DOAG

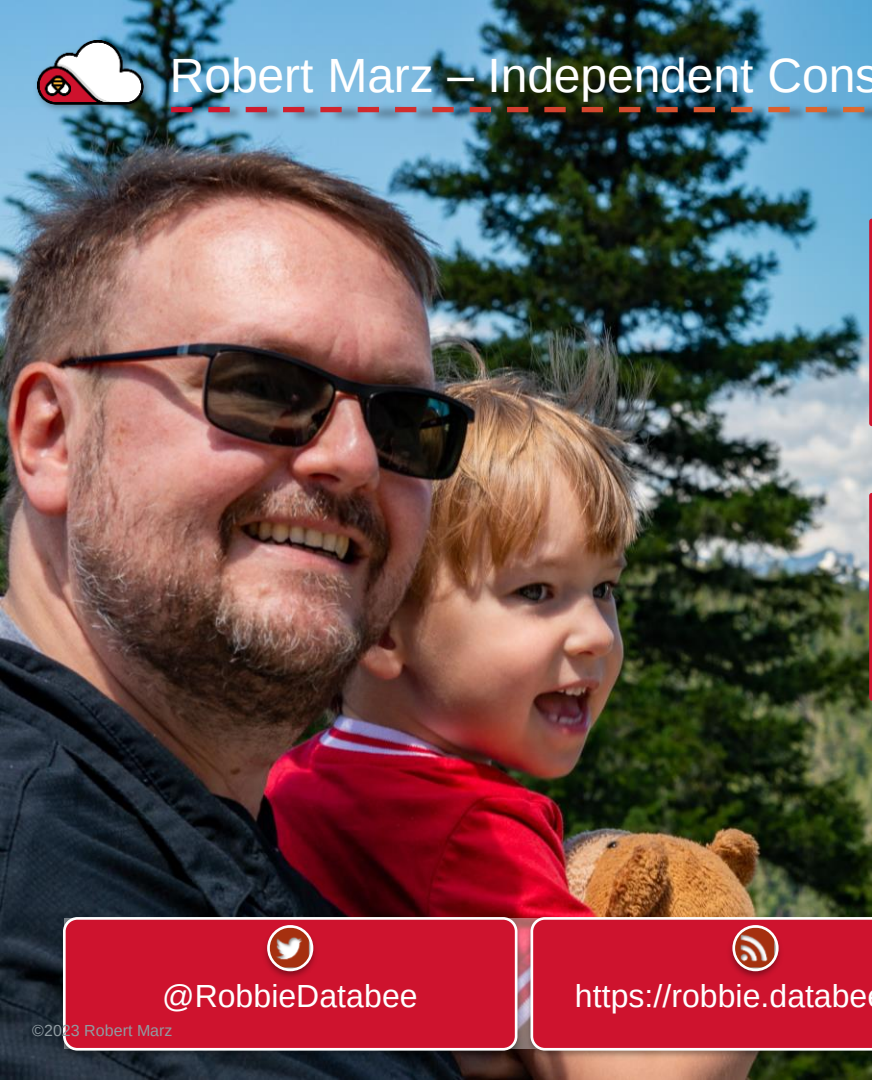
datenbank.doag.org

Einsteiger: Anwendungsentwicklung mit REST und JSON

Donnerstag 25.05.2023 | 11:15 - 12:00 | Amsterdam 1

Robert Marz
DATABEE
Die IT-Architekten

Robert Marz – Independent Consultant



Primary Role

Senior Technical Architect
with database centric view of the world

DOAG (German Oracle User Group)

Active Member of Database Community
Responsible for Cloud Topics

Databees.

