Using ORDS, REST, PL/SQL and Oracle Text to build a Text Search API

Presented on 12th of October 2022
at

HrOUG 2022

Rovinj

by

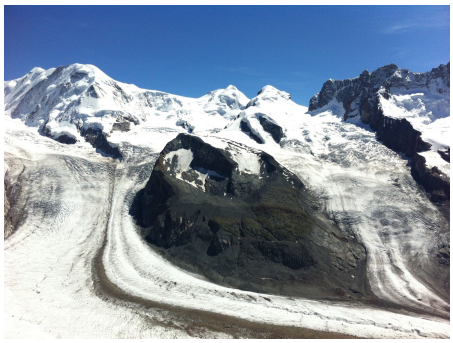
Niall Mc Phillips - Long Acre sàrl
niall.mcphillips@longacre.ch
@Niall_McP

long acre
solutions today

1




2



3

4

About me: Niall Mc Phillips

Owner - Long Acre sàrl

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

5

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 ACE Director | Oracle ACE Pro | Oracle ACE Associate

Oracle ACE

ace.oracle.com/nominate

ace.oracle.com

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

6

What we're going to see today

- A short intro to Oracle Text searching
- Getting started with ORDS
- Creating an ORDS module
- Creating ORDS templates and handlers
- Creating PL/SQL web services for ORDS
- Adding features to the web services
- Securing your web service using OAuth2

What is Oracle Text

- 1st version with Oracle8 (1997) was called Oracle ConText (hence the CTXSYS schema name).
- Oracle8i (1999) renamed to Oracle Intermedia Text.
- Oracle9i (2001) renamed to Oracle Text
- An integral part of all Oracle databases *including Oracle Express Edition (XE) and Autonomous DB.*
- Out of the box - Everything is already there inside of your Oracle DB!

Searching using Oracle Text

- *really fast* and quite easy to start using
- just create an index and start searching
- index varchar2, XML, JSON, clobs and blobs (like pdfs)
- uses the “contains” clause for querying
- allows AND/OR and more complex logic
- + many more advanced features…

Searching with “like”

This is the basic “naïve” textual search that can work for very small datasets.

- it will not use an index if there is a wildcard at the start of the search string

```
where mytext like '%dog%'
```

- it is case-sensitive
  - *`where lower(mytext) like '%dog%'`*

Creating a simple Oracle Text index

Example: table HIST_EVENTS

(34'000 rows)

	COLUMN_NAME	DATA_TYPE
1	ID	NUMBER
2	THEDATE	VARCHAR2(255 BYTE)
3	CATEGORY1	VARCHAR2(255 BYTE)
4	CATEGORY2	VARCHAR2(255 BYTE)
5	DESCRIPTION	VARCHAR2(255 BYTE)

Creating a simple Oracle Text index

```
create index indexname
on tablename (columnname)
indextype is ctxsys.context;
```

```
create index txt_hist_events$1
          on hist_events (description)
indextype is ctxsys.context;

Index TXT_HIST_EVENTS$1 created.

(6 secs)
```

Searching with contains

```
select * from tablename
where
contains(searchcolumn,'searchtext') > 0;
```

Cloud DEMO on Autonomous DB – Basic Searches

Table HIST_EVENTS

- ID
- THEDATE
- CATEGORY1
- CATEGORY2
- DESCRIPTION – we will index this column

Scoring search results

- The **score** of a search result gives an idea of the relevance of the result. High score indicates a higher relevance.
- Scores are always in the 1 to 100 range
- Scores have absolutely no meaning outside of their own query and cannot be compared between different queries, sub-queries or datasets.

Scoring search results - syntax

```
select score(1), t.* from tablename t
where
contains(searchcolumn,'searchtext',1) > 0
order by 1 desc;
```

Note that the (1) in score(1) matches the ,1) in the contains clause

Oracle Text operator grammar and syntax

Searching with AND and OR operators

Operator	Symbol	Description	Example Expression
`AND`	&	Use the **AND** operator to search for documents that contain at least one occurrence of *each* of the query terms. Score returned is the minimum of the operands.	`'cats AND dogs'` `'cats & dogs'`
`OR`	|	Use the **OR** operator to search for documents that contain at least one occurrence of *any* of the query terms. Score returned is the maximum of the operands.	`'cats | dogs'` `'cats OR dogs'`

