

Build Fast, Secure Web Applications with the PL/SQL Gateway and Oracle Multimedia

Background

Oracle Multimedia is a feature of Oracle Database that enables the efficient management and retrieval of image, audio, and video data in the database. It supports most popular multimedia formats (including DICOM, the standard medical image format) with built-in metadata extraction and basic image processing. Oracle Multimedia PL/SQL and Java APIs facilitate rapid application development of multimedia applications. Oracle Multimedia provides the infrastructure for secure and scalable data management of multimedia data.

The PL/SQLGateway, a feature of Oracle Database, enables the creation of web applications using PL/SQL. It enables web browsers to invoke PL/SQL stored procedures that can perform database operations and display results on a web page.

Workshop Introduction

In this workshop attendees will learn how to create multimedia web pages. They will use the PL/SQL Gateway to create web pages that will access and display multimedia content stored in the database using Oracle Multimedia. Attendees will: (1) configure a freshly created Oracle database to display multimedia web pages, (2) create a table to store multimedia data, and (3) write procedures to perform operations required by a multimedia application.

The database features used in this workshop are Oracle Multimedia and the embedded PL/SQL Gateway (Apache – versions 1.3 and 2.0 – can also be used but is not covered in this workshop). The database being used in this workshop is Oracle 11gR1, but this setup will work equally well on Oracle 10gR2 and Oracle 11gR2. It is also designed to work on Unix, Linux, Windows, and all Oracle supported platforms. Platform specific areas are highlighted. This workshop is not supported on Oracle XE (running 10.2.0.1) as Oracle Multimedia is not supported on this database version.

Basic knowledge of SQL, HTML and CSS is expected in this workshop.

The workshop is divided into two sections.

A. Configuration of the database and quick load of examples multimedia data: This section shows how quickly one can web enable an Oracle Database by configuring the components necessary for the embedded PL/SQL gateway. This section will also create multimedia tables and load some data into them.

B-E. Multimedia Application Development: These sections show how easily one can create web enabled multimedia applications using Oracle Multimedia and the embedded PL/SQL Gateway. HTML forms are created to display, load, and process multimedia images. The final advanced section will go through the basics of securing the application from URL modification and SQL code injection.

Note: To run the SQL code that is part of this document, you can type or copy and paste the short commands into SQL*Plus. For the longer scripts you can copy and paste the code into a file with an extension ‘.sql’. and then execute them as follows:

```
SQL> @script.sql
```

The scripts are also available in the directory C:\sqlscripts.

A: Configuration

A.1: Database Configuration

The goal of this section is to show you how to configure Oracle Database for the embedded PL/SQL gateway. In this section you will also create a schema to hold multimedia tables. Basic knowledge of configuring Oracle is assumed.

All configuration is done via SQL*Plus. SQL*Plus represents the lowest common interface available across all Oracle versions. The command line interface when used with a text editor enables quick and easy modification of most Oracle features.

Tools like Oracle SQL Developer and Enterprise Manager can also be used, but it is best to have a good understanding of the core capabilities before using GUI development tools.

a. Use SQL*Plus to connect to the database as SYS:

```
sqlplus / as sysdba
```

b. Check to make sure the database parameter dispatchers is correctly configured to allow the database to talk to the listener.

```
SQL> show parameter dispatchers
```

The output should look like the following:

NAME	TYPE	VALUE
dispatchers	string	(PROTOCOL=TCP) (SERVICE=orclXDB)

orcl is the SID (unique identifier) of the database installed on your machine. You can find out the SID of your database using the command:

```
select name from v$database;
NAME
-----
ORCL
```

If this parameter is not set or is not set correctly, then the database will not be able to talk to the listener and the embedded gateway will not work.

c. Unlock the anonymous account:

```
alter user anonymous ACCOUNT UNLOCK;
```

The anonymous account is used by Oracle to connect to the database to find additional connection information about the embedded gateway. The gateway also uses Oracle XML DB, so with the query in the next check that Oracle XML DB is enabled:

d. Check that Oracle XML DB is enabled, required for the gateway. This is done by checking that user SYS has the XDBADMIN role:

```
select * from dba_role_privs where granted_role = 'XDBADMIN';
```

e. All web interfaces use a port to connect to the database. The Oracle Listener (which will handle all incoming HTTP requests and will act as a HTTP server, similar to Apache), needs to know which port to listen on for incoming HTTP Requests. Talk to your network administrators to determine which port you

can use.

The script below will open up two ports. One is for FTP and one is for HTTP requests.

Windows platforms can access port 80 without any problems. Unix platforms require root access to ports with numbers less than 1024. This means starting up the listener with permissions equivalent to that of root which can get quite tricky, which is why on Unix most sites use HTTP ports above 1024.

For this workshop, we will use port 8890 for http and port 2121 for ftp:

```
DECLARE
  v_cfg XMLType;
BEGIN
SELECT updateXML(DBMS_XDB.cfg_get(),
                '/xdbconfig/descendant::ftp-port/text()', '2121',
                '/xdbconfig/descendant::http-port/text()', '8890')
  INTO v_cfg FROM DUAL;
DBMS_XDB.cfg_update(v_cfg);
COMMIT;
END;
/
```

f. We will create a schema called `multimedia` for use in this workshop. All tables and programs will be created in this schema. The following commands create the schema and give it the required permissions.

We give minimal permissions to the schema. Additional permissions can be given as required. Remember that you are still connected as `SYS`.

```
create user multimedia identified by workshop;
grant create session, create table, create procedure to multimedia;
grant create view, create role, create trigger to multimedia;
```

-- We assume tablespace `temp` and tables `users` exist. They were created beforehand.

```
alter user multimedia temporary tablespace temp;
alter user multimedia quota unlimited on users;
alter user multimedia default tablespace users;
```

– The following two permissions are optional but can prove to be useful. They allow the schema to access `dbms_lock.sleep` command (useful for asynchronous processing) and `dbms_file_transfer`, allowing the schema to manipulate external multimedia files.

```
grant execute on dbms_file_transfer to multimedia;
grant execute on dbms_lock to multimedia;
```

g. In this step we create the HTTP DAD (or Device Access Descriptor). The DAD defines the mapping between the virtual directory and the actual schema

e.g. if the URL to access was <http://localhost:8890/mm/program>

Then `/mm/` is the virtual directory and needs to be mapped to the schema in the database.