DATABEE
Die IT-Architekten

Oracle ACE Pro

@RobbieDatabee

https://robbie.databee.org

robert.marz@databee.org

Oracle ACE Program

500+ technical experts helping peers globally

The **Oracle ACE Program** recognizes and rewards community members for their technical and community contributions to the Oracle community

3 membership tiers

Oracle ACE Director | Oracle ACE Pro | Oracle ACE Associate

For more details on Oracle ACE Program:
ace.oracle.com

Oracle ACE

Nominate

yourself or someone you know:

ace.oracle.com/nominate

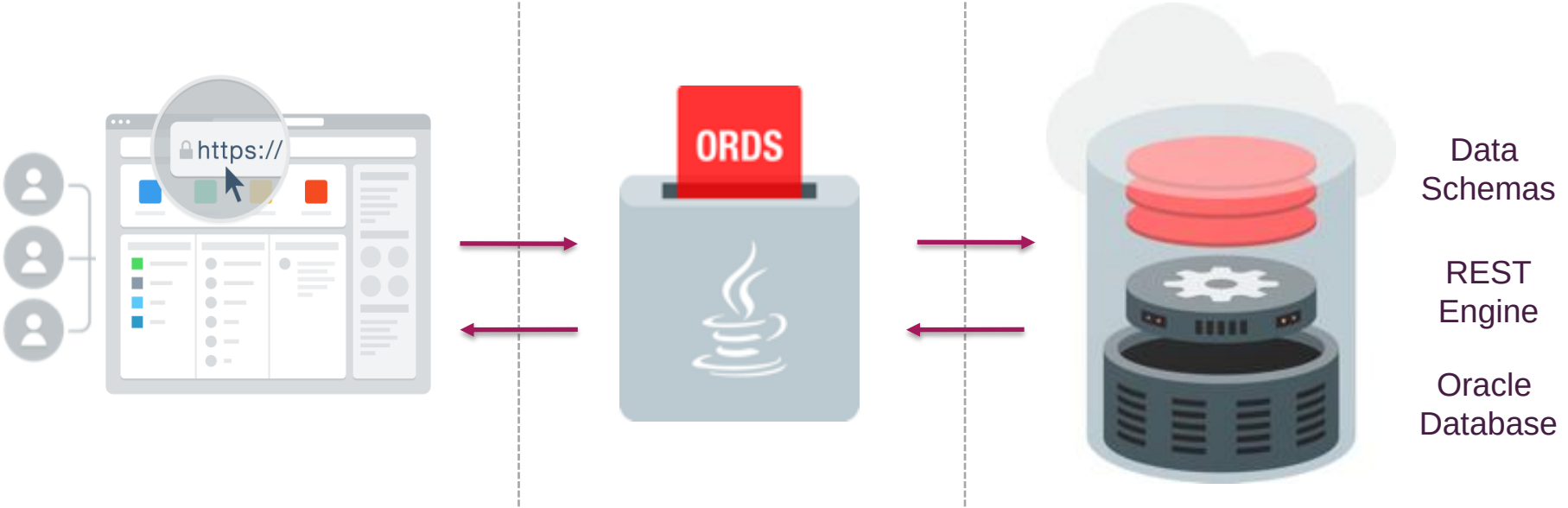
Connect: **aceprogram_ww@oracle.com** Facebook.com/OracleACEs @oracleace

Oracle RDBMS RESTful Apps: An Overview

What is REST

REST	**RE**presentational **S**tate **T**ransfer	doctoral dissertation
		by Roy Fielding, 2000
	programming paradigm	distributed systems
		Web services.
RESTful Applications	implements 6 constraints	**Uniform Interface (API via URIs)**
		Stateless, Client-Server, Layered System
		Cacheable, Code on Demand
Implementation	Transport protocol	http(s)
	content	JSON Documents

Oracle REST Data Services: 3-Tier Architecture



Browser / REST Client

Mid Tier

Database Tier

ORDS = Oracle REST Data Services

Oracle REST Data Services (ORDS)

