25 Years of HrOUG!

1

Make your PL/SQL pop with SODA

Presented on 15th October 2021 at

HrOUG 2021

Rovinj, Croatia

by

Niall Mc Phillips - Long Acre sàrl

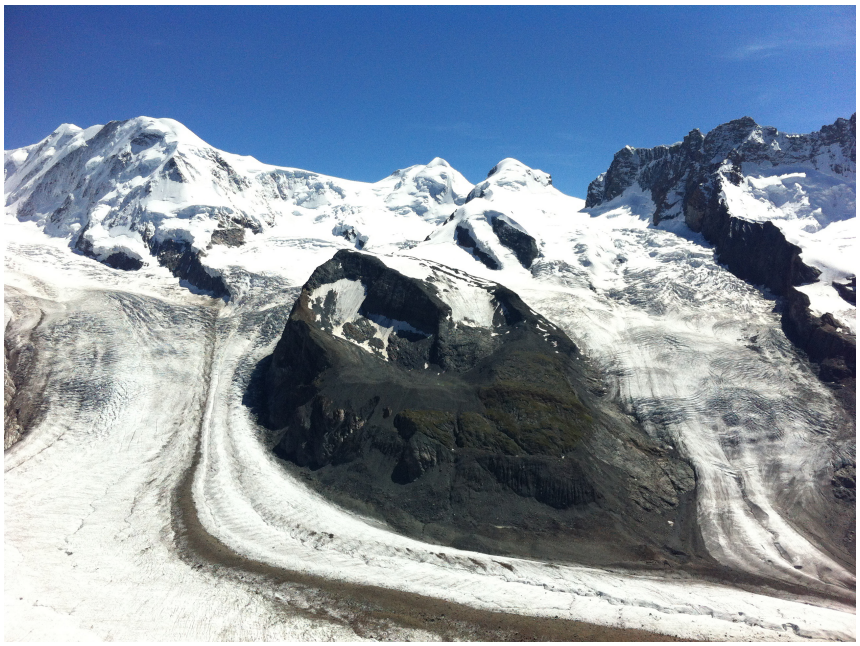
niall.mcphillips@longacre.ch

@Niall_McP

2

3

4



5

6

About me: Niall Mc Phillips

Owner - Long Acre sàrl (founded 2015)
Co-owner and Director - Stephenson and Associates (founded 1995)
Irish / Swiss Living in Geneva, Switzerland.

- Oracle ACE
- Using Oracle database as a Developer and DBA for >30 years
- Developing web applications with Oracle DB since 1995
- Developing with APEX since 2005 (HTML DB 1.6)
- Organizer of the Swiss APEX Meetup group

@NiallMcP
niall.mcphillips@longacre.ch

7

500+ Technical Experts Helping Peers Globally

ORACLE ACE Program

ORACLE ACE Director | ORACLE ACE | ORACLE ACE Associate

3 Membership Tiers
- Oracle ACE Director
- Oracle ACE
- Oracle ACE Associate

bit.ly/OracleACEProgram

Connect:
oracle-ace_ww@oracle.com
Facebook.com/oracleaces
@oracleace

Oracle Groundbreakers

Nominate yourself or someone you know: acenomination.oracle.com

8

What is Oracle SODA

"Simple Oracle Document Access (SODA) is **a set of NoSQL-style APIs that let you create and store collections of documents (in particular JSON) in Oracle Database**, retrieve them, and query them, without needing to know Structured Query Language (SQL) or how the documents are stored in the database."