To do this step, we use the `dbms_epg` PL/SQL package. We define the DAD and control which PL/SQL programs will access it.

-- Create the DAD

-- Note: On first call will return an error as DAD doesn't exist, can be ignored.

```
exec dbms_epg.drop_dad( 'PICTION' );
```

-- This command defines the DAD gives it the name 'mm'. It is case sensitive. We can fine tune it to control which programs (PL/SQL packages) it can access, but for now let us give it full access.

```
exec dbms_epg.create_dad( 'PICTION', '/mm/*' );
```

-- We then map it to a schema. With the embedded gateway, we don't need to manage a password. If we had used mod PL/SQL and Apache then the password should be stored in a dad.conf file (which is encrypted).

```
exec dbms_epg.set_dad_attribute( 'PICTION', 'database-username', 'MULTIMEDIA' );
```

-- These parameters describe how the DAD is accessed. Other Oracle products like APEX or ones that use SSO might have different options here.

```
exec dbms_epg.set_dad_attribute( 'PICTION', 'authentication-mode', 'Basic' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'error-style', 'DebugStyle' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'session-state-management', 'StatelessWithFastResetPackageState' );
```

-- These parameters are important security safeguards that ensure packages which can be accessed from within SQL*Plus by multimedia cannot be accessed from the URL. This is one of the important steps to safeguard against URL modification.

```
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'sys.*' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'dbms.*' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'utl.*' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'owa.*' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'owa.*' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'http.*' );
exec dbms_epg.set_dad_attribute( 'PICTION', 'exclusion-list', 'htf.*' );
```

-- This command defines the table to be used to temporarily store multimedia content (BLOBs) that are loaded via the browser. We will look at this table more closely later in the workshop.

```
exec dbms_epg.set_dad_attribute( 'PICTION', 'document-table-name', 'APACHE_OWS_CONTENT' );
```

-- We then have to authorize the DAD. This is important, because we are effectively saying we are allowing access to the schema without a password using the embedded gateway.

```
exec dbms_epg.authorize_dad( 'PICTION', 'MULTIMEDIA' );
commit;
```

--Note: These commands help if you cannot get in on 8890

--call dbms_xdb.setHttpPort(8890);

```
--alter system register;
```

h. Now that the schema has been created and the gateway configured, we can create some storage to hold the tables:

```
-- We create this tablespace to store the multimedia data
CREATE TABLESPACE IMAGES EXTENT MANAGEMENT LOCAL UNIFORM SIZE 10M
segment space management auto datafile 'multimedia.tbs' size 100M
autoextend on next 100M;
```

```
alter user multimedia quota unlimited on IMAGES;
```

A number of external access points can be used when accessing external files. The `DIRECTORY` is one that Oracle Multimedia uses. Other ones include `utl_file` and it is also possible to use Java to access directories directly. But let us first define an external directory that the `multimedia` schema can access.

- Example on Windows (which is the case in this workshop)

```
create directory mmload as 'C:\multimedia_files';
```

- Example on Unix:

```
create directory mmload as '/u01/multimedia_files';
```

```
grant read on directory mmload to multimedia;
```

It is recommended that you now look at the different example files that are found in this directory.

A.2: Multimedia Schema Configuration

i. We have now finished the initial database configuration. Let us now connect to the database as the new multimedia schema.

```
SQL> connect multimedia/workshop
```

We now create a table (which can be called anything, but has to be the same name we used in the configuration steps above with the following command (that is, it has to be called

```
APACHE_OWS_CONTENT):
```

```
-- NO NEED TO RUN THIS LINE OF CODE – this is just to highlight a line of code we ran earlier
exec dbms_epg.set_dad_attribute( 'PICTION', 'document-table-name',
'APACHE_OWS_CONTENT');
```

All the column names in the statement below have to be created. We can also specify the storage parameters for the BLOB column. The storage parameters used below make the BLOB column use Oracle SecureFiles storage, which is for high-performance storage and retrieval (Oracle SecureFiles can read and write data at device speed, see

http://www.oracle.com/technology/products/multimedia/pdf/dicom/ora_dicom_bench_2008.pdf for a benchmark).

```
create table apache_ows_content
(
    name                varchar2(1000) unique not null,
```

```

        mime_type      varchar2(128),
        doc_size       number,
        dad_charset    varchar2(128),
        last_updated   date,
        content_type   varchar2(128),
        length         number,
        oid            number,
        blob_content   blob
    )
LOB (blob_content) STORE AS SECUREFILE l_blob_apache
    (TABLESPACE IMAGES
     enable storage in row
     STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0)
     NOCACHE LOGGING)
tablespace USERS pctfree 0 storage( pctincrease 0 maxextents
unlimited);

```

j. We are now ready to do a simple *ping* test to see if the web site is working. We first create a simple procedure which will display a basic page and indicate the site is alive:

```

create procedure ping_alive
as
begin
    http.htmlopen;
    http.bold( 'This site is alive' );
    http.htmlclose;
end ping_alive;
/
sho err

```

(No errors should be seen on screen)

Now from a browser test the site to check whether it is working. You could replace localhost with the domain name of your machine.

```
http://localhost:8890/mm/ping_alive
```

You should see a page appearing saying: **This site is alive**. If you don't, then one of the steps we followed has not worked.

Common reasons for failure:

1. The Oracle Listener is not running (from the command line do a `lsnrctl status`. In the output it should indicate it is listening on port 8890)
2. A firewall is blocking the port (from the command line, try doing a `telnet localhost 8890` to see if a response comes back)

k. We now go in and create the multimedia table that will be used to store our photos, video, audio and documents. All programs we write today will access this table.