Evolved

from APEX Listener

ords.war

Java Web Archive

Deploy in Application Server

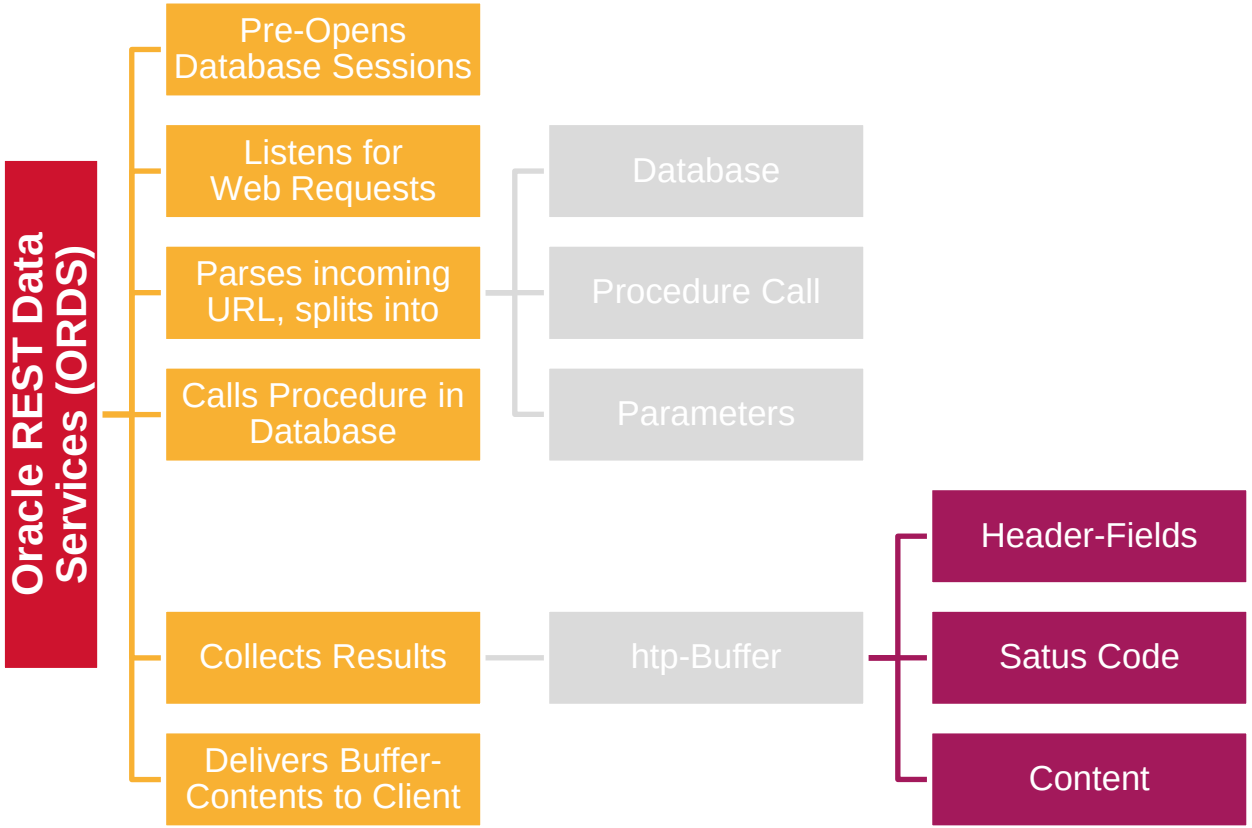
- Tomcat
- Glassfish (deprecated)
- WebLogic

Standalone mode

Brings own http-server
Supported for Production

```
java -jar ords.war
```

What is ORDS' job?



Setting the Stage: Who is Jason?