Searching with NOT and ACCUM operators

`NOT`	~	Use the `NOT` operator to search for documents that contain one query term and not another.	To obtain the documents that contain the term *animals* but not *dogs*, use the following expression: `'animals ~ dogs'`
`ACCUM`	,	Use the `ACCUM` operator to search for documents that contain at least one occurrence of any of the query terms. The accumulate operator ranks documents according to the total term weight of a document.	The following query returns all documents that contain the terms *dogs*, *cats* and *puppies* giving the highest scores to the documents that contain all three terms: `'dogs, cats, puppies'`

long acre
solutions today

Some other operators

EQUIValence (=)

NEAR (;)

MINUS (-)

stem ($)

Fuzzy

soundex (!)

and many more…

full details in Oracle Text Reference at:

https://docs.oracle.com/cd/B28359_01/text.111/b28304/cqoper.htm#CCREF0300

long acre
solutions today

Autonomous Database on Oracle Cloud Demo

- **Searches with CONTAINS**

Escaping terms entered

search for

- Africa and Near East
- “Near” is also an operator so we escape the search words using curly brackets {}

  {Africa}&{Near East}

Preparing text for search

- It can quickly become quite complex to parse and prepare the search text that users enter
- Normally some type of pre-processing is required for real-world scenarios

Pre-processing user-input text for Google-like searches

Baseline principles:

- End-users should not need to know or understand Oracle*Text grammar
- Everyone wants their searches to work “just like Google”

Pre-processing user-input text for Google-like searches

One approach to pre-processing

```
FOR i IN 1..50 LOOP  -- try to get rid of multiple spaces
        v_text := replace(v_text,'  ',' ');
    END LOOP;
    v_text := replace(v_text,'*','%');  -- wildcard chars
    v_text := replace(v_text,'?','_');  -- wildcard chars
    v_text := replace(v_text,'%',null);
    v_text := replace(v_text,'''',null);
    v_text := replace(v_text,',',null);
    v_text := replace(v_text,';',null);
    v_text := replace(v_text,'.',null);
    v_text := replace(v_text,'+','&');
    v_text := replace(v_text,' &','&');
    v_text := replace(v_text,'& ','&');
    etc...
```


Pre-processing user-input text for Google-like searches

- While researching for this presentation I found a great PL/SQL package* written and made freely available by Roger Ford, the Oracle Text Product Manager.

PARSER package:

https://blogs.oracle.com/searchtech/oracle-text-query-parser

I really wish I had found this a few years ago - I would have saved so much time that I spent writing my own ;)

The PARSER package

We will use the parser.simpleSearch function to transform "Google-like" syntax into Oracle Text syntax.

e.g. "Ad Hoc Committee" becomes
({Ad Hoc Committee})

PARSER examples

assessment damages becomes
({assessment},{damages})

+assessment +damages becomes
({assessment}&{damages})

+assessment -damages becomes
({assessment}) ~{damages}

What is ORDS

Oracle REST Data Services (ORDS) bridges HTTPS and your Oracle Database.

A mid-tier Java application, ORDS provides

- a Database Management REST API,
- SQL Developer Web,
- a PL/SQL Gateway,
- SODA for REST,

and the ability to publish RESTful Web Services for interacting with the data and stored procedures in your Oracle Database.

Getting started with ORDS

Step 1: Enable your schema – *only needs to be done once*

in PL/SQL:

```
begin
  ords.enable_schema
    (p_enabled => true,
     p_schema => 'SCHEMANAME',
     p_url_mapping_type => 'BASE_PATH',
     p_url_mapping_pattern => 'hist',
     p_auto_rest_auth => false);
  commit;
end;
/
```

Create a procedure to search

Create a PL/SQL procedure to perform the search

```
procedure searchEvents (p_text in varchar2) is
   v_clob  clob;
begin
   ....
   do something ...
   ....

   owa_util.mime_header('application/json');
   outputClob(v_clob);

end searchEvents;
```


Create an ORDS module

in PL/SQL:

```
begin
  ords.define_module(
    p_module_name    => 'events',
    p_base_path      => 'events/',
    p_items_per_page => 0);
  commit;
end;
/
```

Note: this is a destructive command and will remove the existing module

long acre
solutions today

Create a template and handler (1)

```
begin
   ords.define_template(
   p_module_name    => 'events',
   p_pattern        => 'search/:text');

  ords.define_handler(
    p_module_...
```


