Media Upload and Retrieval Web Sample Applications

This chapter describes the development of several types of media upload and retrieval applications using Oracle interMedia ("interMedia") object types. Section 3.1 describes the development of the interMedia PL/SQL photo album sample Web application that uses the PL/SQL Gateway and PL/SQL Web Toolkit for Oracle Application Server and Oracle Database. Section 3.2 describes the development of the following interMedia photo album Web applications that use the interMedia image object type:

- Java servlet application that uses Oracle interMedia Java Classes for Servlets and JSP (see Section 3.2.1)
- JavaServer Pages (JSP) application that uses Oracle interMedia Java Classes for Servlets and JSP (see Section 3.2.2)
- Active Server Pages (ASP)/Visual Basic (VB) application for the Microsoft Internet Information Server (IIS) Web Server (see Section 3.2.3)

Note:
The sample applications described in Section 3.2 differ from the PL/SQL photo album sample application described in Section 3.1.

Section 3.3 describes the interMedia Code Wizard application for the PL/SQL Gateway that uses the interMedia image, audio, video, and heterogeneous media object types. This chapter assumes the following:

- You are familiar with:
  - Developing PL/SQL applications using the PL/SQL Gateway and PL/SQL Web Toolkit
  - Developing Java-based Web applications using JDBC, creating Java source code, compiling it into byte code (.class) files, and deploying class files into respective servlet containers required by Oracle HTTP Server for Oracle Application Server and Oracle Database
  - Developing ASP/VB scripts for the Microsoft IIS Web Server
- You have already installed and configured the following sample applications:
  - Oracle interMedia PL/SQL Web Toolkit Photo Album application
  - Oracle interMedia Java Servlet Photo Album application
  - Oracle interMedia JSP Photo Album application
  - Oracle interMedia ASP/VBSScript Photo Album application
  - Oracle interMedia Code Wizard for the PL/SQL Gateway application
See the README.txt file for each respective sample application for installation and configuration information.

Oracle interMedia PL/SQL Photo Album Sample Application

The interMedia PL/SQL Web Toolkit Photo Album sample application demonstrates how to perform the following operations:

- Use the interMedia image object type to upload, retrieve, and process media data stored in Oracle Database.
- Combine the image metadata methods of interMedia with the XML document management capabilities of Oracle XML DB and the full-text indexing and search features of Oracle Text to create a solution that can extract, store, and search metadata that is embedded in binary image files.
- Collect new metadata from a user, format the metadata into an XML document, and store the document in the binary image.

When installed, the photo album application creates a number of schema objects that are important to the following discussion. These objects include the photos table, which is defined by the following CREATE TABLE statement:
CREATE TABLE photos( id NUMBER PRIMARY KEY,  
description VARCHAR2(40) NOT NULL,  
metaORDimage XMLTYPE,  
metaEXIF XMLTYPE,  
metaIPTC XMLTYPE,  
metaXMP XMLTYPE,  
image ORDSYS.ORDIMAGE,  
thumb ORDSYS.ORDIMAGE )

-- storage images with 32K chunk
--
LOB(image.source.localdata) STORE AS (chunk 32k)
--
-- but the thumbnails with only 16k
--
LOB(thumb.source.localdata) STORE AS (chunk 16k)
--
-- and bind the XMLType columns to the interMedia metadata schemas
XMLType COLUMN metaORDImage
XMLSCHEMA "http://xmlns.oracle.com/ord/meta/ordimage"
ELEMENT "ordImageAttributes"
XMLType COLUMN metaEXIF
XMLSCHEMA "http://xmlns.oracle.com/ord/meta/exif"
ELEMENT "exifMetadata"
XMLType COLUMN metaIPTC
XMLSCHEMA "http://xmlns.oracle.com/ord/meta/iptc"
ELEMENT "iptcMetadata"
XMLType COLUMN metaXMP
XMLSCHEMA "http://xmlns.oracle.com/ord/meta/xmp"
ELEMENT "xmpMetadata";

The data types for the image and thumb columns are defined as interMedia image object types. These columns are used to store the full-size images and the generated thumbnail images, respectively. The LOB storage clauses indicate that storage for the full-size images is to be allocated in 32-kilobyte chunks, thus enabling the fastest reading and writing of the image data. Similarly, storage for the thumbnail images is allocated in 16-kilobyte chunks, enabling fast access and efficient storage. In addition, using a smaller chunk size reduces the allocation of empty space.