JSONJava Script Object Notation

Java **S**cript **O**bject **N**otation

Lightweight Format

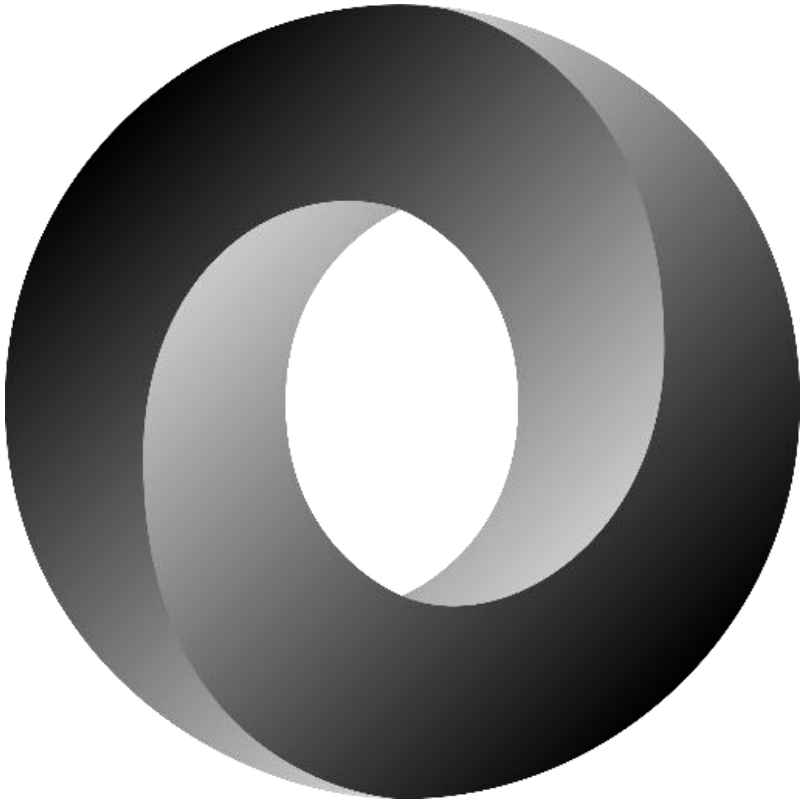
Text-based
Data Interchange
Document Format (can be stored as Files)

Originally designend

to persist JavaScript Objects

Standards:

json.org
ECMA-404



JSON: Types of Values

```
{
  "string":"String are quoted",
  "escape": "\n control chars",
  "number": 1234,
  "boolean": true,
  "collection": {
                  "object":"fields",
                  "can":"nested"
                },
  "array": [ "one",
             "two",
             {"three":true}]
}
```

Types of Values

string	“always enclosed in double quotes”	
number	Integer	-256
	Float	256.123
	E-notation	2.3e3
Boolean	true	
	false	
	null	
Object	Unordered collection	
	Nested Objects or Arrays	
Array	Orderd list of values	
	Simple Types or Objects	

JSON Document Format: Odds & Ends

©whitehoune - stock.adobe.com

Characterset

always Unicode

Number encoding

always English, Decimal Point

Whitespace

between Tokens is ignored
space, linefeed, carriage return, tab

Dates are represented as strings

usually ISO 8601 Zulu (UTC) Time
"2019-11-19**T**15:13:23**Z**"

JSON is Schemaless

no constraints for your implementation
Developers Hell - when dealing with documents not produced by your code
JSON Schema addon – not yet a standard

Querying JSON: Dot-Notation

```
select d.department_data.department
  from departments_json d
```

```
select d.department_data.employees[0].name
  from departments_json d
 where department_id = 110;

EMPLOYEES
Gietz, William
```

```
select d.department_data.employees[*].name
  from departments_json d
 where department_id = 110;

EMPLOYEES
["Gietz, William","Higgins, Shelley"]
```

```
-- Column needs is_json constraint
-- or "treat as json" function (18c)

with j_data as (
  select treat (
         d.department_data as json
        ) as department_data
  from   departments_json d
  where  department_id = 110
)
  select j.department_data.department
  from   j_data j
  where  department_data is json;

DEPARTMENT
Accounting
```

