
CERN

Oracle REST Data Services

REST

- It is an architectural style for applications, based on HTTP protocol
- Model resource, not action
  - GET http://ordstest.cern.ch/user/damian
  - POST http://ordstest.cern.ch/user/hubert
  - ~~POST http://ordstest.cern.ch/user/adduser~~

REST

- Uses HTTP verbs

HTTP VERB	ACTION
GET	Read
PUT	Update
POST	Create
DELETE	Delete

- Success and error messages through HTTP Response codes (ex. 200 OK, 401 Unauthorized)

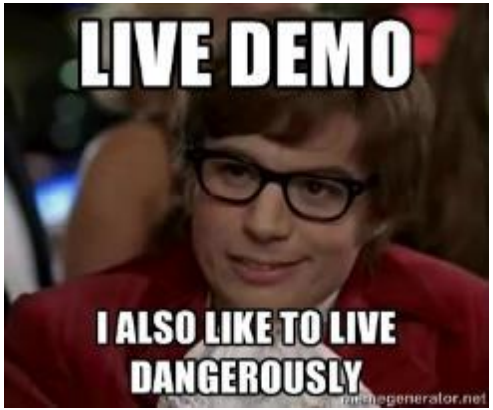
ORDS - a mid-tier Java application, maps HTTP(S) verbs (GET, POST, PUT, DELETE, etc.) to database transactions and returns results formatted using JSON.

I have an Oracle database at CERN,
how do I use the RESTful capabilities?

Demo

A walk through different use cases:

- GET some data with AutoREST, or specific records with where clause
- Add database records
- Useful queries



REST-enable a schema

```
BEGIN
ORDS.ENABLE_SCHEMA(
p_enabled => TRUE,
p_schema => 'DMOSKALI_ORDS_REST',
p_url_mapping_type => 'BASE_PATH',
p_url_mapping_pattern => 'myschema',
p_auto_rest_auth => FALSE);
commit;
END;
```

ENABLE_SCHEMA enables Oracle REST Data Services to access the named schema.

https://oraweb.cern.ch/ords/devdb11/**myschema**

AutoREST

„Autoenablement of RESTful Services makes it very easy to set up some basic Oracle microservices that use simple queries (e.g., SELECT * and primary key lookup), insert, update, and delete operations. One quick workflow can auto enable one or multiple tables in a schema or even all the tables in a schema.” Oracle Blog.

Enable AutoREST

```
BEGIN
  ORDS.ENABLE_OBJECT(
p_enabled => TRUE,
p_schema => 'DMOSKALI_ORDS_REST',
p_object => 'USERS',
p_object_type => 'TABLE',
p_object_alias => 'users',
p_auto_rest_auth => FALSE);

commit;
END;
```

p_object_alias will map to the URL:
https://oraweb.cern.ch/ords/devdb11/**myschema/users**

Our service should be accessible now, it's that easy!

AutoREST GET

```
curl https://oraweb.cern.ch/ords/devdb11/myschema/users/
```

```
curl https://oraweb.cern.ch/ords/devdb11/myschema/users/1
```

```
curl https://oraweb.cern.ch/ords/devdb11/myschema/users/?q={"NAME":"Smith"}
```

AutoREST POST

Insert a new record:

POST
https://oraweb.cern.ch/ords/devdb11/myschema/users/

Content-Type: application/json

```
{"USERID":6, "NAME":"Marek", "SURNAME":"Nowak"}
```

Delete, update and other examples in an oracle-base tutorial:

https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-autorest#enable-ords-and-autorest

GET with a custom module

Allows to define custom SQL queries and map them to URLs.

First, let's define a module. Module is an identifier for a set of endpoints – not a programing unit.

```
BEGIN
ORDS.DEFINE_MODULE(
                p_module_name => 'demo_module',
                p_base_path => 'demo/');
COMMIT;
END;
```

https://oraweb.cern.ch/ords/devdb11/**myschema**/**demo/**

GET with a custom module

Next define a template. The relation of modules and templates to the URLs is the following:


```
BEGIN
ORDS.define_template(
            p_module_name => 'demo_module',
            p_pattern => 'hello/:name');
COMMIT;
END;
```