Oracle Multimedia has object data types for different types of multimedia data: `ORDImage`, `ORDAudio`, `ORDVideo`, and `ORDDoc`. All of these data types include a `BLOB` attribute to store the multimedia content and other attributes to store metadata extracted from the data. You can refer to **Oracle® Multimedia User's Guide 11g Release 1 (11.1)** and **Oracle® Multimedia Reference**

11g Release 1 (11.1) for more information on these data types. For a quick look at the data types you can describe them from SQL*Plus to see their attributes and methods:

```
SQL> desc ORDSYS.ORDIMAGE
```

Now let us create the table:

```
create table myimages
(
  pk          integer,
  name        varchar2(100),
  image_type  varchar2(100),
  details     varchar2(4000),
  mm_photo_tnail ORDSYS.ORDIMAGE,
  mm_photo    ORDSYS.ORDIMAGE,
  mm_audio    ORDSYS.ORDAUDIO,
  mm_video    ORDSYS.ORDVIDEO,
  mm_doc      ORDSYS.ORDDOC
)
LOB (mm_photo_tnail.source.localdata) STORE AS SECUREFILE
l_mm_photo_tnail
  (TABLESPACE IMAGES
   enable storage in row
   STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0)
   NOCACHE LOGGING)
LOB (mm_photo.source.localdata) STORE AS SECUREFILE l_mm_photo
  (TABLESPACE IMAGES
   disable storage in row
   STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0)
   NOCACHE LOGGING)
LOB (mm_audio.source.localdata) STORE AS SECUREFILE l_mm_audio
  (TABLESPACE IMAGES
   disable storage in row
   STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0)
   NOCACHE LOGGING)
LOB (mm_video.source.localdata) STORE AS SECUREFILE l_mm_video
  (TABLESPACE IMAGES
   disable storage in row
   STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0)
   NOCACHE LOGGING)
LOB (mm_doc.source.localdata) STORE AS SECUREFILE l_mm_doc
  (TABLESPACE IMAGES
   disable storage in row
   STORAGE (MAXEXTENTS UNLIMITED PCTINCREASE 0)
   NOCACHE LOGGING)
tablespace USERS pctfree 20 storage( pctincrease 0 maxextents
unlimited);
```

Note the storage parameters, which gives us fine grained control over how the blob is stored as well as where it is stored. As described above these storage parameters mean that BLOBs in the table will use Oracle SecureFiles storage. See section 8.2 of **Oracle® Multimedia User's Guide 11g Release 1 (11.1)** for more information on how to use Oracle Securefiles. The benchmark at

http://www.oracle.com/technology/products/multimedia/pdf/dicom/ora_dicom_bench_2008.pdf shows the performance advantages.

If you use Oracle Database 10g, remove the word SECUREFILE, as it is not supported in Oracle Database10g.

l. We now can insert the rows of multimedia data into this table. Loading in multimedia data is a two stage process. First, the relational data in the rows and the initialized multimedia objects should be inserted. In the second step the multimedia data itself is loaded.

-- First step: Insert the rows

```
insert into
myimages(pk,name,image_type,details,mm_photo_tnail,mm_photo) values
(100,'Callie','PHOTO','Photo of my Dog in
Australia',ordsys.ordimage.init(),ordsys.ordimage.init());
insert into
myimages(pk,name,image_type,details,mm_photo_tnail,mm_photo) values
(200,'OpenWorld Coffee','PHOTO','Having a break at OpenWorld
2007',ordsys.ordimage.init(),ordsys.ordimage.init());
insert into myimages(pk,name,image_type,details,mm_video) values
(300,'OpenWorld Tradeshow 2007','VIDEO','Delegates wandering around
at 2007 OpenWorld Tradeshow',ordsys.ordvideo.init());
insert into myimages(pk,name,image_type,details,mm_audio) values
(400,'Eternal Donut Introduction Podcast','AUDIO','Opening Podcast
for the Eternal Donut Multimedia. See www.eternal-
donut.com',ordsys.ordaudio.init());
insert into myimages(pk,name,image_type,details,mm_doc) values
(500,'Eternal Donut Paper','DOCUMENT','PDF discussing the eternal
donut podcast',ordsys.orddoc.init());

commit;
```

m. Second step: Load the multimedia content

Now that the rows with the initialization for multimedia data are inserted, we need to load in the multimedia data into each row. As we have a mixture of multimedia types (Photo, Audio, Video) the following PL/SQL procedures show how to load and process each different multimedia type.

When working with multimedia and processing two important points need to be made:

1. It is best to work with one data item at a time
2. When working on a row containing multimedia, it has to be locked. This is done by doing a select for update on the row.

```
declare
  cursor c1(vpk integer) is select * from myimages where pk = vpk for
update;
  crec          myimages%ROWTYPE;
  ctx          RAW(4000) := NULL;
begin
  open c1(100);
  fetch c1 into crec;
  close c1;
-- The first command tells Oracle Multimedia where the physical file
```

```

is
-- Remember that MMLOAD is the name of the directory created in step
h.
  crec.mm_photo.setSource('FILE', 'MMLOAD', 'callie1.jpg' );
-- We then tell Oracle Multimedia to import it.
  crec.mm_photo.import(ctx);
-- We then tell Oracle Multimedia to analyze it and extract its
properties.
  crec.mm_photo.setproperties;
-- We then issue a command to copy it and create a smaller thumbnail
of the original photo.
  crec.mm_photo.processcopy('maxScale=100 100',crec.mm_photo_tnail);

-- We then update the row in the database.
  update myimages set row = crec where pk = crec.pk;
  commit;
end;
/

```

By issuing the following object query from SQL*Plus we can validate that the row was correctly processed:

```

select b.mm_photo_tnail.contentlength photo_tnail,
b.mm_photo.contentlength photo_length from myimages b where pk = 100;

```

n. The following PL/SQL script will load in another photo

```

declare
  cursor c1(vpk integer) is select * from myimages where pk = vpk for
update;
  crec      myimages%ROWTYPE;
  ctx      RAW(4000) := NULL;
begin
  open c1(200);
  fetch c1 into crec;
  close c1;
  crec.mm_photo.setSource('FILE', 'MMLOAD', 'openworld.jpg' );
  crec.mm_photo.import(ctx);
  crec.mm_photo.setproperties;
  crec.mm_photo.processcopy('maxScale=100 100',crec.mm_photo_tnail);
  -- create tnail
  update myimages set row = crec where pk = crec.pk;
  commit;
end;
/