Chris Saxon: How to Store, Query, and Create JSON Documents in Oracle Database

Querying JSON: The JSON_TABLE Operator (1/2)

JSON_TABLE

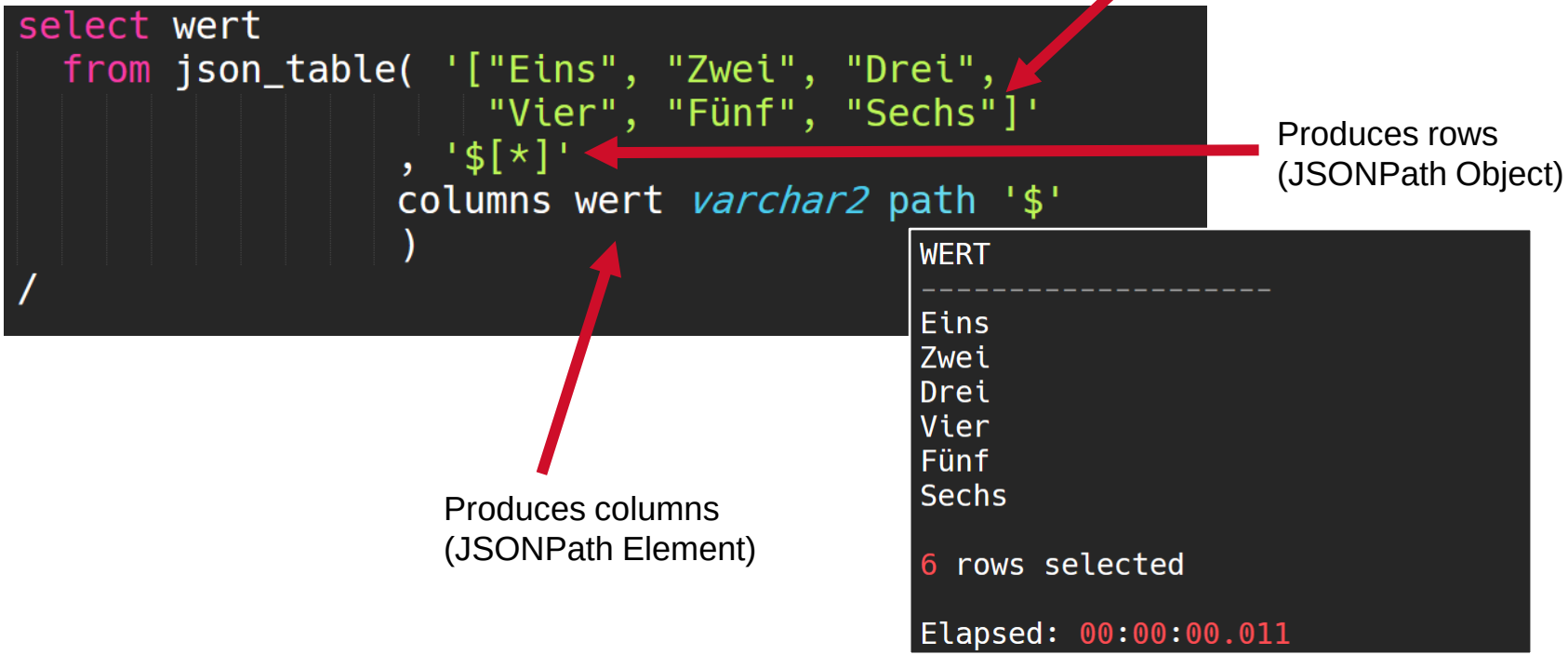
Lives inside the SQL-From-Clause

Produces **Rows** and **Columns**

Accepts LOBs with JSON data

Included in SQL:2016 Standard

Querying JSON: The JSON_TABLE Operator (2/2)



Generating JSON

Generate JSON Documents

- SQL-Functions starting with 12gR2
- JSON_Object
- JSON_Array
- JSON_ObjectAgg
- JSON_ArrayAgg