SODA basic concepts - Documents

A SODA Document is a JSON object accompanied by some metadata.

```
{
  "id": 1488,
  "date": "308/06/26",
  "description": "On June 26 Pope Marcellus I succeeds Pope Marcellinus as the 30th pope."
}
```

SODA basic concepts - Collections

SODA Documents are grouped into SODA Collections

```
{
  "id": 1488,
  "date": "308/06/26",
  "description": "On June 26 Pope Marcellus I succeeds Pope Marcellinus as the 30th pope."
}
{
"id": 1498,
"date": "309",
"description": "Pope Marcellus I is banished from Rome, as is his successor Eusebius later that
      year."
}
{
"id": 1499,
"date": "309/04/18",
"description": "Pope Eusebius succeeds Pope Marcellus I as the 31st pope."
}
```

SODA basic concepts

For those of us with a relational DB background:

Collections ≈ Tables

Documents ≈ Rows

SODA implementations

- SODA for PL/SQL
- SODA for REST
- SODA for Node.js
- SODA for Python
- SODA for Java
- SODA for C

Some SODA Operations

- Create collection
- Open existing collection
- Drop collection
- List existing collections
- Insert documents into a collection
- Find documents in a collection
- Index a collection
- …

Setting up SODA – SODA_APP role

Grant the SODA_APP role to your user.

From an Admin account:

```
grant soda_app to niall;
```

Setting up SODA – Enable for ORDS

Enable your schema for ORDS

```
BEGIN
  ORDS.enable_schema
   (p_enabled => TRUE,
    p_schema => 'SODAUSER',
    p_url_mapping_type => 'BASE_PATH',
    p_url_mapping_pattern => 'sodauser',
    p_auto_rest_auth => FALSE );
 COMMIT;
END;
/
```

Create a Collection

Create a collection – use *dbms_soda.create_collection*

```
declare
  v_collection  soda_collection_t;
begin
  v_collection :=
      dbms_soda.create_collection
                ('sodaHistorical');
end;
/
```

List all collections

Use dbms_soda.list_collection_names

```
set serveroutput on
declare
  v_collections  soda_collname_list_t;
begin
  v_collections := dbms_soda.list_collection_names;

  if v_collections.count > 0 then
    for iLoop in 1 .. v_collections.count
    loop
      dbms_output.put_line
         ('Collection name: ' || v_collections(iLoop));
    end loop;
  end if;
end;
/
```

Examining a collection using SQL

```
select * from user_tables
 where lower(table_name) like 'soda%'


desc "sodaHistEvents"


select * from "sodaHistEvents";
```

(note the Oracle-assigned key)

Drop a collection

Use dbms_soda.drop_collection

```
set serveroutput on
declare
    v_status  number;
begin
    v_status := dbms_soda.drop_collection('sodaHistorical');
end;
/
```

SODA types

Type	Description
SODA_Collection_T Type	This SODA type represents a SODA collection. This type is not persistable.
SODA_Document_T Type	This SODA type represents a document with content, usually in JSON format. This type is not persistable.
SODA_Operation_T Type	This SODA type performs read/write operations, such as document finds with filtering and pagination, removes, and replaces on a SODA collection. This type is not persistable.
SODA_Cursor_T Type	This SODA type represents the result set of documents. This type is not persistable.

SODA_Collection_T type functions

Subprogram	Description
CREATE_INDEX Function	Creates an index using an index specification expressed in JSON. Three types of specifications are supported. Each specifying a different type of index: for B-tree, JSON search with Data Guide, and Spatial.
DROP_INDEX Function	Drops the named index.
FIND Function	Returns the SODA_OPERATION_T object. This is the only way to get the reference of SODA_Operation_T as there is no constructor.
FIND_ONE Function	Fetches the document matching the key.
GET_DATA_GUIDE Function	Returns the JSON data guide as a CLOB.
GET_METADATA Function	Returns the metadata of the collection in JSON format.
GET_NAME Function	Returns the name of the collection.
INSERT_ONE Function	Inserts a document into the collection.
INSERT_ONE_AND_GET Function	Inserts a document into the collection and returns a result document with all components except for content.
REMOVE_ONE Function	Removes the document matching the key.
REPLACE_ONE Function	Replaces the content and (optionally) the media type of the document matching the key.
REPLACE_ONE_AND_GET Function	Replaces the content and (optionally) the media type of the document matching the key and returns a result document with all components (except content).

