JSON in Database

Demo User's Guide

Contents

Initial requirements

- SSH private key to Access the database server in the cloud. This private key is provided along with this manual.
- SSH client app, to login to the database servers
- Database servers public IP

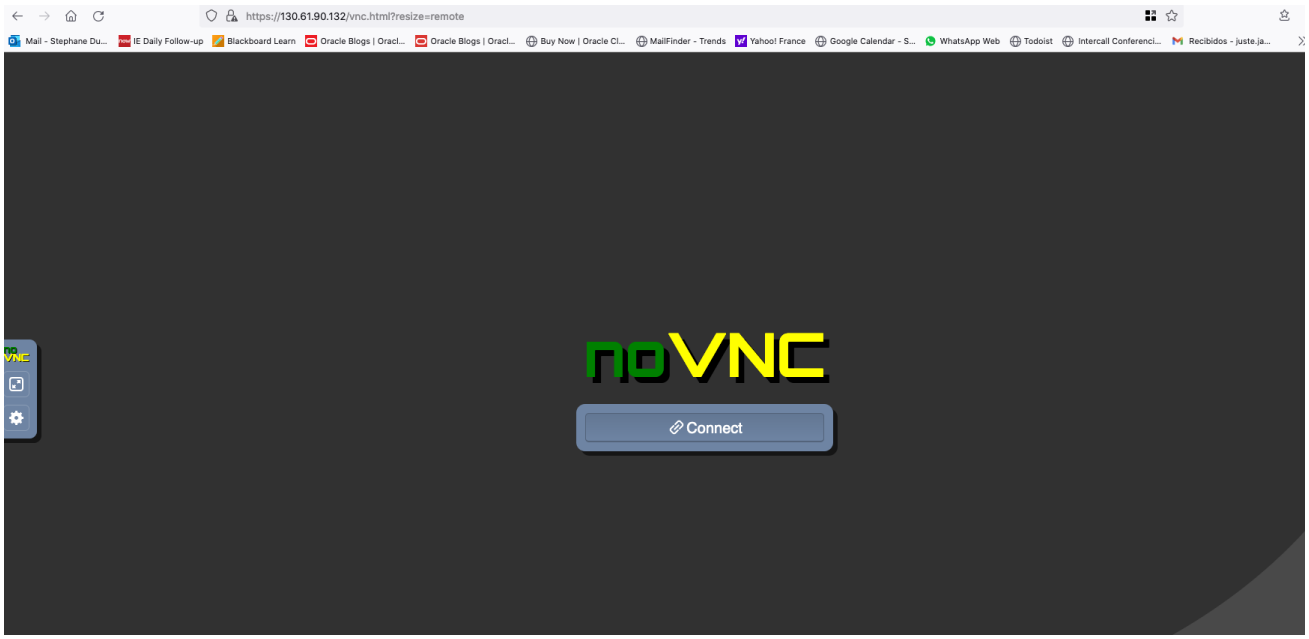
Access to the server

Gain access to the server: you can access your machine by either ssh or vnc.

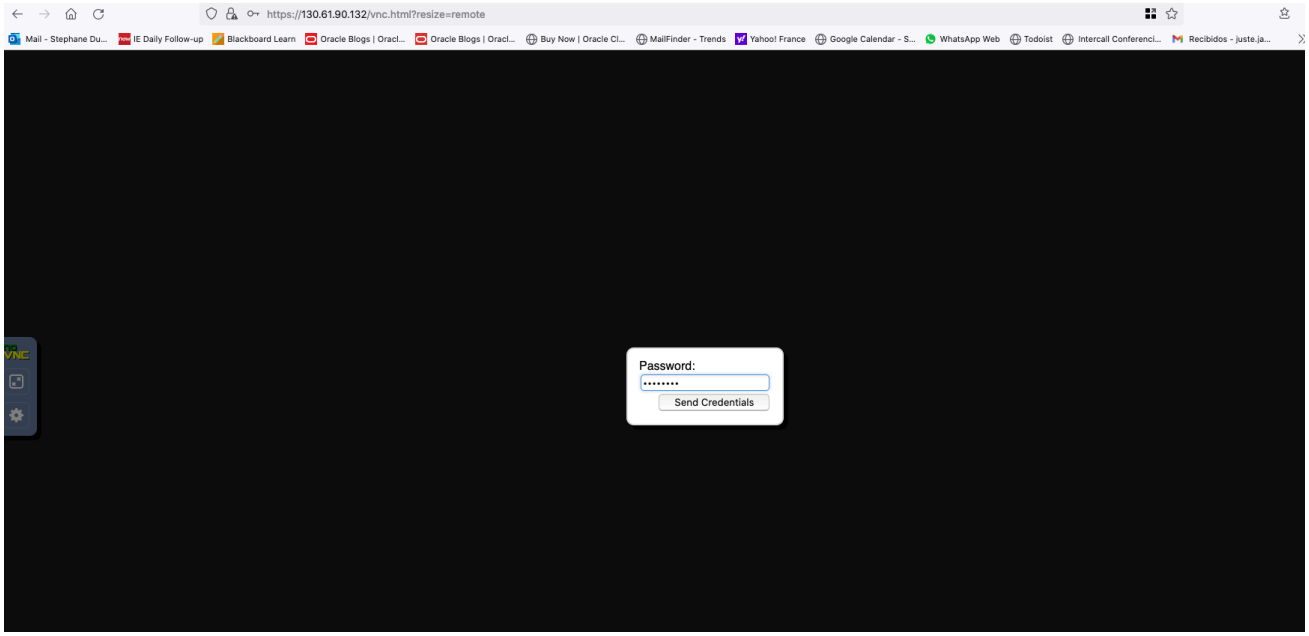
If you access using vnc, use the following URL in your browser:

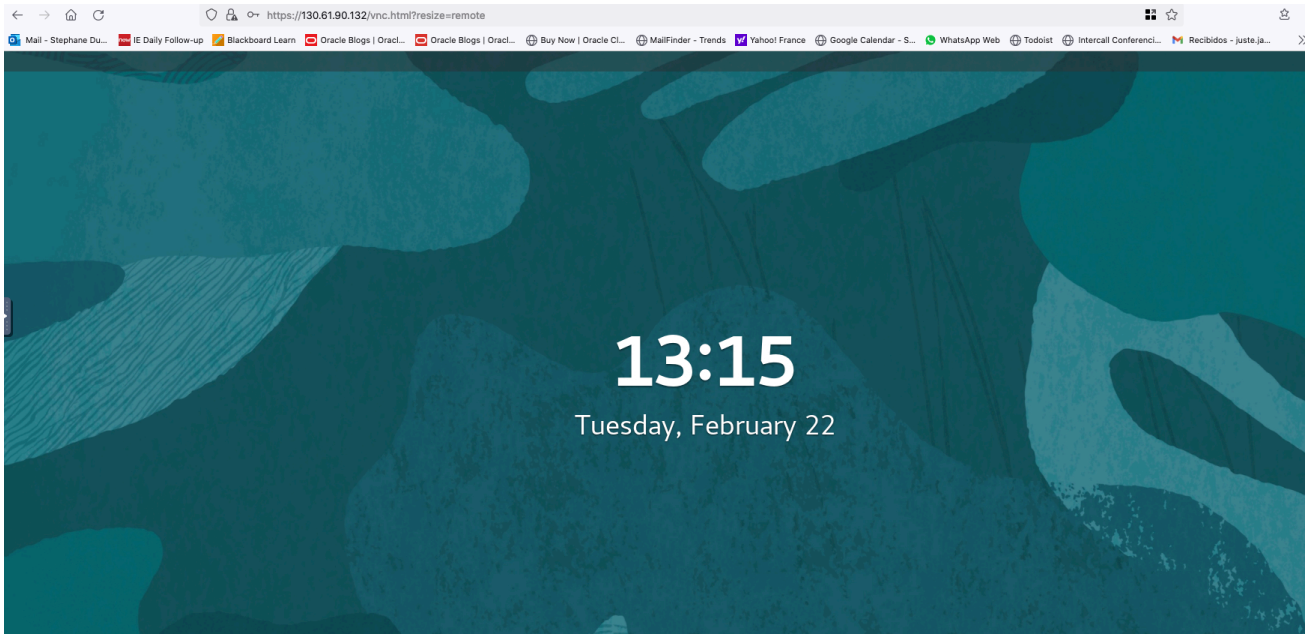
`https://<your public IP>/vnc.html?resize=remote`

You will be redirected to the following screen:



Click "Connect", and use the value provided by the instructor for the password.





Click on the screen and hit "Enter":

To run the lab, use the SQL script located in the "JSON_IN_DB" folder on the desktop (StepByStep.demo.sql). Use this script to copy and paste the commands in the terminal tool (runnable from the toolbar at the bottom of the screen).

If you use VNC access, open a terminal windows. You are already connected as user "oracle".

If you use ssh, access to the primary database server as "**opc**". Then connect as user "oracle":

```
ssh -i privateKey opc@<public ip of node 1>

## Connect as "oracle" user and show some database instance parameters

sudo su - oracle
```

You are now ready to run the demo.

SODA for Python

SODA stands for Simple Oracle Document Access. It provides drivers for python, PL/SQL, C, etc ... that allows to manage documents in the database, in a NoSQL manner. Using SODA you can create collections, insert documents into that collections, perform Query by Example, and so on.

In the next steps, we will present an example using SODA for Python.

Using the terminal window, ensure you are in "/home/oracle" directory, and review the content of the soda_basic.py script:

```
[oracle@myoracledb1 ~]$ ls -ltr | grep soda_basic.py
-rw-r--r--.  1 oracle oinstall     3415 Apr 20 10:00 soda_basic.py


[oracle@myoracledb1 ~]$ cat soda_basic.py
#------------------------------------------------------------------------------
# soda_basic.py
```

```
#   A basic Simple Oracle Document Access (SODA) example.
#
# This script requires cx_Oracle 7.0 and higher.
# Oracle Client must be at 18.3 or higher.
# Oracle Database must be at 18.1 or higher.
# The user must have been granted the SODA_APP privilege.
#------------------------------------------------------------------------------

import cx_Oracle as oracledb

connection = oracledb.connect("soe", "soe", "myoracledb:1521/orclpdb1")

# The general recommendation for simple SODA usage is to enable autocommit
connection.autocommit = True

# Create the parent object for SODA
soda = connection.getSodaDatabase()
[...]
```

Pay attention to the "import" command: we will use the "cx_Oracle" library, that allows connection to Oracle database, and includes the SODA for Python driver.