Generate JSON Documents – JSON_Object

```
select json_object ('id2' value cust_id,
                    'name' value (first ||' '|| last),
                    'joined' value join_date) customers
  from customers;
```

```
CUSTOMERS
----------------------------------------------------------------
{"id2":1,"name":"Eric Cartman","joined":"2017-07-10T16:26:15"}
{"id2":2,"name":"Kenny McCormick","joined":"2017-07-10T16:26:15"}
{"id2":3,"name":"Kyle Brofloski","joined":"2017-07-10T16:26:15"}
{"id2":4,"name":"Stan Marsh","joined":"2017-07-10T16:26:15"}
```

Generate JSON Documents JSON_Array

```
select json_array(first, last) customers
  from customers;
```

```
CUSTOMERS
----------------------
["Eric","Cartman"]
["Kenny","McCormick"]
["Kyle","Brofloski"]
["Stan","Marsh"]
```

Generate JSON Documents JSON_ArrayAgg

```
select json_arrayagg(
         json_object('id'       value item_id,
                     'name'     value name,
                     'quantity' value stock_quantity)
                ) stock_items
  from items
 where stock_quantity > 10;
```

```
STOCK_ITEMS
--------------------------------------------------------
[{"id":345,"name":"Border Patrol Costume","quantity":20},
{"id":429,"name":"Air Guitar","quantity":14}]
```

Working with JSON

Comparing JSON Documents

```
select case
       when json_equal('{"one": 1, "two":[ "three", 4]}'
                  ,'{  "two":[ "three"
                         , 4]
                         ,"one": 1 }'
                  )
        then 'EQUAL'
        else 'DIFFERENT'
      end compare
 from dual;
```

```
COMPARE
--------------
EQUAL
```

```
select case
       when json_equal('{"one": 1, "two":[ "three", 4]}'
                  ,'{"one": 1, "two":[ 4, "three"]}'
                  )
        then 'EQUAL'
        else 'DIFFERENT'
      end compare
 from dual;
```

```
COMPARE
--------------
DIFFERENT
```

Updating JSON Docs – json_merge_patch

JSON_merge_patch()	• – new Feature in 19c
Creates / Updates / Delete	• Objects in JSON Docs Works like Unix patch or SQL merge

```
json_mergepatch('{"FirstName":"Eric", "LastName":"Cartman"}', '{"LastName":„Fox"}')
{"FirstName":"Eric", "LastName":"Fox"}

json_mergepatch('{"FirstName":"Eric", "LastName":"Cartman"}', '{"Salaray":1000}')
{"FirstName":"Eric", "LastName":"Cartman", "Salaray":1000}

json_mergepatch('{"FirstName":"Eric", "LastName":"Cartman"}', '{"FirstName":null}')
{"LastName":"Cartman"}
```

Designing a REST API

APIs should be human readable

“Programs must be written for people to read, and only incidentally for machines to execute.”

Harold Abelson, Structure and Interpretation of Computer Programs, 1984

This applies to APIs, as well.

API Grammar

Nouns / What?

Your API Objects
  e.g. contracts, cars, VirtualMachines, …
  GET /products
  GET /VirtualMachines/4711

Verbs / How?

http Methods
  e.g GET, POST, PUT, DELETE

Relations

Sub-resources
  e.g DELETE
    /VirtualMachines/4711/VMDiskMapping/5

HTTP methods: The Verbs

HTTP Methods

Actions	part of http-request
Common Methods	GET, POST, PUT, DELETE
	OPTIONS, HEAD, TRACE
	CONNECT
Expandable	Make up your own

The HTTP-Protocol - methods

Server

```
# Listens on Port 8080 like a Webserver
nc -l 8080

GET /ords/VM/4711 HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.58.0
Accept: */*


POST /ords/VM/4711 HTTP/1.1
Host: localhost:8080
…

TRALALLA /ords/VM/4711 HTTP/1.1
...
```

