Generation

Oracle Database is also able to generate new JSON data - from relational as well as from JSON data. This allows, for example, the generation of a report in JSON format.

The following example show how data in the sample tables employees and departments is joined and the result returned as a new JSON document.

```
select JSON_ObjectAgg(d.name VALUE (
    select JSON_ArrayAgg(JSON_Object (e.name))
    from employees e
    where e.department_no = d. department_no)
from departments d;

------------------------
{"ACCOUNTING":[
      {"name":"CLARK"},
      {"name":"KING"},
      {"name": "MILLER"}
   ],
 "RESEARCH":[
      {"name": "SMITH"},
      {"name": "JONES"},
 ...
```

"Autonomous JSON helps consolidate database footprint and reduces distractions that plague app dev teams.... making them more productive."

Mark Peters
ESG
bit.ly/AJD_ESG

ORACLE

Note that the SQL/JSON generation operators are simply added to an otherwise conventional SQL query, showing again how well JSON and tables can be used together in Oracle Database. It is possible also to insert the generated documents into a collection to make it accessible to SODA or the MongoDB API.

Conclusion: Why Use Oracle Database as a Document Store?

This technical report has described Oracle Database features that support schema-flexible development using JSON documents stored inside collections. Today, many NoSQL systems support this development paradigm. Why should an organization choose to use Oracle Database instead of a NoSQL system?

Oracle Database was built for enterprise applications. Many capabilities that are taken for granted with a modern enterprise class relational database are simply not provided by the typical NoSQL document store:

- Sophisticated indexing, query optimization and parallel execution
- Fully ACID compliant transactions without size/duration limits.
- Advanced security features such as data masking and key management
- Data management features such as compression and data archiving
- Robust backup capabilities with object-level point-in-time recovery
- Built-in procedural languages and server-side functions

NoSQL systems typically lack the functionality for reporting and analytical operations. As the volume and value of the JSON documents increases, there is a growing need to be able to perform cross-document reporting and analysis. Developers previously had to export data from NoSQL and apply a complex ETL (*e*xtract, *t*ransform, and *l*oad) process to make it available to a data store that supports flexible reporting. While many NoSQL systems are now recognizing the need for a tabular, structured format for accessing data, and some are even introducing basic SQL-like languages, Oracle Database delivers the full power of mature ISO-standard SQL to JSON document stores today, with its advanced SQL analytic capabilities and scalable parallel SQL infrastructure.

Organizations using NoSQL document stores also have to face the issue that their data can become siloed: their relational data is managed by one database and their JSON documents are managed by another. Using a separate data store for JSON documents means that when it becomes necessary to combine information that has been stored as JSON with other kinds of data that the organization manages (which typically includes relational data), special application code needs to be developed and maintained to accomplish even rudimentary tasks.

The Oracle Converged Database delivers document-store functionality designed for application developers, while allowing these application developers to leverage all of the other benefits of Oracle's mature database platform

"Oracle Autonomous JSON Database is between 2.3 and 3.2 times faster than MongoDB Atlas and between 2.0 and 4.1 times faster than AWS DocumentDB"

David Floyer
Wikibon
https://bit.ly/AJD_Wikibon

ORACLE

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

🅱 blogs.oracle.com   f facebook.com/oracle   🐦 twitter.com/oracle

12   Business / Technical Brief / JSON-based
Application Development
with Oracle Database
(and MongoDB compatibility) / Version 2.1

Copyright © 2022, Oracle and/or its affiliates / Public

ORACLE