The table also defines four columns of type XMLType to store XML documents that contain four different kinds of image metadata. Each column is bound to a specific interMedia metadata schema. Each metadata schema defines precisely the data model of the metadata document. These schemas are registered with Oracle XML DB when the database is created. The column definitions specify that the database uses structured storage to manage the XML metadata documents. Some advantages of using structured storage to manage XML include optimized memory management, reduced storage requirements, B-tree indexing, and in-place updates. For more information about XML DB, see Oracle XML DB Developer's Guide.

When installed, the photo album application also creates other schema objects. These schema objects include two types of indexes that accelerate metadata searches: a CONTEXT text index and a CTXXPATH text index. The CONTEXT type is a text index over all columns that contain descriptive information about the image. These columns include photos.description, which is a VARCHAR2 data type, and the following four XMLType columns: photos.metaIPTC, photos.metaEXIF, photos.metaXMP, and photos.metaORDimage. The CONTEXT text index is used to accelerate metadata searches by implementing the photo album search feature that allows users to search for photographs by keyword or phrase.
The **CONTEXT** text index is created by the following statements. (This example assumes that the photo album application has been installed in the SCOTT schema.)

```sql
-- create preference PA_CTXIDX
ctx_ddl.create_preference('SCOTT.PA_CTXIDX', 'MULTI_COLUMN_DATASTORE');

-- create a multi-column datastore
ctxcols := 'description, '||
          'SCOTT.photo_album.getClob(METAIPTC), '||
          'SCOTT.photo_album.getClob(METAEXIF), '||
          'SCOTT.photo_album.getClob(METAXMP), '||
          'SCOTT.photo_album.getClob(METAORDIMAGE)';
ctx_ddl.set_attribute( ctxpref, 'COLUMNS', ctxcols );

-- create the CONTEXT text index
create index pa_ctx_idx on photos(description)
indextype is ctxsys.context
parameters ( 'DATASTORE SCOTT.PA_CTXIDX' );
```

For more information about creating and using text indexing, see *Oracle Text Application Developer's Guide*. The **CTXXPATH** type text index is used to accelerate metadata searches by allowing users to search only certain types of image metadata as well as limit the search to specific portions of an XML document. For example, the following statements create three text indexes of type **CTXXPATH** to speed up existsNode() queries on columns of **XMLType**:

```sql
create index pa_path_iptc_idx  on photos( metaIptc )
indextype is ctxsys.ctxxpath;

create index pa_path_exif_idx  on photos( metaExif )
indextype is ctxsys.ctxxpath;

create index pa_path_xmp_idx  on photos( metaXMP )
indextype is ctxsys.ctxxpath;
```

For more information about creating and using **CTXXPATH** indexes, see *Oracle Text Application Developer's Guide*.

During the installation, as prescribed by the PL/SQL Gateway, a document upload table is defined by the following CREATE TABLE statement:

```sql
CREATE TABLE PHOTOS_UPLOAD(name VARCHAR2(256) UNIQUE NOT NULL,
                            mime_type VARCHAR2(128),
                            doc_size NUMBER,
                            dad_charset VARCHAR2(128),
                            last_updated DATE,
                            content_type VARCHAR2(128),
                            blob_content BLOB);
```
Each image uploaded using the PL/SQL Gateway is stored in the PHOTO_UPLOAD table. An upload procedure (insert_new_photo) automatically moves the uploaded image from the specified PHOTO_UPLOAD table to the photo album applications table called photos.

The PL/SQL procedures view_upload_form, print_upload_form, and insert_new_photo are the primary application components that implement the Upload photo page. Together, view_upload_form and print_upload_form create the HTML page that is displayed. The page contains a form tag, a portion of which is shown in Example 3-5. The target of the form is PHOTO_ALBUM.INSERT_NEW_PHOTO. Example 3-5 contains some relevant lines of code in the print_upload_form procedure.

Example 3-5 Procedure print_upload_form

```html
<form action="PHOTO_ALBUM.INSERT_NEW_PHOTO"
method="post"
enctype="multipart/form-data">
database.
```
Procedure **insert_new_photo**

Procedure **insert_new_photo** receives the form, processes the inputs, and stores the new image in the database.