```

```

select b.mm_photo_tnail.contentlength photo_tnail,
b.mm_photo.contentlength photo_length from myimages b where pk = 200;

```

o. The following PL/SQL script will load in the video.

```

declare
  cursor c1(vpk integer) is select * from myimages where pk = vpk for

```

```

update;
  crec      myimages%ROWTYPE;
  ctx      RAW(4000) := NULL;
begin
  open c1(300);
  fetch c1 into crec;
  close c1;
  crec.mm_video.setSource('FILE', 'MMLOAD', 'tradeshow.mov' );
  crec.mm_video.import(ctx);
  crec.mm_video.setproperties(ctx,TRUE);
  update myimages set row = crec where pk = crec.pk;
  commit;
end;
/

set long 5000
select b.mm_video.videoduration video_length, b.mm_video.comments
from myimages b where pk = 300;

```

p. The following PL/SQL script will load in an Audio file:

```

declare
  cursor c1(vpk integer) is select * from myimages where pk = vpk for
update;
  crec      myimages%ROWTYPE;
  ctx      RAW(4000) := NULL;
begin
  open c1(400);
  fetch c1 into crec;
  close c1;
  crec.mm_audio.setSource('FILE', 'MMLOAD', 'episode_0.mp3' );
  crec.mm_audio.import(ctx);
  crec.mm_audio.setproperties(ctx,TRUE);
  update myimages set row = crec where pk = crec.pk;
  commit;
end;
/

set long 5000
select b.mm_audio.audioduration audio_length, b.mm_audio.comments
from myimages b where pk = 400;

```

q. The following script will load in a document. Oracle Text can be used to index the document (but this is beyond the scope of this workshop.). By using Oracle text you can extract an HTML version of the document as well as a summary or gist of the document.

```

declare
  cursor c1(vpk integer) is select * from myimages where pk = vpk for
update;
  crec      myimages%ROWTYPE;
  ctx      RAW(4000) := NULL;
begin
  open c1(500);

```

```
fetch c1 into crec;
close c1;
crec.mm_doc.setSource('FILE', 'MMLOAD', 'podcast_paper.pdf' );
crec.mm_doc.import(ctx,FALSE);
-- crec.mm_doc.setproperties(ctx,TRUE);
update myimages set row = crec where pk = crec.pk;
commit;
end;
/
```

```
set long 5000
select b.mm_doc.mimetype doc_mimetype,
dbms_lob.getlength(b.mm_doc.source.localdata) doc_length from
myimages b where pk = 500;
```

B. Multimedia Application Development

This part of the workshop shows you how to write mod PL/SQL programs to query the tables created in the first section of this workshop and to display output in HTML in a browser. All web pages are dynamically generated. No static html pages are used. The programs do the following:

- a. Query of relational data columns of a table and displaying on a HTML page
- b. Querying of multimedia attributes from the table and displaying on the HTML page
- c. Querying and displaying multimedia content (images) from this table on the HTML Page

- a. Querying the table for relational data.

All HTML page development will be done in mod PL/SQL and managed through the HTP package (which also has a matching HTF package of functions).

Most HTML commands are supported, but if they aren't you can easily put your own commands in using the generic `http.p` package.

As each `http` call is made, the HTML generated is stored in an `owa` PL/SQL array. This means that the generated HTML page can be as large as the available PL/SQL memory. When the procedure being called in the `http` call has finished, the embedded gateway takes this `owa` array and converts it into an HTTP request and returns it back to the browser.

The following procedure shows how you can easily query a table and display its output as an HTML page:

```
create or replace procedure mytable
as
  cursor c1 is select * from myimages order by pk;
begin

  http.htmlopen;
  http.tableopen( cattributes=>'style="border: black solid 1px;
padding: 10px; background-color: #FFFFFF;"' );
  http.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:center; font-weight:bold; background-color:#CCCCCC;"' );
  http.tabledata( htf.bold( 'Primary Key' ) );
  http.tabledata( htf.bold( 'Image Name' ) );
  http.tabledata( htf.bold( 'Image Type' ) );
  http.tabledata( htf.bold( 'Comments' ) );
  http.tablerowclose;
  for clrec in c1 loop
    http.tablerowopen;
    http.tabledata( clrec.pk );
    http.tabledata( clrec.name );
    http.tabledata( clrec.image_type );
    http.tabledata( clrec.details );
    http.tablerowclose;
  end loop;
  http.tableclose;
  http.htmlopen;
end mytable;
```

```
/
sho err
```

Once created, we can then execute the procedure from a URL by simply referring to it as follows:

```
http://localhost:8890/mm/mytable
```

Note: Anonymous PL/SQL blocks cannot be called from the URL. Functions also cannot be called, but Packages can be called.

b. The following procedure queries the tables and shows attributes of the multimedia objects we loaded.

```
create or replace procedure mytableB
as
  cursor c1 is select * from myimages order by pk;
begin

  http.htmllopen;
  http.tableopen( cattributes=>'style="border: black solid 1px;
padding: 10px; background-color: #FFFFFF;"' );
  http.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:center; font-weight:bold; background-color:#CCCCCC;"' );
  http.tabledata( htf.bold( 'Primary Key' ) );
  http.tabledata( htf.bold( 'Image Name' ) );
  http.tabledata( htf.bold( 'Image Type' ) );
  http.tabledata( htf.bold( 'Comments' ) );
  http.tabledata( htf.bold( 'Image Information' ) );
  http.tablerowclose;
  for clrec in c1 loop
    if mod(clrec%ROWCOUNT,2) = 1
    then
      http.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:left; background-color:#F0F0F0;"' );
    else
      http.tablerowopen;
    end if;
    http.tabledata( clrec.pk );
    http.tabledata( clrec.name );
    http.tabledata( clrec.image_type );
    http.tabledata( clrec.details );
    http.p( '<td valign="TOP">' );
    http.tableopen;
    case clrec.image_type
      when 'PHOTO'
      then
        http.tablerowopen;
        http.tabledata( 'Width x Height' );
        http.tabledata( htf.bold(clrec.mm_photo.width || ' x ' ||
clrec.mm_photo.height) );
        http.tablerowclose;
        http.tablerowopen;
        http.tabledata( 'Mimetype' );
        http.tabledata( htf.bold(clrec.mm_photo.mimetype) );
```

```

    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_photo.source.srclocation) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Name' );
    http.tabledata( htf.bold(clrec.mm_photo.source.srcname) );
    http.tablerowclose;

when 'VIDEO'
then
    http.tablerowopen;
    http.tabledata( 'Duration' );
    http.tabledata( htf.bold(clrec.mm_video.videoduration) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Mimetype' );
    http.tabledata( htf.bold(clrec.mm_video.mimetype) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_video.source.srclocation) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Name' );
    http.tabledata( htf.bold(clrec.mm_video.source.srcname) );
    http.tablerowclose;

when 'AUDIO'
then
    http.tablerowopen;
    http.tabledata( 'Duration' );
    http.tabledata( htf.bold(clrec.mm_audio.audioduration) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Mimetype' );
    http.tabledata( htf.bold(clrec.mm_audio.mimetype) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_audio.source.srclocation) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Name' );
    http.tabledata( htf.bold(clrec.mm_audio.source.srcname) );
    http.tablerowclose;

when 'DOCUMENT'
then
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_doc.source.srclocation) );