Create a template and handler (2)

```
….
ords.define_handler(
    p_module_name    => 'events',
    p_pattern        => 'search/:text',
    p_method         => 'GET',
    p_source_type    => ords.source_type_plsql,
    p_source         =>
'begin
    histEvents.searchEvents(p_text => :text);
 end;');
Commit;
end;
/
```

Test your ORDS URL

The ORDS URL on the Oracle Autonomous Cloud can be found under “Service Console” -> “Development”

For example, the base URL one that I’m using for this presentation is something like this:

https://LP5XX9XYZ98EKBTM-LONGACRE21.adb.eu-frankfurt-1.oraclecloudapps.com/ords/

So our historical events module URL resembles this:

https://LP5XX9XYZ98EKBTM-LONGACRE21.adb.eu-frankfurt-1.oraclecloudapps.com/ords/hist/events/search/:text

Search returning a json object list

```
….
select json_object
       (key 'score' is score(1),
        key 'date' is h.thedate,
        key 'description' is h.description)
  from hist_events h
 where contains(h.description, p_text, 1) > 0
order by h.id;
```

Returns a list of objects… malformed json - 1

Search returning a json array

```
….
select json_arrayagg(
        json_object
         (key 'score' is score(1),
          key 'date' is h.thedate,
          key 'description' is h.description)
               returning clob) as Events
  from hist_events h
 where contains(h.description, p_text, 1) > 0;
```

Returns an array of json objects - 2

Add a result count (1)

```
select count(*)
     into v_count
     from hist_events h
    where contains(h.description, p_text, 1) > 0;
```

3

Add a result count (2)

```
select
  json_object
   ('resultcount'  VALUE v_count,
    'events'       VALUE
      json_arrayagg(
         json_object
           (key 'score' is score(1),
            key 'date' is h.thedate,
            key 'description' is h.description)
                                  returning clob)
     returning clob)
  into v_clob
  from hist_events h
 where contains(h.description, p_text, 1) > 0;

4
```


Add pagination(1) – module

```
begin

  ords.define_template(
   p_module_name    => 'events',
   p_pattern        => 'search4/:text');

  ords.define_handler(
    p_module_name    => 'events',
    p_pattern        => 'search4/:text',
    p_method         => 'GET',
    p_source_type    => ords.source_type_plsql,
    p_source         =>
'begin
  histEvents.searchEvents4
  (p_text => :text,
   p_offset => nvl(:offset,0),
   p_pagesize => nvl(:pagesize,10));
end;',
    p_items_per_page => 0);

  commit;
end;
/
```

Add pagination(2) – modify procedure

```
select json_object
       ('resultcount'  VALUE v_count,
        'events' value json_arrayagg(
                          json_object(key 'score' is h.score,
                                      key 'id'    is h.id,
                                      key 'date'  is h.thedate,
                                      key 'description' is h.description)
                          returning clob)
        returning clob)
  into v_clob
  from (select he.*,
               score(1) as score
          from hist_events he
         where contains(he.description, p_text, 1) > 0
         order by id
        offset p_offset rows fetch next p_pagesize rows only) h;
```

Note that the offset and pagesize are in the sub-query

Return HTTP status codes(1)

Add to module(1)

```
  ords.define_handler(
    p_module_name    => 'events',
    p_pattern        => 'search5/:text',
    p_method         => 'GET',
    p_source_type    => ords.source_type_plsql,
    p_source         =>
'begin
  histEvents.searchEvents5
  (p_text     => :text,
   p_offset   => nvl(:offset,0),
   p_pagesize => nvl(:pagesize,10),
   po_http_status => :status);
end;',
    p_items_per_page => 0);
```

5

Return HTTP status codes(2)

Add to module(2)

```
ords.define_parameter
     (p_module_name        => 'events',
      p_pattern            => 'search5/:text',
      p_method             => 'GET',
      p_name               => 'X-ORDS-STATUS-CODE',
      p_bind_variable_name => 'status',
      p_source_type        => 'HEADER',
      p_access_method      => 'OUT');
```


Return HTTP status codes(3)

```
procedure searchEvents5 (p_text     in varchar2,
                         p_offset   in integer default 0,
                         p_pagesize in integer default 10,
                         po_http_status out number)
is
  v_count   integer;
  v_clob    clob;
begin

   po_http_status := 200;
.................

   if v_count = 0 then
      po_http_status := 404;
   end if;
   owa_util.mime_header('application/json');
   outputClob(v_clob);

exception
  when others then
    po_http_status := 500;
    htp.p('*Error*: '||sqlcode||' - '||sqlerrm);
end searchEvents5;
```