First, the **insert_new_photo** procedure checks that a file name was entered into the upload form. The image size, MIME type, and BLOB locator for the image content are selected from the document upload table, and the size is checked to ensure that the image is not of zero length. If the **description** field is blank, a description is created using the file name.

Next, the ORDSYS.ORDIMAGE.INIT( ) function is called to initialize the **thumb** and **image** ORDImage object type columns with an empty BLOB for the new row to be stored in the **photos** table. A SQL SELECT FOR UPDATE statement fetches the newly initialized thumbnail image and full-size image object type columns for updating. A DBMS_LOB.COPY operation loads the image from the upload table into the **image** ORDImage object type column.

The ORDImage object method setProperties( ) reads the image and sets the image object attributes. Because some browsers cannot display some image formats inline, in this sample application, BMP formatted images are converted to a JPEG image format (for images with more than 8 bits of color), or a GIF image format (for images with less than 9 bits of color) by calling the get_preferred_format function. A processCopy( ) operation is performed on the full-size image to create the thumbnail image.

The ORDImage object getMetadata( ) method is called to extract all supported types of image metadata. The root element of each XML document in the return vector is examined to discover the metadata type so that the documents can be stored in the correct columns.

Then, a SQL UPDATE statement stores the full-size image, the thumbnail image, and the image metadata documents in the database. Procedure **sync_indexes** is called to force an update of the text indexes. Finally, the form data input is deleted from the document upload table. A success message is returned to the browser, and the browser is redirected to the View album page.

Example 3-6 contains some relevant lines of code in the **insert_new_photo** procedure.

```sql
--
-- Make sure a file name has been provided. If not, display an error
-- message, then re-display the form.
--
IF new_photo IS NULL OR LENGTH( new_photo ) = 0
THEN
    print_page_header;
    print_error( 'Please supply a file name.' );
    print_upload_form;
    print_page_trailer( TRUE );
    return;
END IF;

--
-- Get the length, MIME type and the BLOB of the new photo from the
-- upload table.
--
SELECT doc_size,
       mime_type,
       blob_content
INTO   upload_size,
       upload_mime_type,
       upload_blob
FROM photos_upload
WHERE name = new_photo;
```
-- Make sure we have a valid file.
--
IF upload_size = 0
THEN
  print_page_header;
  print_heading( 'Error message' );
  htp.print( '<hr size="-1"><p>Please supply a valid image file.</p>' );
  print_upload_form;
  print_page_trailer( TRUE );
  return;
END IF;

-- If the description is blank, then use the file name.
--
IF c_description IS NULL
THEN
  c_description := new_photo;
  pos := INSTR( c_description, '/', -1 );
  IF pos > 0
    THEN
      c_description := SUBSTR( c_description, pos + 1 );
    END IF;
  c_description := SUBSTR( 'Image from file: ' ||
                         c_description || '.', 1, 40 );
END IF;

-- Insert a new row into the table, returning the newly allocated sequence number.
INSERT INTO photos ( id, description, metaExif, metaIPTC, metaXMP,
                      image, thumb )
VALUES ( photos_sequence.nextval, c_description, NULL, NULL, NULL,
          ORDSYS.ORDIMAGE.INIT(), ORDSYS.ORDIMAGE.INIT() )
RETURN id
INTO new_id;

-- Fetch the newly initialized full-size and thumb-nail image objects.
--
SELECT image,
       thumb
INTO new_image,
       new_thumb
FROM photos
WHERE id = new_id
FOR UPDATE;