```

```

        http.tablerowclose;
        http.tablerowopen;
        http.tabledata( 'Name' );
        http.tabledata( htf.bold(c1rec.mm_doc.source.srcname) );
        http.tablerowclose;

        else null;
        end case;
        http.tableclose;
        http.p( '</td>' );
        http.tablerowclose;
    end loop;
    http.tableclose;
    http.htmlopen;
end mytableB;
/
sho err

```

<http://localhost:8890/mm/mytableB>

c. To display multimedia data in the webpage we need to create an additional procedure to handle the binary content of multimedia data. The initial program then references that new procedure to display the binary content. The HTML tag is used. All references to the tag are done asynchronously by the browser, so if you decide to display 100 images on the page, the browser will make 100 calls to retrieve the binary data for each image. Depending on the browser, most will not issue 100 calls, but rather issue 4 or 8 at a time to the server to ensure efficient processing.

Our procedure to handle the multimedia data transfer is called `myphoto_show`, and we use HTTP GET syntax to pass to this procedure the primary key values for the multimedia data.

```

--
-- Now show the thumbnail of the photos
set define off

create or replace procedure mytableC
as
    cursor c1 is select * from myimages order by pk;
begin

    http.htmlopen;
    http.tableopen( cattributes=>'style="border: black solid 1px;
padding: 10px; background-color: #FFFFFF;"' );
    http.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:center; font-weight:bold; background-color:#CCCCCC;"' );
    http.tabledata( htf.bold( 'Primary Key' ) );
    http.tabledata( htf.bold( 'Image Name' ) );
    http.tabledata( htf.bold( 'Image Type' ) );
    http.tabledata( htf.bold( 'Comments' ) );
    http.tabledata( htf.bold( 'Image Information' ) );
    http.tabledata( htf.bold( 'Thumbnail' ) );
    http.tablerowclose;
    for c1rec in c1 loop
        if mod(c1%ROWCOUNT,2) = 1

```

```

then
  http.tablerowopen( cattributes=>'valign="MIDDLE" style="text-align:left; background-color:#F0F0F0;");
  else
    http.tablerowopen;
end if;
http.tabledata( clrec.pk );
http.tabledata( clrec.name );
http.tabledata( clrec.image_type );
http.tabledata( clrec.details );
http.p( '<td valign="TOP">' );
http.tableopen;
case clrec.image_type
when 'PHOTO'
  then
    http.tablerowopen;
    http.tabledata( 'Width x Height' );
    http.tabledata( htf.bold(clrec.mm_photo.width || ' x ' ||
clrec.mm_photo.height) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Mimetype' );
    http.tabledata( htf.bold(clrec.mm_photo.mimetype) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_photo.source.srclocation) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Name' );
    http.tabledata( htf.bold(clrec.mm_photo.source.srcname) );
    http.tablerowclose;

when 'VIDEO'
  then
    http.tablerowopen;
    http.tabledata( 'Duration' );
    http.tabledata( htf.bold(clrec.mm_video.videoduration) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Mimetype' );
    http.tabledata( htf.bold(clrec.mm_video.mimetype) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_video.source.srclocation) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Name' );
    http.tabledata( htf.bold(clrec.mm_video.source.srcname) );
    http.tablerowclose;

when 'AUDIO'

```

```

then
    http.tablerowopen;
    http.tabledata( 'Duration' );
    http.tabledata( htf.bold(clrec.mm_audio.audioduration) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Mimetype' );
    http.tabledata( htf.bold(clrec.mm_audio.mimetype) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Location' );
    http.tabledata( htf.bold(clrec.mm_audio.source.srclocation) );
    http.tablerowclose;
    http.tablerowopen;
    http.tabledata( 'Name' );
    http.tabledata( htf.bold(clrec.mm_audio.source.srcname) );
    http.tablerowclose;

when 'DOCUMENT'
    then
        http.tablerowopen;
        http.tabledata( 'Location' );
        http.tabledata( htf.bold(clrec.mm_doc.source.srclocation) );
        http.tablerowclose;
        http.tablerowopen;
        http.tabledata( 'Name' );
        http.tabledata( htf.bold(clrec.mm_doc.source.srcname) );
        http.tablerowclose;

    else null;
end case;
http.tableclose;
http.p( '</td>' );

http.p( '<td valign="TOP">' );
case clrec.image_type
when 'PHOTO'
    then
-- here we have the command to make a call to myphoto_show to display the image
thumbnail.
        http.anchor( 'myphoto_show?vpk=' || clrec.pk,
htf.img( 'myphoto_show?vpk=' || clrec.pk ||
'&vshow=THUMBNAIL',cattributes=>'border=0 title="' || clrec.name ||
'" alt="' || clrec.name || '"' ));
        else
            http.anchor( 'myphoto_show?vpk=' || clrec.pk, clrec.name);
        end case;
    http.p( '</td>' );

    http.tablerowclose;
end loop;
http.tableclose;
http.htmlopen;

```

```

end mytableC;
/
sho err

```

The procedure `myphoto_show` is crucial to display the binary content of multimedia data. Let us look closely at what it needs to do. To display binary content in HTML pages it needs to know the `mimetype` which is in the header of multimedia data and indicates the type of the binary data. The browser uses the `mimetype` to know how to deal with the data. `image/jpeg`, `image/png`, `image/gif` are examples of `mimetypes`. `image/jpeg` indicates that a binary file is an image with file format `jpeg`.