Client

```
curl \
    http://localhost:8080/ords/VM/4711



curl --request POST \
    http://localhost:8080/ords/VM/4711


curl --request TRALALLA \
     http://localhost:8080/ords/VM/4711
```

HTTP Status Codes

1xx Informational „Hold on“	100 Continue
	101 Switching Protocols
	102 Processing
2xx Success „Here you go“	200 OK
	201 Created
	208 Already Reported
3xx Redirection „Go away“	301 Moved Permanently
	304 Not modified
	307 Temporary Redirect
4xx Client Error „You fucked up“	400 Bad Request
	401 Unauthorized
	404 Not Found
5xx Server Error „I fucked up“	500 Internal Server Error
	502 Bad Gateway
	503 Service Unavailable

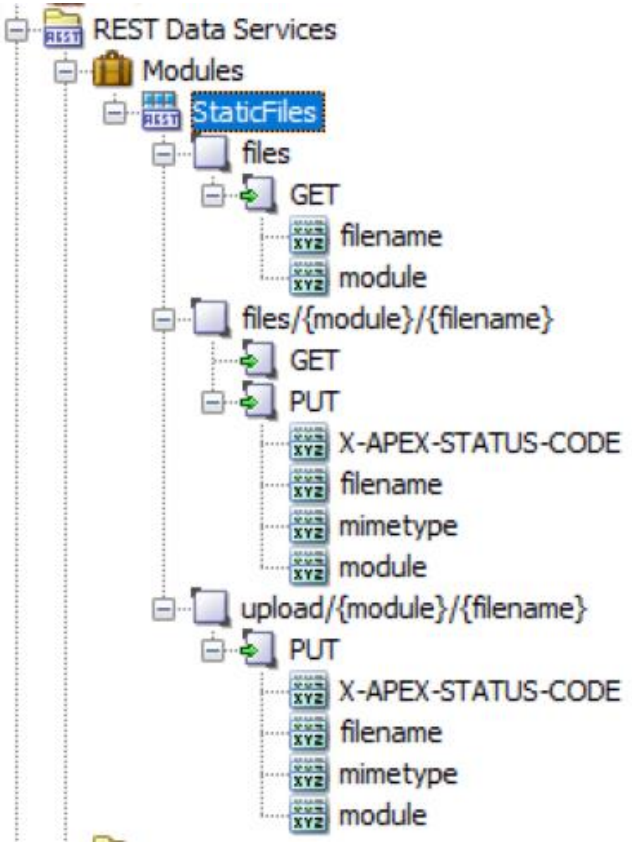
API Design Best Practices

- Try it, Test it
- Be redundant
- Use nouns, but no verbs, Nouns are plural
- GET method should never alter states
- Use HTTP headers
- Use **H**ypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate (HATEOAS)
- Provide Filtering, Sorting, Field Selection & Paging

Modules, Template & Handlers

REST Data Service

Module	logically groups a set of URLs	Like a PL/SQL Package
	Name	Include API Version# In Name & URI Prefix
	URI Prefix	Part of the URL
Template	URI Pattern	following URI Prefix from Module may contain parameters
Handler	http-Methods	GET, POST PUT, DELETE one per Template
	Parameters	http-header URI
	Your Code	goes here



REST Services – PL/SQL enable schema

```
ORDS.ENABLE_SCHEMA(
    p_enabled             => TRUE,
    p_schema              => 'APEX_USER',
    p_url_mapping_type    => 'BASE_PATH',
    p_url_mapping_pattern => 'util',
    p_auto_rest_auth      => FALSE);
```