-- Load the photo from the upload table into the image object.
--
DBMS_LOB.COPY( new_image.source.localData, upload_blob, upload_size );
new_image.setLocal();
--
-- Set the properties. If the image format is not recognized, then
-- the exception handler will set the MIME type and length from the
-- upload table.
--
BEGIN
  new_image.setProperty();
EXCEPTION
    WHEN OTHERS THEN
      new_image.setContentLength(upload_size);
      new_image.setMimeType(upload_mime_type);
END;
--
-- Some image formats are supported by interMedia but may not be able
-- to be displayed in-line by a browser. The BMP format is one example.
-- Convert the image to a GIF or JPEG based on number of colors in the
-- image.
--
IF new_image.getContentFormat IS NOT NULL AND
  ( new_image.getMimeType = 'image/bmp' OR
    new_image.getContentType = 'image/x-bmp' )
THEN
  BEGIN
    new_image.process('fileFormat=' ||
        get_preferred_format( new_image.getContentFormat() ));
  EXCEPTION
    WHEN OTHERS THEN
      NULL;
  END;
END IF;
--
-- Try to copy the full-size image and process it to create the thumbnail.
-- This may not be possible if the image format is not recognized.
--
BEGIN
  new_image.processCopy( thumb_scale, new_thumb );
EXCEPTION
    WHEN OTHERS THEN
      new_thumb.deleteContent();
      new_thumb.setContentLength(0);
END;
--
-- Fetch the metadata and sort the results
--
BEGIN
  metav := new_image.getMetadata( 'ALL' );
  FOR i IN 1..metav.count() LOOP
    meta_root := metav(i).getRootElement();
    CASE meta_root
      WHEN 'ordImageAttributes' THEN xmlORD := metav(i);
      WHEN 'xmpMetadata' THEN xmlXMP := metav(i);
      WHEN 'iptcMetadata' THEN xmlIPTC := metav(i);
    END CASE;
  END LOOP;
END;
WHEN 'exifMetadata' THEN xmlEXIF := metav(i);
ELSE NULL;
END CASE;
END LOOP;
EXCEPTION
WHEN OTHERS THEN
NULL;
END;

-- Update the full-size and thumb-nail images in the database.
-- Update metadata columns
UPDATE photos
SET image = new_image,
thumb = new_thumb,
metaORDImage = xmlORD,
metaEXIF = xmlEXIF,
metaIPTC = xmlIPTC,
metaXMP = xmlXMP
WHERE id = new_id;

-- Update the text indexes
sync_indexes;

-- Delete the row from the upload table.
DELETE FROM photos_upload WHERE name = new_photo;
COMMIT;

-- Redirect browser to display full album.
print_page_header(
'\n<meta http-equiv="refresh" content="2;url=PHOTO_ALBUM.VIEW_ALBUM">'
);
print_heading( 'Photo successfully uploaded into photo album' );
Browsing the Photo Album

The home page for the photo album application, View album, displays the contents of the photo album as thumbnail images in four-column format. Each thumbnail image is also a link to the View entry page. When you click a thumbnail image link, the application displays the full-size image on a View entry page. Included under each thumbnail image on the View album page is the image description that was entered when the image was uploaded to the album. The description is also a link to the View metadata page where all the metadata for this photograph can be examined.

Near the top of the View album page, there is a text entry field (in the shape of a rectangular box) that accepts user input for a full text search through all the photo album metadata. The Search button to the right of the text field initiates the search. The search results are displayed on the Search album page, which is discussed in Section 3.1.2.3.

At the top of the View album page, there is a navigation bar, which includes links to other photo album pages. From the View album page, you can navigate to the Search metadata page or the Upload photo page. These pages are described in Section 3.1.2.7 and Section 3.1.2.2, respectively.

Figure 3-1 shows the View album page for an album that contains five images.
The PL/SQL procedures `view_album`, `print_album`, `print_image_link`, and `deliver_media` are the primary application components that implement the View album page. The `view_album` procedure is a public procedure that takes a single optional argument. By default, the argument has a `NULL` value. Or, it can have the value of the string entered in the text entry field on the Search album page. When the search argument is `NULL`, the SELECT statement retrieves the `id`, `description`, and `thumb` columns for all
entries in the `photos` table. When the search string is not `NULL`, the SELECT statement uses the CONTAINS operator to restrict the result set to only images with metadata that matches the search string. (Section 3.1 describes how the application creates a multicolumn text index over the four XMLType columns `PHOTOS.METAIPTC`, `PHOTOS.METAEXIF`, `PHOTOS.METAXMP`, and `PHOTOS.METAORDIMAGE` as well as the `PHOTOS.DESCRIPTION` column.)

Example 3-1 contains some relevant lines of code in the `view_album` procedure.

Example 3-1 Procedure `view_album`

```
--
-- no search criteria so fetch all entries
--
IF search IS NULL THEN
  OPEN album_cur FOR
    SELECT id, description, thumb
    FROM photos
    ORDER BY id;
  print_album( album_cur, 'The photo album is empty.' );
  CLOSE album_cur;
ELSE
  -- -- use the full-text index to select entries matching the search criteria
  --
  OPEN album_cur FOR
    SELECT id, description, thumb
    FROM photos
    WHERE CONTAINS( description, trim(search) ) > 0
    ORDER BY id;
  print_album( album_cur, 'No photos were found.' );
  CLOSE album_cur;
END IF;
```

The SELECT statement is bound to the cursor variable `album_cur` and passed to the procedure `print_album`, which creates the HTML output.