Oracle Multimedia extracts the `mimetype` as part of the `setProperties` call. This information is stored in the `ORDImage` object allowing us to easily reference it.

The embedded gateway provides two useful commands for dealing with binary data:

1. `owa_util.mime_header` – allows us to specify that the HTTP request coming back is binary and has the specified data type
2. `wpg_docload.download_file` – allows us to give the embedded gateway a pointer to a BLOB which will then be returned to the browser.

So `myphoto_show` first specifies the `mime_header` and then second step passes a pointer to the BLOB. The gateway handles the rest. Getting the `mime_header` right is important.

```

create or replace procedure myphoto_show( vpk in integer, vshow in
varchar2 default null )
as
  cursor c1(vpk integer) is select * from myimages where pk = vpk;
  crec          myimages%ROWTYPE;
  NL_CHAR      constant char(1) := chr(10);
begin
  open c1(vpk);
  fetch c1 into crec;
  close c1;

  if vshow = 'THUMBNAIL'
  then
    owa_util.mime_header(crec.mm_photo_tnail.mimetype);
    wpg_docload.download_file(crec.mm_photo_tnail.source.localdata);
    return;
  end if;

  case crec.image_type
  when 'PHOTO'
  then
    owa_util.mime_header(crec.mm_photo.mimetype);
    wpg_docload.download_file(crec.mm_photo.source.localdata);
  when 'VIDEO'
  then
    owa_util.mime_header(crec.mm_video.mimetype);
    wpg_docload.download_file(crec.mm_video.source.localdata);
  when 'AUDIO'
  then
    owa_util.mime_header(crec.mm_audio.mimetype);
    wpg_docload.download_file(crec.mm_audio.source.localdata);

```

```

when 'DOCUMENT'
then
    http.prn('Content-type: ' || 'application/octet-stream' ||
NL_CHAR);
    http.prn('Content-Disposition: ' || 'attachment;' || ' filename="'
|| crec.mm_doc.source.srcname || '"' || NL_CHAR); -- attachment or
inline
    http.prn('Content-Transfer-Encoding: binary' || NL_CHAR);
    http.prn('Content-Length: ' ||
to_char(dbms_lob.getlength(crec.mm_doc.source.localdata)) || NL_CHAR);
    http.prn(NL_CHAR);
    wpg_docload.download_file(crec.mm_doc.source.localdata);
    else null;
end case;

end myphoto_show;
/
sho err

```

Now try this again:

<http://localhost:8890/mm/mytableC>

C. Load Multimedia Data into the Database

Now we will see how we can load an image into the database with a browser. The embedded gateway has a nice feature that when a file is pushed up from the browser to the server, it is stored in the `APACHE_OWS_CONTENT` table, which was defined in the initial configuration step using this command:

```
exec dbms_epg.set_dad_attribute( 'PICTION', 'document-table-name',
'APACHE_OWS_CONTENT');
```

We now create a procedure that displays a HTML form.

The `http.formopen` command generates a `<form>` tag. To load a file, this tag has to be correctly specified with two important notes:

1. We have to include the DAD name in the call to the database. This is an exception to the rule for HTML calls, but is required so that the embedded gateway knows which DAD should be used for loading the image into the database. To make it flexible, rather than hardcoding as `/mm/` we can use the `owa_util` to reference the current DAD:

```
owa_util.get_cgi_env('SCRIPT_NAME')
```

2. We need to tell the browser that on submission of the form, we are passing one or more files. If we don't include this line, the file content will not be sent; only the names will be sent by the browser. This is done using the command: `enctype="multipart/form-data"`

```
create or replace procedure ask_image
as
begin
  http.htmlopen;
  http.formopen( owa_util.get_cgi_env('SCRIPT_NAME') || '/' ||
'load_image', cattributes=>'enctype="multipart/form-data" ');

  http.tableopen( cattributes=>'style="border: black solid 1px;
padding: 10px; background-color: #FFFFFF;" ');
  http.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:center; font-weight:bold; background-color:#CCCCCC;" ');
  http.tabledata( htf.bold( 'Please Specify Image to Load' ) );
  http.tabledata( '<INPUT TYPE="FILE" NAME="MYFILE" SIZE="40"
MAXLENGTH="1000">' );
  http.tablerowclose;
  http.tableclose;
  http.nl;
  http.formsubmit( 'ACTION', 'Load Image' );
  http.formclose;
  http.htmlclose;
end ask_image;
/
sho err
```

Once the file is loaded into the database it needs to be processed. The name of the file is passed to the embedded gateway through a parameter called `MYFILE` (as found in the HTML command: `<INPUT TYPE="FILE" NAME="MYFILE" SIZE="40" MAXLENGTH="1000">`). The names have to match.

The directory the file came from is not passed to the database. Instead a randomly generated set of characters is passed up along with the filename

e.g. ABC123RRR/myfile.jpg

This string of characters will be unique and is used as the primary key to access the `apache_ows_content` table.