SODA_Operation_T type functions

Subprogram	Description
COUNT Function	Returns a count of the number of documents in the collection that match the criteria. If skip(...) or limit(...) were chained together with this count(), an exception is raised.
FILTER Function	Sets the filter (also known as QBE or query-by-example) criteria on the operation. Returns the same SODA_OPERATION_T object so that further criteria can be chained together if required.
GET_CURSOR Function	Returns a SODA_CURSOR_T object that can be used to iterate over the documents that match the criteria.
GET_ONE Function	Returns a single SODA_DOCUMENT_T object that matches the criteria. Note that, if multiple documents match the criteria, only the first document is returned.
KEY Function KEYS Function	Specifies that the document with the specified key should be returned. This causes any previous calls made to this function and KEYS(...), when they appear in the same chain, to be ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.
LIMIT Function	Sets a limit on the specified number of documents the operation should return. This setting is only usable for read operations such as GET_CURSOR. For write operations, any value set using this method is ignored. Returns the same SODA_OPERATION_T object so that further operation criteria can be chained together, if needed.
REMOVE Function	Removes all of the documents in the collection that match the criteria. Returns the number of documents that was removed.
REPLACE_ONE Function	Replaces a single document in the collection with the specified document. Returns a number that indicates if the document was replaced or not. Currently, before calling this function, you must call the function KEY(...) to uniquely identify the document being replaced. Any components set in the input document with the exception of content and media type are not used during the replace. They are ignored.
REPLACE_ONE_AND_GET Function	Replaces a single document in the collection with the specified document. Returns a result document if the document was replaced, NULL otherwise. Currently, before calling this function, you must call the function KEY(...) to uniquely identify the document being replaced. This function is similar to REPLACE_ONE. The only difference is that REPLACE_ONE_AND_GET also returns the result document with updated components, such as version and last-modified timestamp. The result document does not contain the content component. Any components set in the input document with the exception of content and media type are not used during the replace. They are ignored.

Inserting into a collection

Just for the sake of this example, let's generate some json from an existing table.

```
select json_object(key 'id' is h.id,
                   key 'theDate' is h.thedate,
                   key 'category1' is h.category1,
                   key 'category2' is h.category2,
                   key 'description' is h.description)
  from hist_events h;

...
{"id":1,"theDate":"-297",
  "category1":"By place",
  "category2":"Greece",
  "description":"Demetrius Poliorcetes returns to Greece with
the aim of becoming master of Macedonia. While Demetrius is in
Greece, Lysimachus seizes his possessions in Asia Minor."
}
...
```

Inserting into a collection

Next let's create a a view

```
create view hist_events_json as
select json_object
         (key 'id' is h.id,
          key 'theDate' is h.thedate,
          key 'category1' is h.category1,
          key 'category2' is h.category2,
          key 'description' is h.description) as event_json
  from hist_events h;
```

Inserting into a collection

Use the INSERT_ONE method to insert the documents.

```
declare
    v_collection  soda_collection_t;
    v_document    soda_document_t;
    v_status      number;
begin
    -- open the collection
    v_collection := dbms_soda.open_collection('sodaHistorical');
    -- insert
    for rec in (select * from vw_hist_events_json)
    loop
        v_document := soda_document_t
            (b_content => utl_raw.cast_to_raw(rec.event_json));
        -- Insert a document
        v_status := v_collection.insert_one(v_document);
        -- dbms_output.put_line('Status: ' || v_status);
    end loop;
end;
```

Blob

Query-by-Example (QBE).