In this program, we will create a collection, create an index, insert, update and remove documents, and perform queries by example using filters with the "like$" and "$regex" operators.

Run the script from the operating system command line:

```
[oracle@myoracledb1 ~]$ python3 soda_basic.py
The key of the new SODA document is:  FE745F1679464FC3BF5E7CC044E2310D
Retrieved SODA document dictionary is:
{'name': 'Matilda', 'address': {'city': 'Melbourne'}}
Retrieved SODA document string is:
{"name": "Matilda", "address": {"city": "Melbourne"}}
Names matching 'Ma%'
Matilda
May
Collection has 4 documents
Removing documents
Dropped 2 documents
Collection has 2 documents
```

SODA from a CLI Tool

In the following steps, we are going to use a command line tool to execute some SODA commands: **sqlcl**.

```
unset ORACLE_HOME
cd /home/oracle/sqlcl/bin

./sql soe/soe@myoracledb:1521/orclpdb1

-- Get a list of collections in SOE schema:
SQL> soda list
```

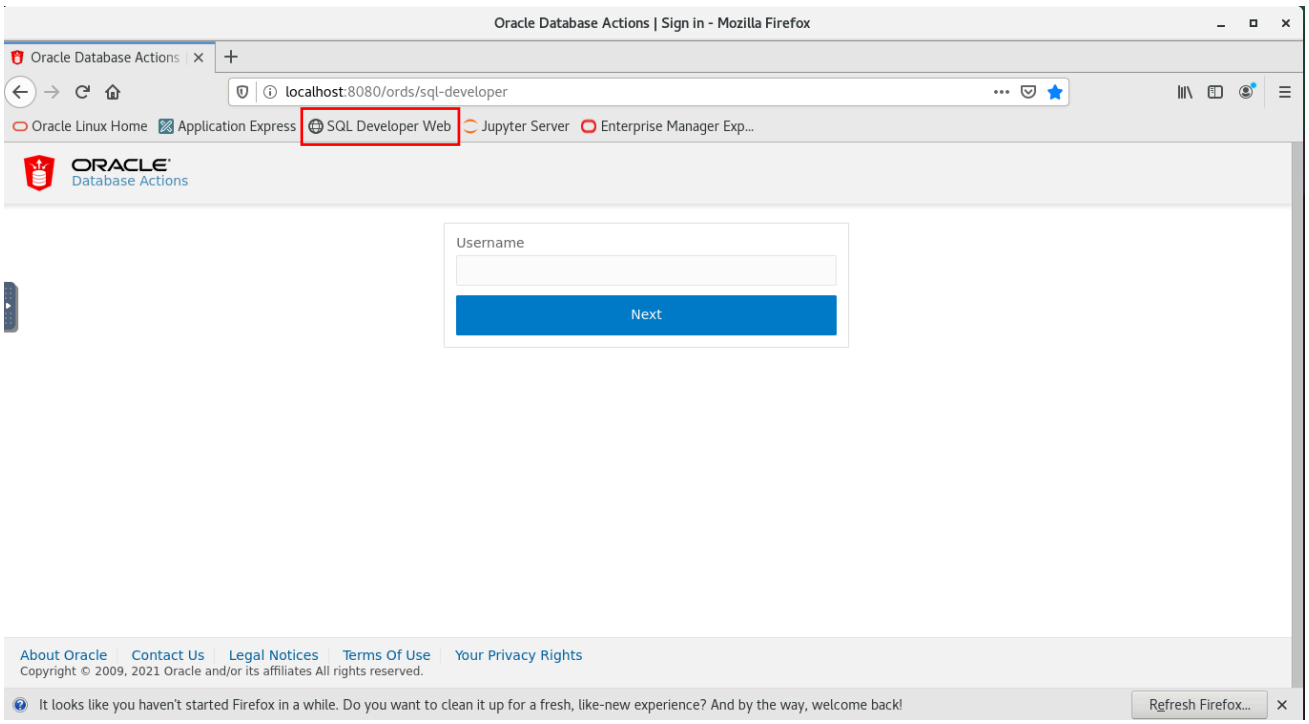
```
List of collections:

       mycollection

-- This is the collection we created with the python script !!!
-- Let's count the documents in that collection:

SQL> soda count mycollection

 2 rows selected.

-- Get all the documents keys in that collection

SQL> soda get mycollection -all
       KEY                                      Created On

       FE745F1679464FC3BF5E7CC044E2310D         2022-04-26T09:12:26.862897000Z
       F1968DFB875D4F33BF6FAAAABC074186         2022-04-26T09:12:26.878073000Z

 2 rows selected.

-- Get a particular document using its key (replace by your key)

SQL> soda get mycollection -k FE745F1679464FC3BF5E7CC044E2310D

Key:          FE745F1679464FC3BF5E7CC044E2310D
Content:      {"name": "Matilda", "address": {"city": "Sydney"}}
----------------------------------------

 1 row selected.

-- Insert a new document

SQL> soda insert mycollection {"name" : "Alex"}

 JSON document inserted successfully.

SQL> soda get mycollection -all
       KEY                                      Created On

       FE745F1679464FC3BF5E7CC044E2310D         2022-04-26T09:12:26.862897000Z
       F1968DFB875D4F33BF6FAAAABC074186         2022-04-26T09:12:26.878073000Z
       B99390DB7F95468EB6FF74D699301C88         2022-04-26T09:22:05.885546000Z

 3 rows selected.

SQL> soda get mycollection -k B99390DB7F95468EB6FF74D699301C88

Key:          B99390DB7F95468EB6FF74D699301C88
Content:      {"name" : "Alex"}
----------------------------------------

 1 row selected.

-- This new document hasn't the same structure as the existing ones. That's the power of
JSON, we can store schemaless (schema on read) information.

-- Now remove this document:
```

```
SQL> soda remove mycollection -k B99390DB7F95468EB6FF74D699301C88
Successfully removed 1 document.

-- And insert a new one:

SQL> soda get mycollection -all
      KEY                                  Created On

      FE745F1679464FC3BF5E7CC044E2310D     2022-04-26T09:12:26.862897000Z
      F1968DFB875D4F33BF6FAAAABC074186     2022-04-26T09:12:26.878073000Z

 2 rows selected.

SQL> soda insert mycollection {"name": "Mick", "address": {"city": "Sydney"}}

 JSON document inserted successfully.

SQL> soda get mycollection -all
      KEY                                  Created On

      FE745F1679464FC3BF5E7CC044E2310D     2022-04-26T09:12:26.862897000Z
      F1968DFB875D4F33BF6FAAAABC074186     2022-04-26T09:12:26.878073000Z
      C49DCD4D711C44A69F8E5FBED552B9D2     2022-04-26T09:26:41.664739000Z

 3 rows selected.

-- Now let's perform a Query by Example, applying a filter:

SQL> soda get mycollection -f {"address": {"city": "Sydney"}}

Key:          FE745F1679464FC3BF5E7CC044E2310D
Content:      {"name": "Matilda", "address": {"city": "Sydney"}}
----------------------------------------

Key:          C49DCD4D711C44A69F8E5FBED552B9D2
Content:      {"name": "Mick", "address": {"city": "Sydney"}}
----------------------------------------

 2 rows selected.

-- We can also create a new collection:

SQL> soda create mynewcollection

 Successfully created collection: mynewcollection

SQL> soda insert mynewcollection {"name": "Charlie", "address": {"city": "London"}}

 JSON document inserted successfully.
```

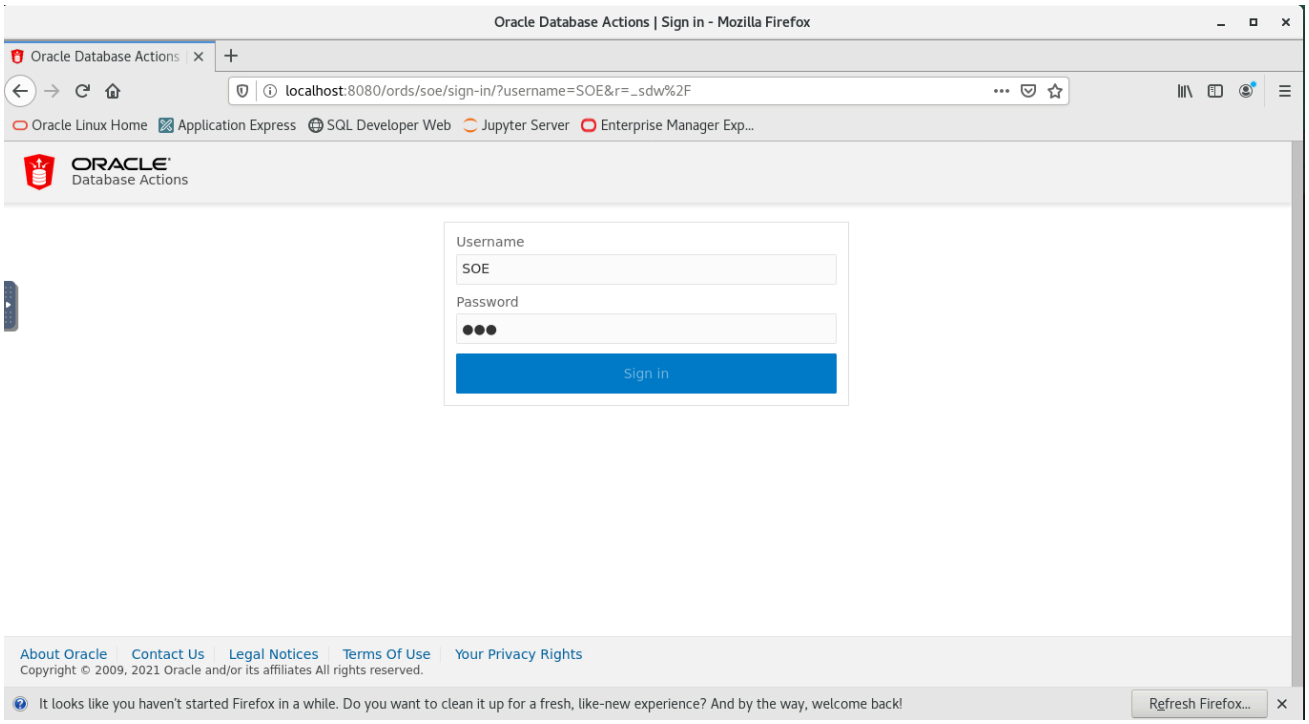
SODA from a GUI Tool

Along with sqlcl, a graphical tool is provided: "Database Actions". From a web interface, we can manage our collections just the way we did it from the CLI. "Database Actions" is provided "out of the box" with the Autonomous Database offering, but can be configured on any database, in the cloud or on premises, along with ORDS (Oracle Rest Data Sevices).