So we now create the procedure that queries the `apache_ows_content` table, retrieves the BLOB from this table, copies it into the `myimages` table, and processes it:

```
create or replace procedure load_image(myfile in varchar2, action in
varchar2 )
as

  cursor c1(vname varchar2) is select * from apache_ows_content where
name = vname;
  cursor c2(vpk integer) is select * from myimages where pk = vpk for
update;

  arec          apache_ows_content%ROWTYPE;
  crec          myimages%ROWTYPE;

begin
  open c1(myfile);
  fetch c1 into arec;
  close c1;
  delete from myimages where pk = 1000;
  insert into
myimages(pk,name,image_type,details,mm_photo_tnail,mm_photo) values
(1000,'My
Photo','PHOTO',null,ordsys.ordimage.init(),ordsys.ordimage.init());
  open c2(1000);
  fetch c2 into crec;
  close c2;

  dbms_lob.copy(crec.mm_photo.source.localdata, arec.blob_content,
dbms_lob.getlength( arec.blob_content ), 1, 1 );
  crec.mm_photo.setproperties;
  crec.mm_photo.processcopy('maxScale=100 100',crec.mm_photo_tnail);

  -- should really specify other values here like srclocation and
srcname
  update myimages set row = crec where pk = crec.pk;

  delete from apache_ows_content where name = myfile;
  commit;

  mytableC;

exception
  when others then
    htp.bold( 'Error adding to load_image:' || sqlerrm );
end load_image;
/
```

sho err

- Exercise: Attendees should enhance this program to include comments about the image.

Invoke the procedure as follows:

http://localhost:8890/mm/ask_image

D. Extract metadata (EXIF) from a photo

We will now write a procedure to take in metadata in the form of XML, and convert the values into tags that are readable and conform to most accepted standards. The program is generic enough to handle IPTC and Adobe XMP as well.

```
create or replace procedure view_metadata( vtg in varchar2, doc in
dbms_xmldom.DOMDocument, parm1 in out nocopy owa.vc_arr, parm2 in out
nocopy owa.vc_arr )
as

bfr          varchar2(32767);
nl           dbms_xmldom.DOMNodeList;
len1         number;
len2         number;
ctr          integer;
x           integer;
n           dbms_xmldom.DOMNode;
n2          dbms_xmldom.DOMNode;
e           dbms_xmldom.DOMELEMENT;
nnm         dbms_xmldom.DOMNamedNodeMap;
n_child     dbms_xmldom.DOMNode;
attrname    varchar2(100);
attrval     varchar2(100);
lasttag     varchar2(1000);
keytag      varchar2(1000);
tag         varchar2(1000);
vtag        varchar2(1000);
dest_offset integer;
src_offset  integer;
lang_context integer;
warning     varchar2(1000);

begin
  -- get all elements
  nl := dbms_xmldom.getElementsByTagName(doc, '*');
  len1 := dbms_xmldom.getLength(nl);
  ctr := 0;
  lasttag := NULL;
  -- loop through elements
  for j in 0..len1-1 loop
    n := dbms_xmldom.item(nl, j);
    e := dbms_xmldom.makeElement(n);
    n_child := dbms_xmldom.getFirstChild(n);
    vtag := dbms_xmldom.getTagname(e);
    x := instr(vtag, ':');
    if x > 0
    then
      tag := upper(substr(vtag,x+1)); -- tag x:y = y
    else
      tag := vtag;
    end if;
  end loop;
end;
```

```

if vtg = 'XMP'
then
  if tag in ('DESCRIPTION','DOCUMENTID', 'LI', 'SEQ', 'ALT',
'FORMAT' )
  then
    keytag := nvl(lasttag, trim(upper(vtag)));
  else
    keytag := tag;
  end if;

elsif vtg = 'IPTC'
then
  keytag := upper(tag);
  case keytag
  when 'IMAGE NAME' then keytag := '5_IMAGE NAME';
  when 'PRIORITY' then keytag := '10_PRIORITY';
  when 'CATEGORY' then keytag := '15_CATEGORY';
  when 'SUPPLEMENTAL CATEGORY' then keytag :=
'20_SUPPLEMENTAL CATEGORY';
  when 'LOCALE' then keytag := '22_LOCALE';
  when 'KEYWORD' then keytag := '25_KEYWORD';
  when 'SPECIAL INSTRUCTIONS' then keytag := '40_SPECIAL
INSTRUCTIONS';
  when 'CREATED DATE' then keytag := '55_CREATED DATE';
  when 'CREATED TIME' then keytag := '60_CREATED TIME';
  when 'ORIGINAL PROGRAM' then keytag := '65_ORIGINAL
PROGRAM';
  when 'BYLINE' then keytag := '80_BYLINE';
  when 'BYLINE TITLE' then keytag := '85_BYLINE TITLE';
  when 'CITY' then keytag := '90_CITY';
  when 'PROVINCE STATE' then keytag := '95_PROVINCE
STATE';
  when 'COUNTRY' then keytag := '100_COUNTRY';
  when 'COUNTRY' then keytag := '101_COUNTRY';
  when 'ORIGINAL TRANSMISSION REFERENCE' then keytag :=
'103_ORIGINAL TRANSMISSION REFERENCE';
  when 'HEADLINE' then keytag := '105_HEADLINE';
  when 'CREDIT' then keytag := '110_CREDIT';
  when 'SOURCE' then keytag := '115_SOURCE';
  when 'COPYRIGHT STRING' then keytag :=
'116_COPYRIGHT';
  when 'COPYRIGHT' then keytag :=
'116_COPYRIGHT';
  when 'CAPTION' then keytag := '120_CAPTION';
  when 'CAPTION WRITER' then keytag := '122_CAPTION
WRITER';
  when 'AUTHOR' then keytag := '115_SOURCE';
  when 'OTHER' then keytag := '230_OTHER';
  else null;
  end case;

else
  keytag := tag;

```

```

end if;

if keytag not in ('.')
then
begin
if lasttag = keytag
then
if parm2(ctr) is null
then
parm2(ctr) := trim(dbms_xmlDOM.getNodeValue(n_child));
else
parm2(ctr) := parm2(ctr) || ',' ||
trim(dbms_xmlDOM.getNodeValue(n_child));
end if;

else
ctr := ctr + 1;
if keytag = replace(keytag, 'PHOTOSHOP:', '')
then
parm1(ctr) := vtg || ':' || keytag;
else
parm1(ctr) := vtg || ':' || keytag || '.' ||
replace(keytag, 'PHOTOSHOP:', '');
end if;
parm2(ctr) := trim(dbms_xmlDOM.getNodeValue(n_child));
end if;
lasttag := keytag;

exception
when others then htp.p(sqlerrm);
end;
end if;

end loop;
end view_metadata;
/
sho err

create or replace procedure display_metadata( vpk in integer )
as
cursor c1(vpk integer) is select * from myimages where pk = vpk;
crec
XMLSequenceType;
meta_root varchar2(40);
parm1 owa.vc_arr;
parm2 owa.vc_arr;

begin
open c1(vpk);
fetch c1 into crec;
close c1;