REST Services – PL/SQL handle GET all Files

```
ORDS.DEFINE_MODULE(
    p_module_name    => 'StaticFile
    p_base_path      => '/static/',
    p_items_per_page =>  25,
    p_status         => 'PUBLISHED'
    p_comments       => NULL);
```

```
ORDS.DEFINE_TEMPLATE(
    p_module_name    => 'StaticFile
    p_pattern        => 'files',
    p_priority       => 0,
    p_etag_type      => 'HASH',
    p_etag_query     => NULL,
    p_comments       => NULL);
```

```
ORDS.DEFINE_HANDLER(
    p_module_name    => 'StaticFiles',
    p_pattern        => 'files',
    p_method         => 'GET',
    p_source_type    => 'json/query',
    p_items_per_page =>  50,
    p_mimes_allowed  => '',
    p_comments       => NULL,
    p_source         =>
'select ''files/''||module||''/''||filename pk_uri
     , module
     , filename
     , mimetype
     , modified
  from static_files'
    );
```

REST Services – PL/SQL handler PUT File

```
ORDS.DEFINE_TEMPLATE(
      p_module_name    => 'StaticFiles',
      p_pattern        => 'files/{module}/{filename}',
      p_priority       => 0,
      p_etag_type      => 'HASH',
      p_etag_query     => NULL,
      p_comments       => NULL);
  ORDS.DEFINE_HANDLER(
      p_module_name    => 'StaticFiles',
      p_pattern        => 'files/{module}/{filename}',
      p_method         => 'PUT',
      p_source_type    => 'plsql/block',
      p_items_per_page =>  0,
      p_mimes_allowed  => 'application/octet-stream',
      p_comments       => NULL,
      p_source         =>
'begin
 merge into static_files dst
   using (select :module module
                , :filename filename
                , :mimetype mimetype
                , :body content
                , sysdate modified
            from dual) src
   on ( dst.module = src.module
    and dst.filename = src.filename )
when matched then update
 set dst.mimetype = src.mimetype
   , dst.content  = src.content
   , dst.modified = src.modified
when not matched then
 insert (module, filename, mimetype, content, modified)
             values ( src.module, src.filename, src.mimetype, src.conter
;
 :status := 201;
end;'
      );
```

```
ORDS.DEFINE_PARAMETER(
    p_module_name         => 'StaticFiles',
    p_pattern             => 'files/{module}/{filename}',
    p_method              => 'PUT',
    p_name                => 'X-APEX-STATUS-CODE',
    p_bind_variable_name  => 'status',
    p_source_type         => 'HEADER',
    p_param_type          => 'INT',
    p_access_method       => 'OUT',
    p_comments            => NULL);
ORDS.DEFINE_PARAMETER(
    p_module_name         => 'StaticFiles',
    p_pattern             => 'files/{module}/{filename}',
    p_method              => 'PUT',
    p_name                => 'mimetype',
    p_bind_variable_name  => 'mimetype',
    p_source_type         => 'HEADER',
    p_param_type          => 'STRING',
    p_access_method       => 'IN',
    p_comments            => NULL);
```

Conclusion

Wanna Try? – Oracle Cloud Free Tier

Always Free cloud services

Services you can use for an unlimited time.

Databases

Two Oracle Autonomous Databases 20GB Data
Oracle APEX, ORDS and SQL Developer

Compute

Two AMD Compute VMs
Up to 4 instances of ARM Ampere A1 Compute with
3,000 OCPU hours and 18,000 GB hours per month

A lot More

Block, Object, and Archive Storage
Load Balancer and data egress
Monitoring and Notifications

Dedicated Talk

Erstberührung mit Oracle Cloud FreeTier, LiveSQL, LiveLabs und Co.
Mittwoch 24.05.2023 | 11:15 - 12:00 | Amsterdam 1

REST & JSON: Backbone of Oracle Based modern Apps

ORDS RESTful Services

Powerful & Flexible

REST enable Tables & Views quickly with AutoREST

JSON

JSON is a flexible Interchange Format

- Lingua Franca of modern Web Apps
- Working with JSON is easy
  - It‘s just Text