GET with a custom module

The last step is to define handlers for our template. This is what links templates to HTTP verbs In this example we define a GET handler.

```
BEGIN ORDS.define_handler(
                p_module_name => 'demo_module',
                p_pattern => 'hello/:name',
                p_method => 'GET',
                p_source_type => ORDS.source_type_query,
                p_source => 'select "Hello " || :name || " from " || nvl(:who,sys_context("USERENV","CURRENT_USER")) "greeting" from dual',
                p_items_per_page => 0);
COMMIT;
END;
```

```
curl https://oraweb.cern.ch/ords/devdb11/myschema/demo/hello/damian
```

```
{"items":[{"greeting":"Hello damian from DMOSKALI_ORDS_REST"}]}
```

Module,Template,Handler

To wrap up:

- **Module** : A container for one or more templates, with an associated path (testmodule1/).
- **Template** : A container for one or more handlers. The template must be unique within the module and is associated with a specific path (emp/), which may or may not include parameters. You can think of it as of a resource.
- **Handler** : A link to the actual work that is done. Typical handler methods include GET, POST, PUT, DELETE. You can create a single handler per http verb per template

REST run procedure

We can map URL requests to procedure invocations. It could be useful when deleting records with foreign keys.

The following procedure removes a user given a surname:

```
CREATE OR REPLACE PROCEDURE remove_user (
            p_surname IN users.surname%TYPE )
AS BEGIN
            DELETE FROM users WHERE surname = p_surname;
END;
```

REST run procedure

Create a template and a handler, we reuse the demo_module.

```
BEGIN

ORDS.define_template(
p_module_name => 'demo_module',
p_pattern => 'user/:surname');

ORDS.define_handler(
p_module_name => 'demo_module',
p_pattern => 'user/:surname',
p_method => 'DELETE',
p_source_type => ORDS.source_type_plsql,
p_source => 'BEGIN remove_user(p_surname => :surname); END;',
p_items_per_page => 0);

COMMIT;
END;
```

REST run procedure

Now we should be able to delete a user:

```
DELETE
https://oraweb.cern.ch/ords/devdb11/myschema/demo/user/
```

```
{"surname": "Nowak"}
```

Tip: metadata catalog

See defined endpoints

https://oraweb.cern.ch/ords/devdb11/myschema/metadata-catalog/

Tip: useful views

- Show defined modules

```
COLUMN name FORMAT A20
COLUMN uri_prefix FORMAT A20

SELECT id, name, uri_prefix FROM user_ords_modules ORDER BY name;
```

- Show defined templates

```
COLUMN uri_template FORMAT A20

SELECT id, module_id, uri_template FROM user_ords_templates ORDER BY module_id;
```

Tip: useful views

- Show defined handlers

```
COLUMN source_type FORMAT A15
COLUMN source FORMAT A20

SELECT id, template_id, source_type, method, source FROM user_ords_handlers ORDER BY id;
```

Authentication: Basic

- We need
  - Role
  - Priviledge
  - Mapping URL<--> Priviledge
- ORDS roles map to CERN e-group.
- Roles and e-groups must follow the pattern **ords-rest-access-{NAME}** *(due to our internal Weblogic configuration)*

Basic auth

- Create a role, this role maps to the e-group **ords-rest-access-ims**

```
BEGIN
 ORDS.create_role(
    p_role_name => 'ords-rest-access-ims'
  );
COMMIT;
END;
```

Basic auth

- Define a privilege, that contains the previously created role. A role can have multiple privileges.

```
DECLARE
  l_arr OWA.vc_arr;
BEGIN
  l_arr(1) := 'ords-rest-access-ims';

  ORDS.define_privilege (
    p_privilege_name => 'rest-access-test',
    p_roles          => l_arr,
    p_label          => 'Users data',
    p_description    => 'Allow access to Users data.'
  );
  COMMIT;
END;
```

Basic auth

- Define a mapping, here we say that all URL Patterns ‘/users/*’ are protected.

```
BEGIN
  ORDS.create_privilege_mapping(
    p_privilege_name => 'rest-access-test',
    p_pattern => '/users/*'
  );
  COMMIT;
END;
```

```
curl https://oraweb.cern.ch/ords/devdb11/myschema/users/
```

- Now we should get 401 Unathorized.

ORDS at CERN

- Mobility.ch comes 2 times per day to fetch the list of people who can drive one of their vehicles at CERN (those with V permit)
- EDH consumes a web service at https://oraweb.cern.ch/ords/edmsdb/adams3/api/edh/ to approve/reject our access requests (they are created from ADaMS in EDH) - average 250 calls per day

https://twiki.cern.ch/twiki/bin/view/DB/DevelopingOracleRestfulServices

Official docker image:

https://github.com/oracle/docker-images/tree/master/OracleRestDataServices

Questions?