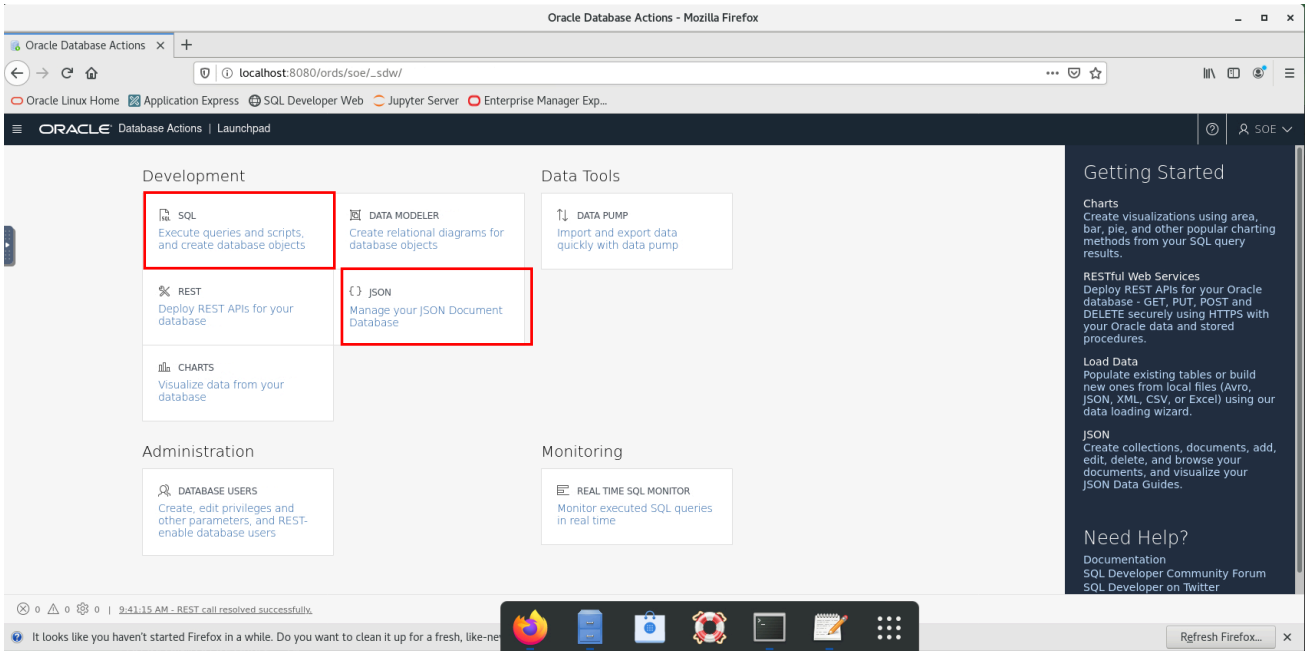
Connect to your server through VNC, and launch a web browser. Click on the "SQL Developer Web button on the toolbar:



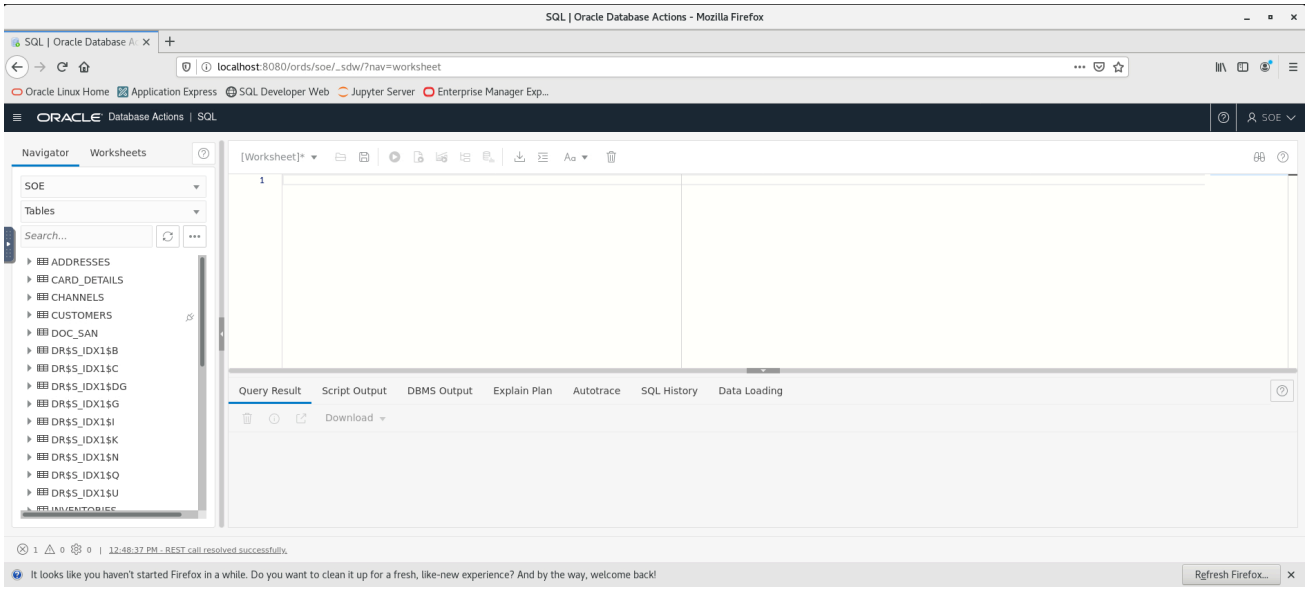
Use "SOE" as username, and click "Next".



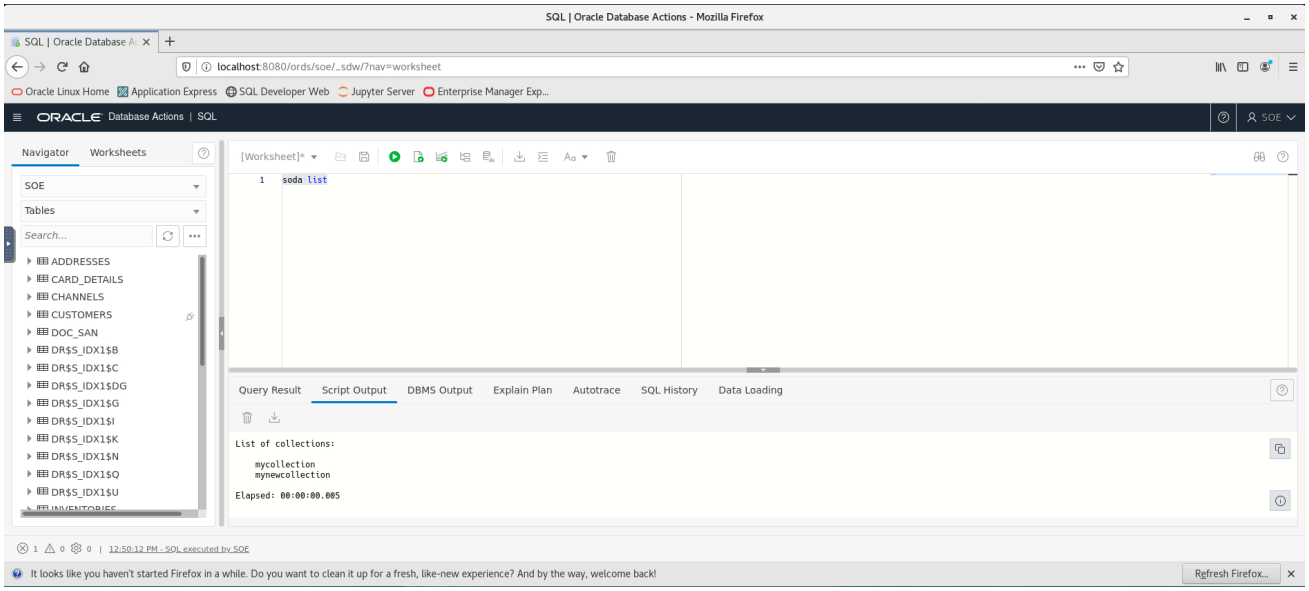
Use "soe" in lowercase for the password and click "Sign in".