The `print_album` procedure uses the HTP and HTF packages from the PL/SQL Web Toolkit to create the HTML tags that format the output into a four-column table. Each cell in the table contains two links or anchor tags. The first link is to the View entry page, which displays the full-size version of the image. This anchor is implemented by `PHOTO_ALBUM.VIEW_ENTRY`, and passes `entry_id` as a query string input argument.

If an image is in a format that InterMedia does not support, the application will not be able to create a thumbnail version of the image. In this case, the content of the anchor link is the text view image. Example 3-2 contains some relevant lines of code in the `print_album` procedure.

Example 3-2 Procedure `print_album`

```
-- escape the description text
sc_description := htf.escape_sc( entry.description );
--
```
-- Display the thumb-nail image as an anchor tag which can be used
-- to display the full-size image. If the image format isn't
-- supported by interMedia, then a thumb-nail wouldn't have been
-- produced when the image was uploaded, so use the text '[view
-- image]' instead of the thumb-nail.
--
htp.print( '<td headers="c' || colIdx || '" align="center" >
  <a href="PHOTO_ALBUM.VIEW_ENTRY?entry_id=' ||
    entry.id || '">
    IF entry.thumb.contentLength > 0
    THEN
      print_image_link( 'thumb', entry.id, sc_description,
        entry.thumb.height, entry.thumb.width );
    ELSE
      htp.prn( '[view image]' );
    END IF;
  htp.print( '</a>' );
-- Create link to the metadata
  htp.prn('<br>);
  htp.anchor( curl=>'PHOTO_ALBUM.VIEW_METADATA?entry_id=' || entry.id,
    ctext=>sc_description );
  htp.prn('</td>');</td>
-- add height and width to tag if non zero
IF height > 0 AND width > 0 THEN
  attributes := attributes || ' height=' || height || ' width=' || width;
END IF;

-- create an alt text if none given
IF alt IS NULL THEN
  IF type = 'thumb' THEN
    alt2 := 'thumb-nail image ';
  ELSE
    alt2 := 'full-size image ';  
  END IF;
  alt2 := alt2 || 'for album entry ' || entry_id;
ELSE
  alt2 := alt;
END IF;

htp.img( curl=>'PHOTO_ALBUM.DELIVER_MEDIA?media=' || type ||
  ...

The procedure print_image_link uses the height and width arguments to populate the height and width attributes of the <img> tag. The description argument is used to create text for the alt attribute. If the description argument is empty, a default string is constructed. Finally, the src attribute is set to the URL PHOTO_ALBUM.DELIVER_MEDIA with two query string arguments, media and entry_id. The media argument controls whether the thumbnail or full-size version of the image is delivered. The entry_id argument identifies the image to be delivered.

Example 3-3 contains some relevant lines of code in the print_image_link procedure.

Example 3-3 Procedure print_image_link

-- add height and width to tag if non zero
IF height > 0 AND width > 0 THEN
  attributes := attributes || ' height=' || height || ' width=' || width;
END IF;

-- create an alt text if none given
IF alt IS NULL THEN
  IF type = 'thumb' THEN
    alt2 := 'thumb-nail image ';
  ELSE
    alt2 := 'full-size image ';  
  END IF;
  alt2 := alt2 || 'for album entry ' || entry_id;
ELSE
  alt2 := alt;
END IF;

htp.img( curl=>'PHOTO_ALBUM.DELIVER_MEDIA?media=' || type ||
  'entry_id=' || entry.id | |
  'height=' || height || ' width=' || width
  'alt=' || alt2
  media=' || media
  entry_id=' || entry.id
  entry');
The procedure `deliver_media` fetches the image content from the database. The `If-Modified-Since` HTTP request header is compared to the last modification time of the image. If the image has not been modified, a response is sent that the browser can display the image from its cache. Otherwise, the image MIME type and last modified time are sent to the Web server, along with the image content.