```

```

case crec.image_type
when 'PHOTO'
then
    metav := crec.mm_photo.getMetadata('ALL');
    for i in 1..metav.count() loop
        meta_root := metav(i).getRootElement();
        CASE meta_root
            WHEN 'xmpMetadata' THEN view_metadata( 'XMP',
DBMS_XMLDOM.newDOMDocument(metav(i)), parm1, parm2 );
            WHEN 'iptcMetadata' THEN view_metadata( 'IPTC',
DBMS_XMLDOM.newDOMDocument(metav(i)), parm1, parm2 );
            WHEN 'exifMetadata' THEN view_metadata( 'EXIF',
DBMS_XMLDOM.newDOMDocument(metav(i)), parm1, parm2 );
            ELSE NULL;
        end case;
    end loop;
else null;
end case;

htp.htmlopen;
htp.tableopen( cattributes=>'style="border: black solid 1px;
padding: 10px; background-color: #FFFFFF;"' );
    htp.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:center; font-weight:bold; background-color:#CCCCCC;"' );
        htp.tabledata( htf.bold( 'XML Tag' ) );
        htp.tabledata( htf.bold( 'XML Value' ) );
    htp.tablerowclose;
if parm1.count > 0
then
    for j in parm1.first..parm1.last loop
        if mod(j,2) = 1
            then
                htp.tablerowopen( cattributes=>'valign="MIDDLE" style="text-
align:left; background-color:#F0F0F0;"' );
            else
                htp.tablerowopen;
            end if;
            htp.tabledata( parm1(j) );
            htp.tabledata( parm2(j) );
            htp.tablerowclose;
        end loop;
    end if;
htp.tableclose;

exception
    when others then htp.bold('Error in display_metadata:' || sqlerrm );
end display_metadata;
/
sho err

http://localhost:8890/mm/display_metadata?vpk=100

```

E. Advanced topics

By now the attendees should have a good appreciation for how easy it is to write mod PL/SQL programs that can use HTTP GET and POST commands to call procedures and retrieve data.

Security is very important for many types of applications. The URL should be protected.

One security hole which has been discussed by a number of folks at length (over the past 15yrs) is the dangers of using dynamic SQL. PL/SQL encourages the use of bind variable and these should be used.

e.g. `execute immediate 'select mycol from mytable where pk=' || mypk;`
is not good programming. It is not efficient and is open to abuse.

Instead, this statement should be rewritten as:

```
execute immediate 'select mycol from mytable where pk=:bindvar' using mypk;
```

Better yet, avoid the use of dynamic SQL altogether and use PL/SQL cursors instead. These are faster, are compiled into the program, and run efficiently.

```
cursor c1(mypk integer) is select mycol from mytable where pk = mypk;

open c1(mypk);
...
```

Web programming languages like PHP, Java and Perl inherently encourage the use of dynamic SQL because of limitations in their design for accessing Oracle databases (they are built generically). This is one of the strengths of mod PL/SQL: it encourages efficient and secure web page programming (when used right).

So as a simple way to secure a web program, follow the rule of never, ever using dynamic SQL without bind variables.

The use of bind variables is just one of many steps in securing the PL/SQL Gateway application. As shown in Part A, the embedded gateway itself can be protected by controlling which PL/SQL programs can and cannot be invoked from the URL.

One of the most important steps for securing a PL/SQL Gateway application is to ensure that all primary key (PK) values are encrypted. This means that they cannot be tampered with in the URL

e.g. in the previous example I had:

http://localhost:8890/mm/display_metadata?vpk=100

by changing `vpk=100` to `vpk=200`, I can change the primary key that is passed in and access rows that possibly I might not be allowed to access.

To prevent this, the value should be encrypted. With mod PL/SQL the important lesson you will learn is that when you design your application, all tables have *integer* primary keys. The following program shows how we can easily encrypt the primary key, and pass its HEX values between screens:

```
create or replace function encrypt_pk( input_string in varchar2 )
  return varchar2
as
  raw_input          raw(128);
```

```

    raw_key          raw(128);
    encrypted_raw    raw(2048);
    maxlength        integer := 16;
    enc_key          constant char(8) := 'ABCDEFGH';
begin
    raw_key := utl_raw.cast_to_raw(enc_key);
    raw_input :=
utl_raw.cast_to_raw(substr(rpad(nvl(upper(input_string),' '),maxlengtht
h,' '),1,maxlength));
    dbms_obfuscation_toolkit.desencrypt(input => raw_input, key =>
raw_key, encrypted_data => encrypted_raw );
    return( rawtohex(encrypted_raw) );

exception
when others
then
    return( NULL );
end encrypt_pk;
/
sho err

create or replace function decrypt_pk( input_string_hex in varchar2 )
return varchar2
as

    raw_key          raw(128);
    encrypted_raw    raw(2048);
    decrypted_raw    raw(2048);
    maxlength        integer := 16;
    enc_key          constant char(8) := 'ABCDEFGH';
begin
    raw_key := utl_raw.cast_to_raw(enc_key);
    encrypted_raw := hextoraw( substr(input_string_hex,1,maxlength) );
    dbms_obfuscation_toolkit.desdecrypt(input => encrypted_raw, key =>
raw_key, decrypted_data => decrypted_raw);

    return( utl_raw.cast_to_varchar2(decrypted_raw) );

exception
when others
then
    return( NULL );
end decrypt_pk;
/
sho err

select encrypt_pk('100') from dual;
select decrypt_pk('11893890AEB6E62625A788356848FCC9') from dual;

```

Exercise: Attendees should modify the program in Section D to use the encryption routine above to encrypt and decrypt the primary key value.

The challenge when using encryption is the management of the `enc_key`, which is the key used for

encrypting. It needs to be kept secure. Unauthorized people can use it to decrypt the primary key from the URL, and/or replace it with or a newly encrypted value.