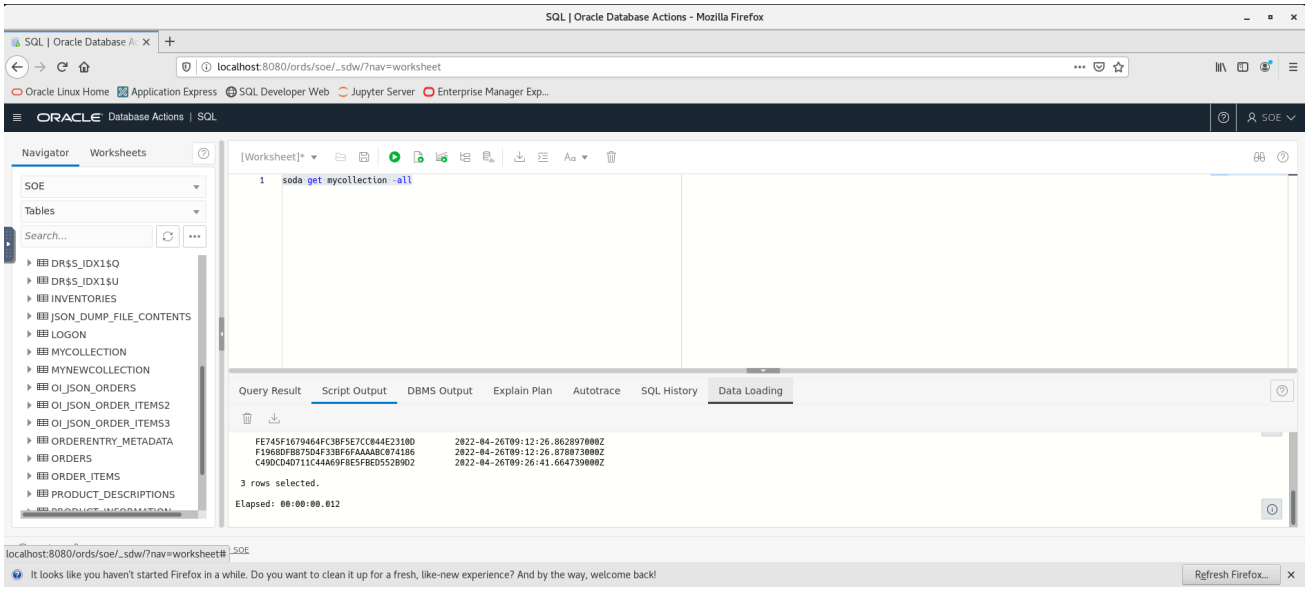


We will use the "SQL" and "JSON" modules, for now click on the SQL module: this will open a SQL worksheet.

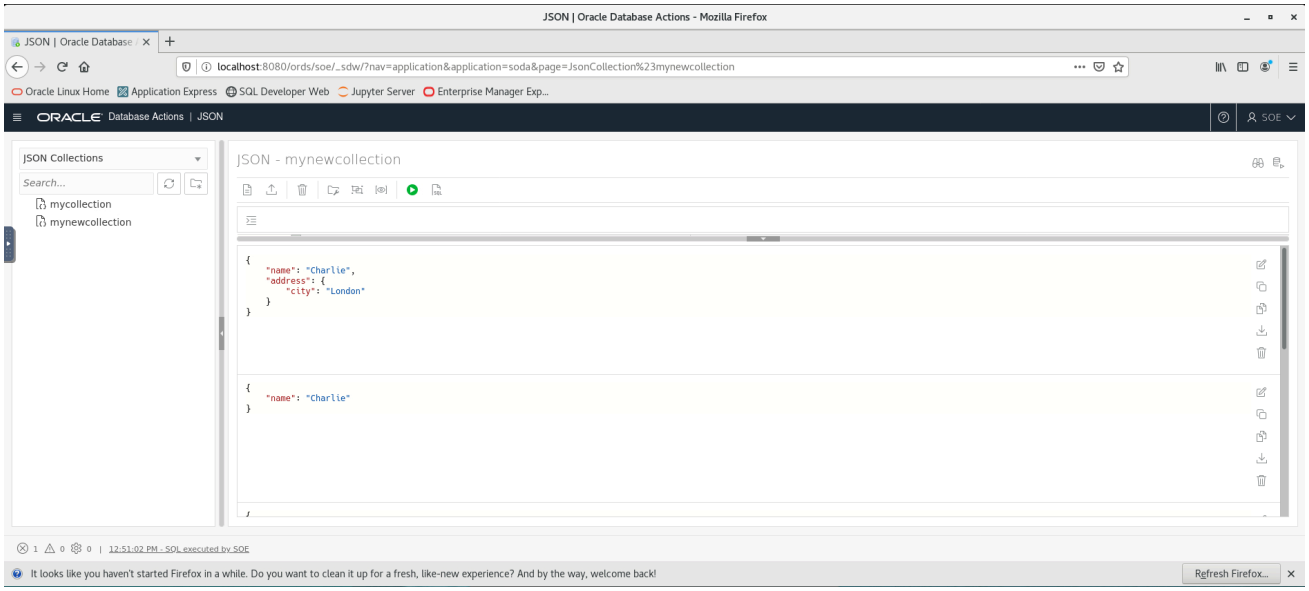


We can use that worksheet to execute SODA command, in the exact same way we used sqlcl:

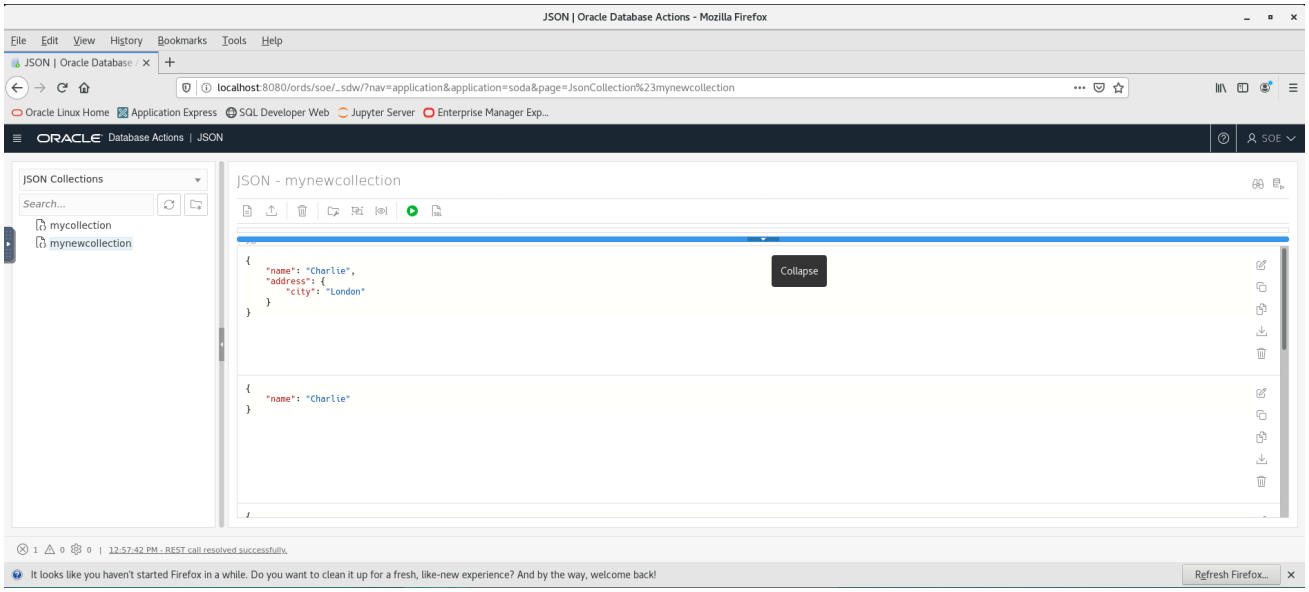




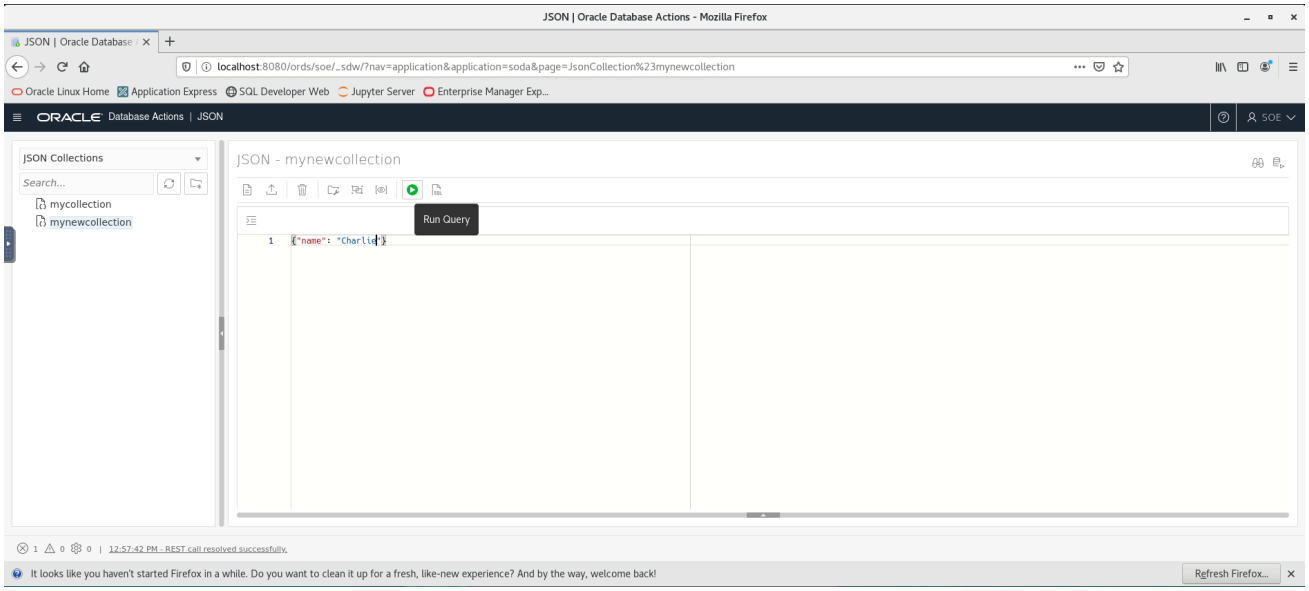
In the title bar, click on "Database Actions" to go back to the main page ("Launchpad"). Then choose JSON: you'll be redirected to the JSON console:



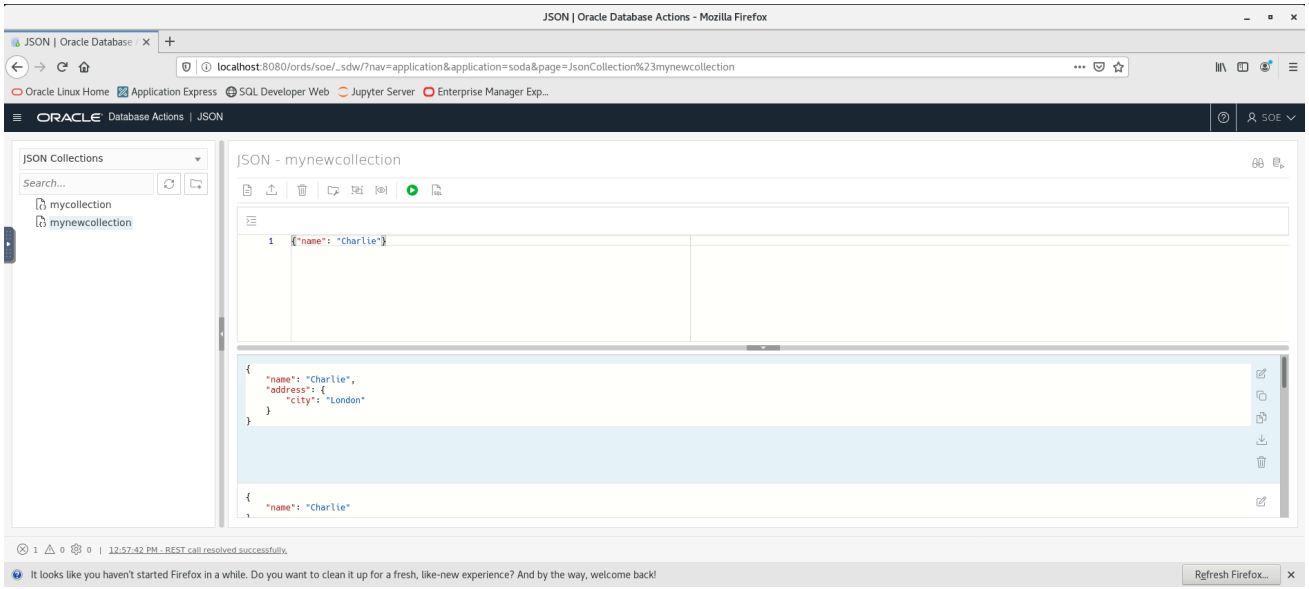
Here you can see your collections, and interact directly with them, without typing any command, using the intuitive GUI.
Let's for example run a "Query by Example": click the blue bar to expand the search window.



Then define a filter, using JSON format:



This filter will retrieve the document whose "name" field equals to "Charlie":



Go back to the SQL worksheet, and let Database Actions opened.

REST Endpoints

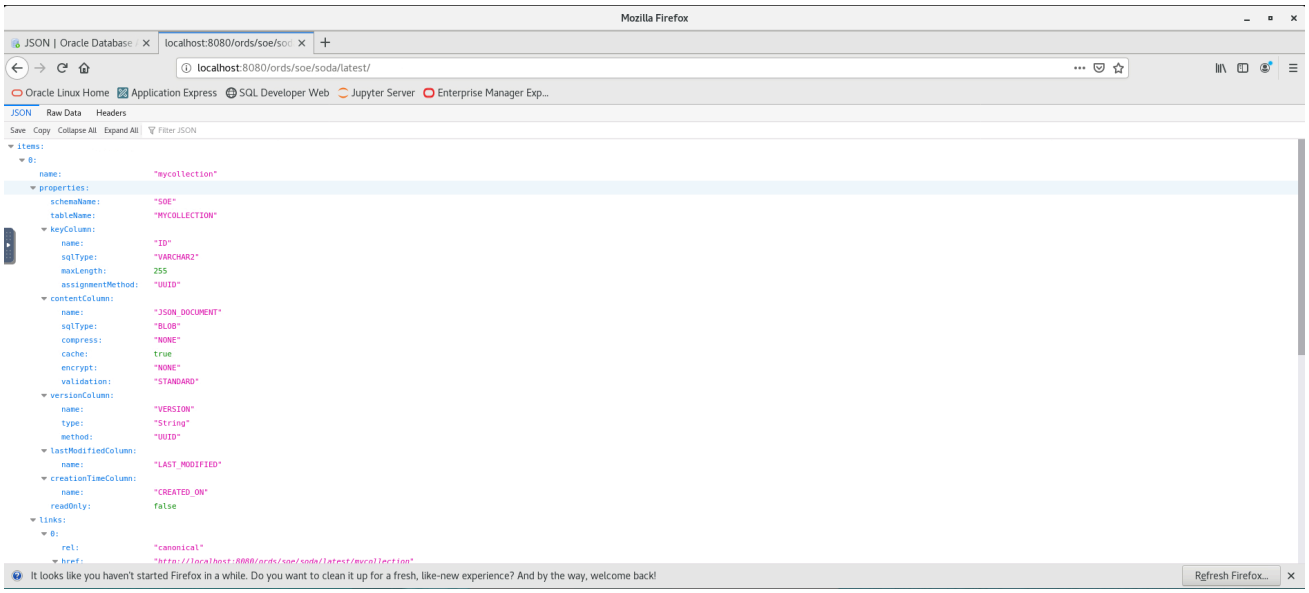
The collections created before can be accessed and managed through REST endpoints.
Go back to your VNC session, and use the following URL in your internet browser:
http://localhost:8080/ords/soe/soda/latest/

You can get the same result from the operating system, running cURL commands:

```
curl -X GET http://localhost:8080/ords/soe/soda/latest/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1108  100  1108    0     0  92333      0 --:--:-- --:--:-- --:--:-- 92333
```

```
{
  "items": [
    {
      "name": "mycollection",
      "properties": {
        "schemaName": "SOE",
        "tableName": "MYCOLLECTION",
        "keyColumn": {
          "name": "ID",
          "sqlType": "VARCHAR2",
          "maxLength": 255,
          "assignmentMethod": "UUID"
        },
[...]
```

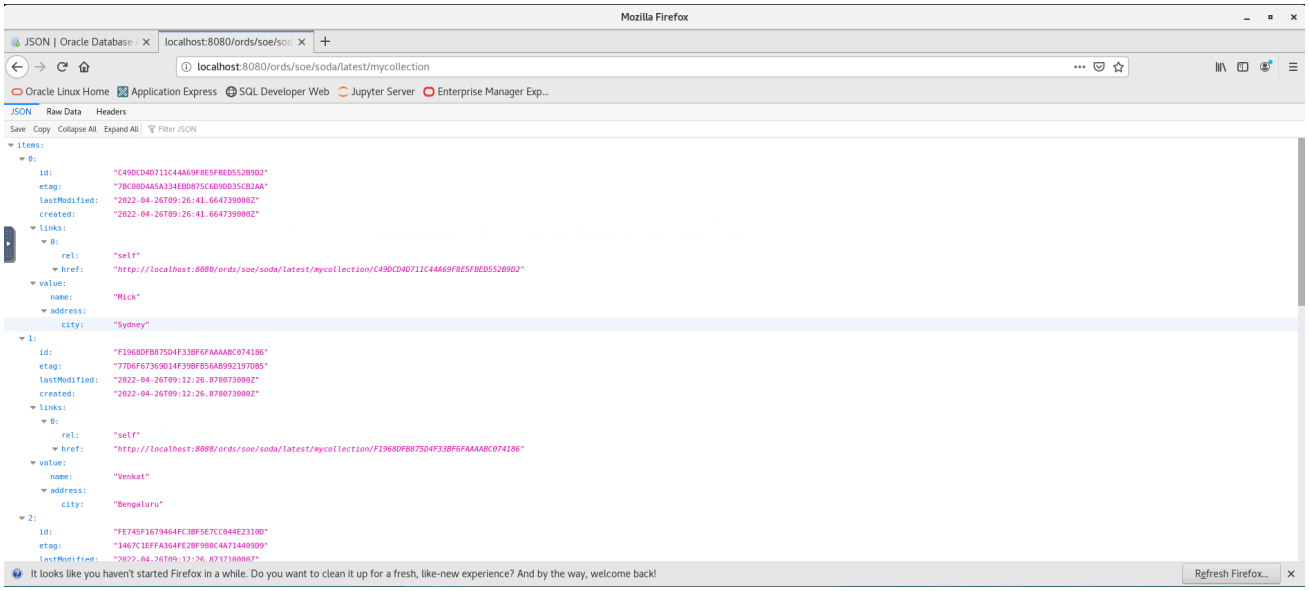

This will return the list of your collections:



If you need information about a particular collection, just complete the previous URL with the collection name, for example:

`http://localhost:8080/ords/soe/soda/latest/mycollection`

This will return the list of documents contained in that collection:



By using REST enpoints, we can create and manage a new collection:

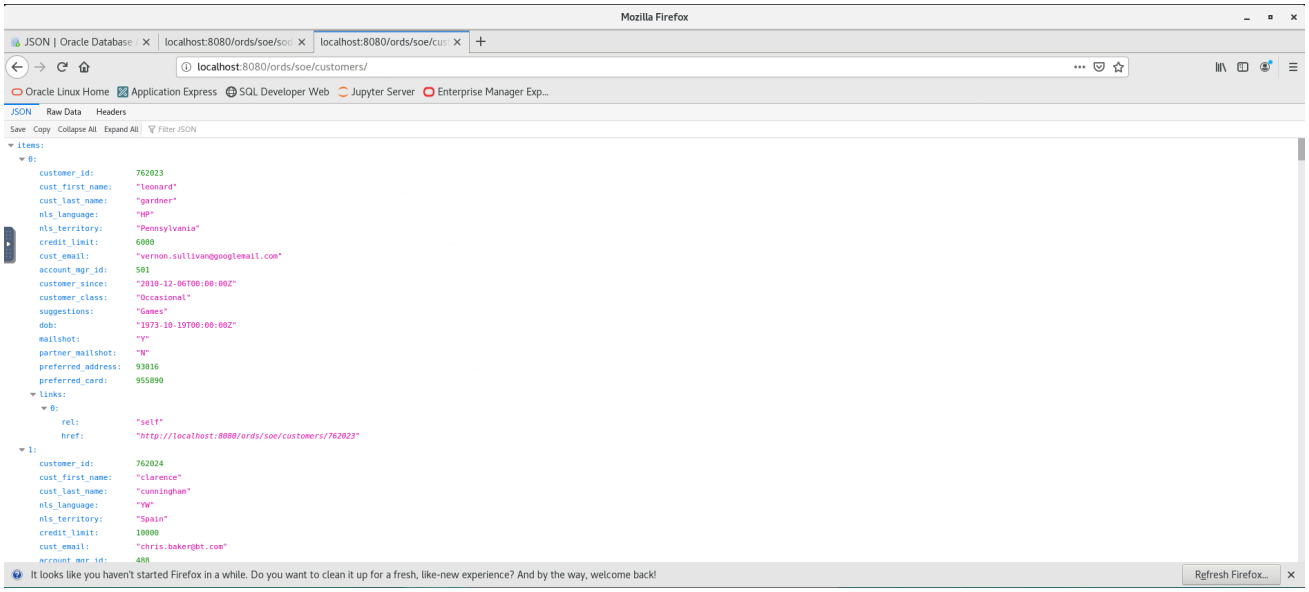
```
-- Use the PUT methdo to create a new collection

curl -X PUT http://localhost:8080/ords/soe/soda/latest/newcollection/

-- Use the POST method to insert a document in that collection

curl -X POST --data-binary '{"name": "Keith", "address": {"city": "London"}}' -H
"Content-Type: application/json"
http://localhost:8080/ords/soe/soda/latest/newcollection

{"items":[{"id":"FE91CC88F3A54F93B84235B9F88BD421","etag":"FA407EAAD2DF42FD97D4C95367E21
1BA","lastModified":"2022-04-26T14:44:24.793685000Z","created":"2022-04-
26T14:44:24.793685000Z"}],"hasMore":false,"count":1}

-- Retrieve the document from the collection with a GET method

curl -X GET http://localhost:8080/ords/soe/soda/latest/newcollection | jq

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1264    0  1264    0     0  48615      0 --:--:-- --:--:-- --:--:-- 48615
{
  "items": [
    {
      "id": "FE91CC88F3A54F93B84235B9F88BD421",
      "etag": "FA407EAAD2DF42FD97D4C95367E211BA",
      "lastModified": "2022-04-26T14:44:24.793685000Z",
      "created": "2022-04-26T14:44:24.793685000Z",
      "links": [
        {
          "rel": "self",
          "href":
"http://localhost:8080/ords/soe/soda/latest/newcollection/FE91CC88F3A54F93B84235B9F88BD4
21"
        }
      ],
```

```
      "value": {
        "name": "Keith",
        "address": {
          "city": "London"
        }
[...]
```

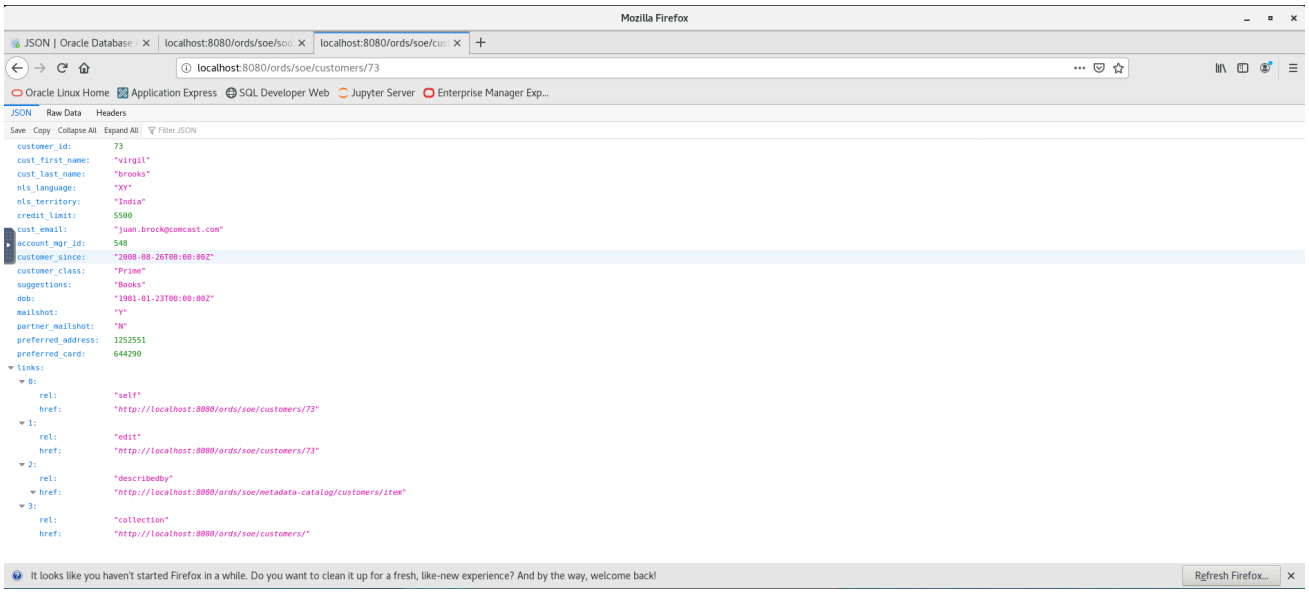
We can also publish relational tables for REST access. Let's publish the CUSTOMERS table for REST access:

```
cd /home/oracle
source .bashrc

sqlplus soe/soe@myoracledb:1521/orclpdb1

-- Publish the CUSTOMERS table for REST access !!!

DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

    ORDS.ENABLE_OBJECT(p_enabled => TRUE,
                       p_schema => 'SOE',
                       p_object => 'CUSTOMERS',
                       p_object_type => 'TABLE',
                       p_object_alias => 'customers',
                       p_auto_rest_auth => FALSE);

    commit;

END;
/
exit
```

In your VNC session, paste the following URL in your internet browser:
`http://localhost:8080/ords/soe/customers/`



Now use that one to select a particular row in the table:
http://localhost:8080/ords/soe/customers/73



SQL Access

We can use SQL to access and manipulate JSON documents. Now we are going to review some native SQL JSON API available in the Oracle Database.

```
cd /home/oracle
source .bashrc

sqlplus soe/soe@myoracledb:1521/orclpdb1

set timing on
-- Return relational data as JSON documents !!!

select json_object (
       'CUST_FIRST_NAME' value CUST_FIRST_NAME,
       'CUST_LAST_NAME' value CUST_LAST_NAME,
       'CUST_EMAIL' value CUST_EMAIL,
       'PREFERRED_CARD' value PREFERRED_CARD,
       'CREDIT_LIMIT' value CREDIT_LIMIT*1.21
       ) as mijson
from customers
where rownum < 6;

MIJSON
--------------------------------------------------------------------------------
{"CUST_FIRST_NAME":"leonard","CUST_LAST_NAME":"gardner","CUST_EMAIL":"vernon.sul
livan@googlemail.com","PREFERRED_CARD":955890,"CREDIT_LIMIT":7260}

{"CUST_FIRST_NAME":"clarence","CUST_LAST_NAME":"cunningham","CUST_EMAIL":"chris.
baker@bt.com","PREFERRED_CARD":632150,"CREDIT_LIMIT":12100}

{"CUST_FIRST_NAME":"gerald","CUST_LAST_NAME":"turner","CUST_EMAIL":"tom.bryant@v
irgin.com","PREFERRED_CARD":203989,"CREDIT_LIMIT":6655}
```

```
{"CUST_FIRST_NAME":"derrick","CUST_LAST_NAME":"woods","CUST_EMAIL":"timothy.brow
n@aol.com","PREFERRED_CARD":1233073,"CREDIT_LIMIT":6050}

MIJSON
--------------------------------------------------------------------------------

{"CUST_FIRST_NAME":"theodore","CUST_LAST_NAME":"white","CUST_EMAIL":"gilberto.le
e@verizon.com","PREFERRED_CARD":1208490,"CREDIT_LIMIT":7260}


Elapsed: 00:00:00.01

-- Return relational data as a JSON list !!!

select JSON_OBJECTAGG (
       KEY to_char(c.CHANNEL_ID) value c.CHANNEL_CLASS || '-' || c.CHANNEL_DESC
       ) as canales
from channels c
order by c.CHANNEL_DESC;  2    3    4    5

CANALES
--------------------------------------------------------------------------------
{"3":"Direct-Direct Sales","9":"Direct-Tele Sales","5":"Indirect-Catalog","4":"I
ndirect-Internet","2":"Others-Partners"}

-- Return relational data as a JSON array !!!

select JSON_ARRAY (
       rownum,
       JSON_OBJECT (KEY 'Descripcion' value  CHANNEL_DESC),
       JSON_OBJECT (KEY 'ID canal' value  CHANNEL_ID)
       ) as canales_array
from channels c;  2    3    4    5    6

CANALES_ARRAY
--------------------------------------------------------------------------------
[1,{"Descripcion":"Direct Sales"},{"ID canal":3}]
[2,{"Descripcion":"Tele Sales"},{"ID canal":9}]
[3,{"Descripcion":"Catalog"},{"ID canal":5}]
[4,{"Descripcion":"Internet"},{"ID canal":4}]
[5,{"Descripcion":"Partners"},{"ID canal":2}]

Elapsed: 00:00:00.01
```

We can perform cross-model queries, mixing relational and JSON data in the same query, for example:

```
SQL> desc OI_JSON_ORDERS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(12)
 O_JSON                                             VARCHAR2(4000)

SQL> desc WAREHOUSES
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 WAREHOUSE_ID                                       NUMBER(6)
```

```
 WAREHOUSE_NAME                                VARCHAR2(35)
 LOCATION_ID                                   NUMBER(4)

SQL> select W.WAREHOUSE_NAME, sum(to_number(json_value (OI.O_JSON, '$.ORDER_TOTAL'))) as
TOTAL
from    OI_JSON_ORDERS OI,
        WAREHOUSES W
where   W.WAREHOUSE_ID = json_value (OI.O_JSON, '$.WAREHOUSE_ID')
and     W.warehouse_name in
('McsRxsWjRxXMFDcobjhEIDdEsO','5eH6XK38SRmNEZCUg43EDIjDICDhbV','PLlypy')
group by W.WAREHOUSE_NAME
order by 1;

WAREHOUSE_NAME                             TOTAL
---------------------------------- ----------
5eH6XK38SRmNEZCUg43EDIjDICDhbV          7190505
McsRxsWjRxXMFDcobjhEIDdEsO       7368607
PLlypy                           7197962

Elapsed: 00:00:05.32
```