Extracting text snippets (1)

Using the built-in Oracle Text package CTX_DOC. The SNIPPET procedure highlights where a piece of relevant text was found with some context and highlighting.

```
ctx_doc.snippet(index_name => 'TXT_HIST_EVENTS$1',
                textkey    => he.rowid,
                text_query => p_text,
                starttag   => '**',
                endtag     => '**',
                separator  => '... ...') as snippet
```

6

Securing your webservice module Overview

1. Create a module containing the webservices that you want to require OAuth2 authentication
2. Create a role e.g. *events_role*
3. Create a privilege to define which role is required for a specific module
4. Create an OAuth2 *client application* with a client_id and client_secret
5. Grant the created role to the *client application*
6. Query the created application to find the client_id and client_secret

Securing your webservice module

2. *Create a role e.g. events_role*

```
begin
 ords.create_role
  (p_role_name => 'events_role');
 commit;
end;
```


Securing your webservice module

1. Create a module containing the webservices that you want to require OAuth2 authentication
2. Create a role e.g. *events_role*
3. Create a privilege to define which role is required for a specific module
4. Create an OAuth2 *client application* with a client_id and client_secret
5. Grant the created role to the *client application*
6. Query the created application to find the client_id and client_secret

Securing your webservice module

3 *Create a privilege to define which role is required for a specific module (1)*

```
declare
  l_roles     owa.vc_arr;
  l_modules   owa.vc_arr;
  l_patterns  owa.vc_arr;
begin
  l_roles(1) := 'events_role';
  l_modules(1) := 'events';
  ords.define_privilege
     ... continued on next page
```


Securing your webservice module

3 *Create a privilege to define which role is required for a specific module (contd.)*

```
...continued from previous page
...
  ords.define_privilege
   (p_privilege_name => 'rest.events',
    p_roles          => l_roles,
    p_patterns       => l_patterns,
    p_modules        => l_modules,
    p_label          => 'Events Privileges',
    p_description    => 'Secures Events API access',
    p_comments       => null);
  commit;
end;
```

Securing your webservice module

1. Create a module containing the webservices that you want to require OAuth2 authentication
2. Create a role e.g. *events_role*
3. Create a privilege to define which role is required for a specific module
4. Create an OAuth2 *client application* with a client_id and client_secret
5. Grant the created role to the *client application*
6. Query the created application to find the client_id and client_secret

Creating an OAuth2 Client Application

4. Create an OAuth2 client application with a client_id and client_secret

```
begin
  oauth.create_client
    (p_name            => Events Client 1',
     p_grant_type      => 'client_credentials',
     p_description     => 'Events API access',
     p_support_email   => 'appowner@xyz.org',
     p_privilege_names => null);
end;
```

Securing your webservice module

1. Create a module containing the webservices that you want to require OAuth2 authentication
2. Create a role e.g. *events_role*
3. Create a privilege to define which role is required for a specific module
4. Create an OAuth2 *client application* with a client_id and client_secret
5. Grant the created role to the *client application*
6. Query the created application to find the client_id and client_secret

Creating an OAuth2 Client Application

5. Grant the created role to the client application

```
begin
  oauth.grant_client_role
       (p_client_name => 'Events Client 1',
        p_role_name   => 'events_role');
  commit;
end;
```

Securing your webservice module

1. Create a module containing the webservices that you want to require OAuth2 authentication
2. Create a role e.g. *events_role*
3. Create a privilege to define which role is required for a specific module
4. Create an OAuth2 *client application* with a client_id and client_secret
5. Grant the created role to the *client application*
6. Query the created application to find the client_id and client_secret

Creating an OAuth2 Client Application

6. Find the client ID and secret

```
select name,
       client_id,
       client_secret
  from user_ords_clients;
```

Securing your webservice module

1. Create a module containing the webservices that you want to require OAuth2 authentication
2. Create a role e.g. *events_role*
3. Create a privilege to define which role is required for a specific module
4. Create an OAuth2 *client application* with a client_id and client_secret
5. Grant the created role to the *client application*
6. Query the created application to find the client_id and client_secret