Queries are expressed as JSON. Comparison operators:

- $all — whether an array field value contains all of a set of values
- $between — whether a field value is between two string or number values (inclusive)
- $eq — whether a field value is equal to a given scalar
- $exists — whether a given field exists
- $gt — whether a field value is greater than a given scalar value
- $gte — whether a field value is greater than or equal to a given scalar
- $hasSubstring — whether a string field value has a given substring (same as $instr)
- $in — whether a field value is a member of a given set of scalar values
- $instr — whether a string field value has a given substring (same as $hasSubstring)
- $like — whether a field value matches a given SQL LIKE pattern
- $lt — whether a field value is less than a given scalar value
- $lte — whether a field value is less than or equal to a given scalar value
- $ne — whether a field valueis different from a given scalar value
- $nin — whether a field value is not a member of a given set of scalar values
- $regex — whether a string field value matches a given regular expression
- $startsWith — whether a string field value starts with a given substring

Query-by-Example (QBE).

Queries are expressed as JSON. Logical operators:

```
{"$and" : [ {"name" : {"$startsWith" : "Jas"}},
            {"drinks" : "tea"} ]
}

{"$or" : [ {"drinks" : "soda"},
           {"address.postcode" : {"$le" : 94000}} ]
}
```

QBE

Find all entries where the date is 1501/12/12
- the qbe string is: *{"theDate" : "1501/12/12"}*

There are comparison operators such as $gt, $gte, $lt, …

Find all entries where the date is between 1798 and 1799
{"theDate" : { "$gte" : "1798", "$lte" : "1799"} }

Extracting data from query results

Metadata
- key: v_document.get_key
- creation timestamp: v_document.get_created_on
- last modified timestamp: v_document.get_last_modified
- version: v_document.get_version

Extract using json_value:

```
v_blob := v_document.get_blob();
```

- date: json_value(v_blob, '$.theDate’)
- description: json_value(v_blob, '$.description')
- entire document: json_query(v_blob, '$' pretty)

Indexing a collection (1)

The default index is a text index on all elements.

```
v_collection :=
dbms_soda.open_collection('sodaHistorical');

-- define the index specification
v_indexspec  := '{"name" : "IND_SODAHISTORICAL$1"}';

-- Create the index
v_status := v_collection.create_index(v_indexspec);
```


Indexing a collection (2)

Create a B-Tree index on a specific element.

```
v_collection :=
dbms_soda.open_collection('sodaHistorical');

-- define the index specification
 v_indexspec :=
    '{"name"   : "IND_SODAHISTORICAL$2",
      "fields" : [{"path"     : "theDate",
                   "datatype" : "string",
                   "order"    : "asc"}]}';

-- Create the index
v_status := v_collection.create_index(v_indexspec);
```

Take a look at the underlying queries

```
select * from v$sqlarea
 where upper(sql_text) like '%JSON_DOCUMENT%';
```


Take a look at the underlying queries

Examples found:

```
SELECT COUNT("ID")
  FROM "NIALL"."sodaHistorical"
 WHERE JSON_TextContains("JSON_DOCUMENT",'$.description', :T0);

SELECT COUNT("ID")
  FROM "NIALL"."sodaHistorical"
 WHERE  JSON_EXISTS("JSON_DOCUMENT",'$?(@.theDate > $B0)'
                      PASSING :B0 AS "B0")
```

Using SODA collections from SQL

You can use simple JSON dot notation to surface JSON values in SQL statements

```
select d.json_document.theDate,
       d.json_document.description as title
  from "sodaHistorical" d
 where d.json_document.theDate like '1916/04%';
```

35

Using SODA collections from SQL

You can also use JSON_TABLE to surface JSON values in SQL statements

```
select jt.*
from "sodaHistorical" soda,
json_table(soda.json_document, '$'
columns
  (nested path '$[*]'
     columns (thedate varchar2(255) path '$.theDate',
              category1 varchar2(255) path '$.category1',
              category2 varchar2(255) path '$.category2',
              description varchar2(255) path '$.description',
              event_json format json path '$')))
as jt;
```

36

REST Interface demo

- Create a collection
- Delete a collection
- Insert a document into a collection
- Delete a document from a collection
- Retrieve from a collection
- QBE retrieve

Any Questions ??