We joined the two tables on "W.warehouse_id", which is a column of the relational table, with " json_value (OI.O_JSON, '$.WAREHOUSE_ID')", which is a field of the JSON document. Then the result is an aggregation on " json_value (OI.O_JSON, '$.ORDER_TOTAL')", which is a field of the JSON document, grouped by " W.WAREHOUSE_NAME", which is a column of the relational table. Let's have a look behind the scene, and get the execution plan of this query:

```
set lines 120
set autotrace traceonly explain

select W.WAREHOUSE_NAME, sum(to_number(json_value (OI.O_JSON, '$.ORDER_TOTAL'))) as
TOTAL
from    OI_JSON_ORDERS OI,
        WAREHOUSES W
where   W.WAREHOUSE_ID = json_value (OI.O_JSON, '$.WAREHOUSE_ID')
and     W.warehouse_name in
('McsRxsWjRxXMFDcobjhEIDdEsO','5eH6XK38SRmNEZCUg43EDIjDICDhbV','PLlypy')
group by W.WAREHOUSE_NAME
order by 1;

Execution Plan
----------------------------------------------------------
Plan hash value: 2556601651

-------------------------------------------------------------------------------------------
---
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
---
|   0 | SELECT STATEMENT    |              |    3 |  1266 |    38M  (1)| 00:25:23 |
|   1 |  SORT GROUP BY            |              |    3 |  1266 |    38M  (1)| 00:25:23 |
|*  2 |   HASH JOIN         |              |    35M|    13G|    38M  (1)| 00:25:23 |
|*  3 |    TABLE ACCESS FULL      | WAREHOUSES   |    3 |    72 |  4   (0)| 00:00:01 |
|   4 |    NESTED LOOPS     |              |    11G|  4328G|    38M  (1)| 00:25:20 |
|   5 |     TABLE ACCESS FULL     | OI_JSON_ORDERS |  1429K|   537M| 21681   (1)|
00:00:01 |
```

```
|   6 |     JSONTABLE EVALUATION |             |     |     |          |      |
-------------------------------------------------------------------------------------------
---

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("W"."WAREHOUSE_ID"=TO_NUMBER("P"."C_01$"))
   3 - filter("W"."WAREHOUSE_NAME"='5eH6XK38SRmNEZCUg43EDIjDICDhbV' OR
              "W"."WAREHOUSE_NAME"='McsRxsWjRxXMFDcobjhEIDdEsO' OR
"W"."WAREHOUSE_NAME"='PLlypy')
```

SEARCH Indexes

JSON documents might be indexed for performance: we might build indexes on particular fields of the JSON documents, to boost queries that are known to access the data with predicates on that particular fields.
We might also create a SEARCH index on a JSON table, to leverage TEXT like searches. In the following steps, we are going to create a new collection, insert a few documents, create a search index, and run some queries.

Connect to sqlcl from the operating system, and run the following commands:

```
unset ORACLE_HOME
cd /home/oracle/sqlcl/bin

./sql soe/soe@myoracledb:1521/orclpdb1

SQL> soda create musiccollection

 Successfully created collection: musiccollection

SQL> soda insert musiccollection {"name": "The Rolling Stones","Title": "Bridges of
Babylon", "Description": "A Great album by the greatest band ever"}

 JSON document inserted successfully.

SQL> soda insert musiccollection {"name": "The Rolling Stones","Title": "Jump Back",
"Description": "An awesome compilation by the greatest band ever"}

 JSON document inserted successfully.

SQL> soda insert musiccollection {"name": "Pink Floyd","Title": "Wish you were here",
"Description": "A pretty good choice"}

 JSON document inserted successfully.

SQL> soda insert musiccollection {"name": "Pink Floyd","Title": "Greatest hits",
"Description": "A pretty good choice"}

 JSON document inserted successfully.

SQL> soda insert musiccollection {"name": "Police","Title": "Every breath you take",
"Description": "Breathtaking, their greatest album"}
```

```
 JSON document inserted successfully.

-- We can easily insert from a file as well:

! echo '{"name": "Eric Clapton","Title": "Unplugged", "Description": "Awesome unplugged
concert"}' > /home/oracle/tt.json
!cat /home/oracle/tt.json
{"name": "Eric Clapton","Title": "Unplugged", "Description": "Awesome unplugged
concert"}

soda insert musiccollection /home/oracle/tt.json

 JSON document inserted successfully.


 Successfully inserted file: /home/oracle/tt.json

-- Now we create an SEARCH index on the underlying table !!!

SQL> info musiccollection
TABLE: MUSICCOLLECTION
        LAST ANALYZED:
        ROWS          :
        SAMPLE SIZE   :
        INMEMORY      :DISABLED
        COMMENTS      :

Columns
NAME             DATA TYPE            NULL  DEFAULT    COMMENTS
*ID              VARCHAR2(255 BYTE)   No
 CREATED_ON      TIMESTAMP(6)         No    sys_extract_utc(SYSTIMESTAMP)
 LAST_MODIFIED   TIMESTAMP(6)         No    sys_extract_utc(SYSTIMESTAMP)
 VERSION         VARCHAR2(255 BYTE)   No
 JSON_DOCUMENT   JSON                 Yes

Indexes
SOE.SYS_C0013436    UNIQUE        VALID                        ID

SQL> CREATE SEARCH INDEX music_search_idx ON musiccollection (JSON_DOCUMENT) FOR JSON;

INDEX MUSIC_SEARCH_IDX created.
```

Now we are able to perform TEXT like queries on top of the collection: from sqlcl, run the following commands:

```
INDEX MUSIC_SEARCH_IDX created.

-- Let's run a search query !!!

SQL> SELECT m.json_document.name as "Band name", m.json_document."Title" as "Title",
m.json_document."Description" as "Description"
  FROM   musiccollection m
  WHERE  JSON_TEXTCONTAINS(json_document, '$.Description', 'greatest band');

              Band name                    Title
Description
```

```
_____________________ _____________________
_____________________________________________
"The Rolling Stones"    "Bridges of Babylon"    "A Great album by the greatest band
ever"
"The Rolling Stones"    "Jump Back"             "An awesome compilation by the greatest
band ever"
```

JSON and security

All the security features of the Oracle Database apply on JSON documents. Let's take an example of Data Redaction.
Still in sqlcl, run the following commands against the OI_JSON_ORDERS table. This table has been pre-built, and contains ORDERS in JSON format:

```
SQL> desc OI_JSON_ORDERS

     Name        Null?              Type
_________ ____________ ___________________
ID         NOT NULL     NUMBER(12)
O_JSON                  VARCHAR2(4000)

SQL> SELECT JSON_QUERY(O_JSON,'$' WITH WRAPPER) from OI_JSON_ORDERS where ID = 12345;


JSON_QUERY(O_JSON,'$'WITHWRAPPER)
_____________________________________________________________________________________
_____________________________________________________________________________________
_____________________________________________________________________________________
_____________________________________________________________________________________
_____________________________________
[{"ORDER_ID":12345,"ORDER_DATE":"2011-05-
31T02:00:00.000000Z","ORDER_MODE":"online","CUSTOMER_ID":808932,"ORDER_STATUS":4,"ORDER_
TOTAL":3989,"SALES_REP_ID":534,"PROMOTION_ID":499,"WAREHOUSE_ID":318,"DELIVERY_TYPE":"Ne
xt_Day","COST_OF_DELIVERY":2,"WAIT_TILL_ALL_AVAILABLE":"ship_when_ready","DELIVERY_ADDRE
SS_ID":4,"CUSTOMER_CLASS":"Occasional","INVOICE_ADDRESS_ID":2,"CARD_NUMBER":"00000000000
0"}]
```

The JSON payload contains information about the credit card number. This information is returned "as is", and we would like to redact it for USER2, but not for USER1.
This is where Data Redaction (part of the Advanced Security Option) comes in action.

First we need to create a view on top of the JSON table, as Data Redaction cannot be applied directly on the JSON column:

```
create or replace view V_ORDERS
as
SELECT       ID as ORDER_ID,
               json_value(O_JSON, '$.ORDER_DATE' returning DATE) as ORDER_DATE,
               json_value(O_JSON, '$.ORDER_MODE' returning VARCHAR2(8)) as ORDER_MODE,
               json_value(O_JSON, '$.CUSTOMER_ID' returning NUMBER(12)) as CUSTOMER_ID,
               json_value(O_JSON, '$.ORDER_STATUS' returning NUMBER(2)) as ORDER_STATUS,
               json_value(O_JSON, '$.ORDER_TOTAL' returning NUMBER(8,2)) as ORDER_TOTAL,
```

```
               json_value(O_JSON, '$.SALES_REP_ID' returning NUMBER(6)) as SALES_REP_ID,
               json_value(O_JSON, '$.PROMOTION_ID' returning NUMBER(6)) as PROMOTION_ID,
               json_value(O_JSON, '$.WAREHOUSE_ID' returning NUMBER(6)) as WAREHOUSE_ID,
               json_value(O_JSON, '$.DELIVERY_TYPE' returning VARCHAR2(15)) as
DELIVERY_TYPE,
               json_value(O_JSON, '$.COST_OF_DELIVERY' returning NUMBER(6)) as
COST_OF_DELIVERY,
            json_value(O_JSON, '$.WAIT_TILL_ALL_AVAILABLE' returning VARCHAR2(15)) as
WAIT_TILL_ALL_AVAILABLE,
            json_value(O_JSON, '$.DELIVERY_ADDRESS_ID' returning NUMBER(12)) as
DELIVERY_ADDRESS_ID,
            json_value(O_JSON, '$.CUSTOMER_CLASS' returning VARCHAR2(30)) as
CUSTOMER_CLASS,
            json_value(O_JSON, '$.CARD_NUMBER' returning VARCHAR2(12)) as CARD_NUMBER,
            json_value(O_JSON, '$.INVOICE_ADDRESS_ID' returning NUMBER(12)) as
INVOICE_ADDRESS_ID
from SOE.OI_JSON_ORDERS;

exit
```

With sqlcl, reconnect to the database as "system" and create a Data Redaction policy on top of the view V_ORDERS:

```
./sql system/"Oracle_4U"@myoracledb:1521/orclpdb1

BEGIN
    SYS.DBMS_REDACT.DROP_POLICY (
        object_schema=> 'SOE',
        object_name => 'V_ORDERS',
        policy_name => 'POL_HIDE_ORDER_TOTAL');
END;
/

BEGIN
SYS.DBMS_REDACT.ADD_POLICY(
object_schema=> 'SOE',
object_name => 'V_ORDERS',
column_name => 'CARD_NUMBER',
column_description => 'Card Number',
policy_name => 'POL_HIDE_ORDER_TOTAL',
policy_description => 'Hide card number',
function_type => DBMS_REDACT.PARTIAL,
function_parameters => 'VVVVVVVVVVVV,VVVVVVVVVVVV,*,1,5',
expression => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') = ''USER2''');
end;
/
exit
```

This policy will redact partially the CARD_NUMBER column of the SOE.V_ORDERS view, by substituting the first 5 digits of the card number by "*". This policy will only apply to USER2.
Now connect as USER2 and query the data:

```
./sql user2/"Oracle_4U"@myoracledb:1521/orclpdb1

select *
```

```
from soe.v_orders
where order_id = 12345;

   ORDER_ID    ORDER_DATE    ORDER_MODE    CUSTOMER_ID    ORDER_STATUS    ORDER_TOTAL
SALES_REP_ID    PROMOTION_ID    WAREHOUSE_ID    DELIVERY_TYPE    COST_OF_DELIVERY
WAIT_TILL_ALL_AVAILABLE    DELIVERY_ADDRESS_ID    CUSTOMER_CLASS    CARD_NUMBER
INVOICE_ADDRESS_ID
_____________ _____________ _____________ _____________ _____________ _____________
_____________ _____________ _____________ _____________ _____________
_____________________________ _____________________ _____________ _____________
_____________________
      12345 31-MAY-11     online                808932               4           3989
534             499             318 Next_Day                           2 ship_when_ready
4 Occasional        *****0000000                      2

exit
```

We observe that the card number value has been redacted. Now let's connect as USER1 and run the same query:

```
./sql user1/"Oracle_4U"@myoracledb:1521/orclpdb1

select *
from soe.v_orders
where order_id = 12345;

   ORDER_ID    ORDER_DATE    ORDER_MODE    CUSTOMER_ID    ORDER_STATUS    ORDER_TOTAL
SALES_REP_ID    PROMOTION_ID    WAREHOUSE_ID    DELIVERY_TYPE    COST_OF_DELIVERY
WAIT_TILL_ALL_AVAILABLE    DELIVERY_ADDRESS_ID    CUSTOMER_CLASS    CARD_NUMBER
INVOICE_ADDRESS_ID
_____________ _____________ _____________ _____________ _____________ _____________
_____________ _____________ _____________ _____________ _____________
_____________________________ _____________________ _____________ _____________
_____________________
      12345 31-MAY-11     online                808932               4           3989
534             499             318 Next_Day                           2 ship_when_ready
4 Occasional        000000000000                      2

exit
```

We observe that the card number value hasn't been redacted for USER1.

Data Redaction policies will apply regardless of the way data is being accessed. Let's access data in the v_orders view with a REST endpoint.
First we need to publish the view for REST access:

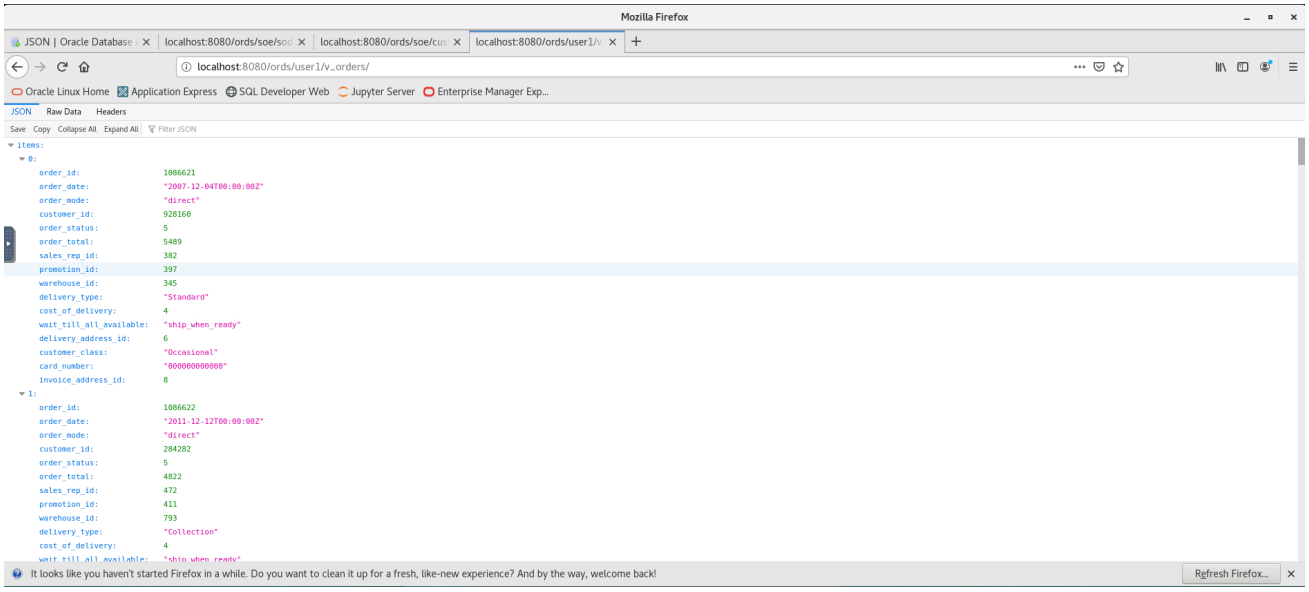
```
cd /home/oracle
source .bashrc
sqlplus system/"Oracle_4U"@myoracledb:1521/orclpdb1

BEGIN
    ords_admin.enable_schema (
        p_enabled               => TRUE,
        p_schema                => 'USER1',
        p_url_mapping_type      => 'BASE_PATH',
```

```
        p_url_mapping_pattern   => 'user1',
        p_auto_rest_auth        => TRUE   -- this flag says, don't expose my REST APIs
    );
    COMMIT;
END;
/

BEGIN
    ords_admin.enable_schema (
        p_enabled               => TRUE,
        p_schema                => 'USER2',
        p_url_mapping_type      => 'BASE_PATH',
        p_url_mapping_pattern   => 'user2',
        p_auto_rest_auth        => TRUE   -- this flag says, don't expose my REST APIs
    );
    COMMIT;
END;
/
exit

-- Reconnect as USER1 and publish the V_ORDERS view:

sqlplus USER1/"Oracle_4U"@myoracledb:1521/orclpdb1

create or replace view V_ORDERS as select * from soe.v_orders;


DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

    ORDS.ENABLE_OBJECT(p_enabled => TRUE,
                       p_schema => 'USER1',
                       p_object => 'V_ORDERS',
                       p_object_type => 'VIEW',
                       p_object_alias => 'v_orders',
                       p_auto_rest_auth => FALSE);

    commit;

END;
/
exit

-- Do the same with USER2:

sqlplus USER2/"Oracle_4U"@myoracledb:1521/orclpdb1

create or replace view V_ORDERS as select * from soe.v_orders;


DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN

    ORDS.ENABLE_OBJECT(p_enabled => TRUE,
                       p_schema => 'USER2',
                       p_object => 'V_ORDERS',
```

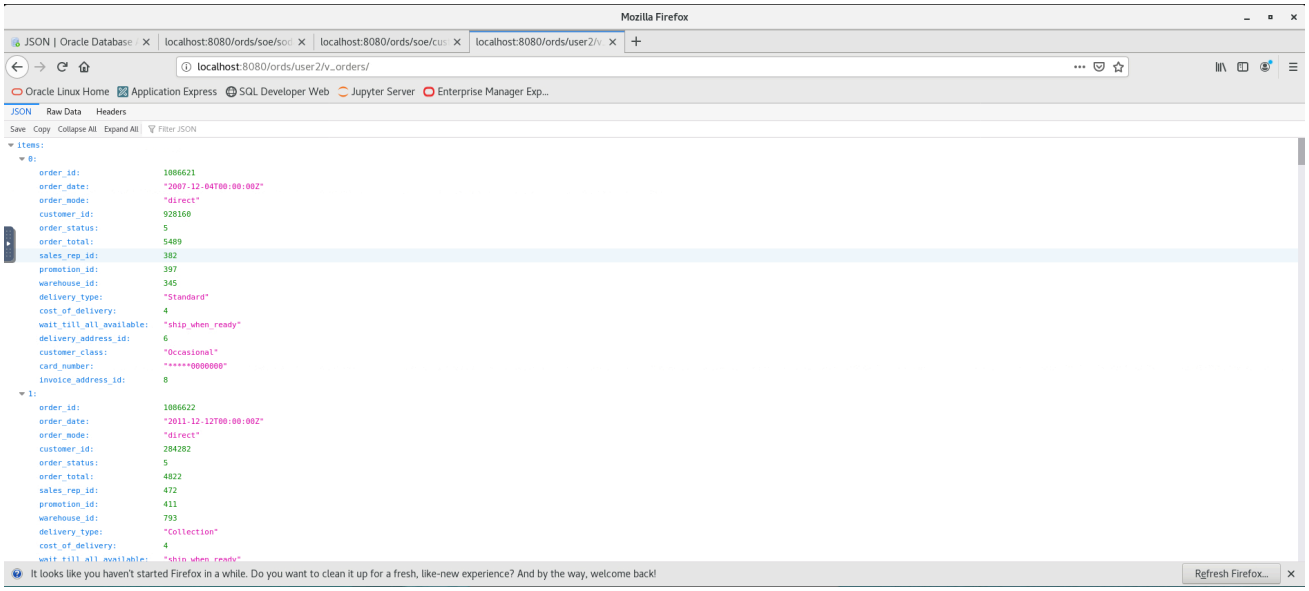
```
                    p_object_type => 'VIEW',
                    p_object_alias => 'v_orders',
                    p_auto_rest_auth => FALSE);

    commit;

END;
/

exit
```

Now test your endpoints, depending on the user:
For USER1: http://localhost:8080/ords/user1/v_orders/
For USER2: http://localhost:8080/ords/user2/v_orders/

For USER1, the card number is not redacted:



For USER2, the card number is redacted:



This concludes this demo.