**Example 3-4** contains some relevant lines of code in the `deliver_media` procedure.

**Example 3-4 Procedure deliver_media**

```sql
--
-- Fetch the thumb-nail or full-size image from the database.
--
IF media = 'thumb'
THEN
    SELECT thumb INTO local_image FROM photos WHERE id = entry_id;
ELSE
    SELECT image INTO local_image FROM photos WHERE id = entry_id;
END IF;

--
-- Check update time if browser sent If-Modified-Since header
--
IF ordplsgwyutil.cache_is_valid( local_image.getUpdateTime() )
THEN
    owa_util.status_line( ordplsgwyutil.http_status_not_modified );
    RETURN;
END IF;

--
-- Set the MIME type and deliver the image to the browser.
--
owa_util.mime_header( local_image.mimeType, FALSE );
ordplsgwyutil.set_last_modified( local_image.getUpdateTime() );
owa_util.http_header_close();

IF owa_util.get_cgi_env( 'REQUEST_METHOD' ) <> 'HEAD' THEN
    wpg_docload.download_file( local_image.source.localData );
END IF;
```
Adding Images to the Photo Album

The **Upload photo** page is used to add new images to the photo album. The page displays a form with two text entry fields. In the **Description:** field, you can optionally enter a word or short phrase that describes the image. In the **File name:** field, enter the name of the image file or click **Browse...** to locate the image file to be uploaded. The **Upload photo** button under the **File name:** field starts the upload operation. When the image is successfully uploaded, the **View album** page appears. From that page, you can display the contents of the photo album, as described in Section 3.1.2.1.

At the top of the **Upload photo** page, there is a navigation bar, which includes links to other photo album pages. From the **Upload photo** page, you can return to the **View album** page or select the **Search metadata** page. These pages are described in Section 3.1.2.1 and Section 3.1.2.7, respectively.

**Figure 3-2** shows an **Upload photo** page with all the entry fields completed.
The PL/SQL procedures `view_upload_form`, `print_upload_form`, and `insert_new_photo` are the primary application components that implement the Upload photo page. Together, `view_upload_form` and `print_upload_form` create the HTML page that is displayed. The page contains a form tag, a portion of which is shown in Example 3-5. The target of the form is `PHOTO_ALBUM.INSERT_NEW_PHOTO`. Example 3-5 contains some relevant lines of code in the `print_upload_form` procedure.

Example 3-5 Procedure `print_upload_form`

```
<form action="PHOTO_ALBUM.INSERT_NEW_PHOTO" method="post">
```
Procedure `insert_new_photo` receives the form, processes the inputs, and stores the new image in the database.

First, the `insert_new_photo` procedure checks that a file name was entered into the upload form. The image size, MIME type, and BLOB locator for the image content are selected from the document upload table, and the size is checked to ensure that the image is not of zero length. If the `description` field is blank, a description is created using the file name.

Next, the ORDSYS.ORDIMAGE.INIT( ) function is called to initialize the `thumb` and `image` ORDImage object type columns with an empty BLOB for the new row to be stored in the `photos` table. A SQL SELECT FOR UPDATE statement fetches the newly initialized thumbnail image and full-size image object type columns for updating. A DBMS_LOB.COPY operation loads the image from the upload table into the `image` ORDImage object type column.

The ORDImage object method `setProperties( )` reads the image and sets the image object attributes. Because some browsers cannot display some image formats inline, in this sample application, BMP formatted images are converted to a JPEG image format (for images with more than 8 bits of color), or a GIF image format (for images with less than 9 bits of color) by calling the `get_preferred_format` function. A processCopy( ) operation is performed on the full-size image to create the thumbnail image.

The ORDImage object `getMetadata( )` method is called to extract all supported types of image metadata. The root element of each XML document in the return vector is examined to discover the metadata type so that the documents can be stored in the correct columns.

Then, a SQL UPDATE statement stores the full-size image, the thumbnail image, and the image metadata documents in the database. Procedure `sync_indexes` is called to force an update of the text indexes. Finally, the form data input is deleted from the document upload table. A success message is returned to the browser, and the browser is redirected to the View album page.

Example 3-6 contains some relevant lines of code in the `insert_new_photo` procedure.

Example 3-6 Procedure `insert_new_photo`

```sql
-- -- Make sure a file name has been provided. If not, display an error -- message, then re-display the form. -- IF new_photo IS NULL OR LENGTH( new_photo ) = 0 THEN
  print_page_header;
  print_error( 'Please supply a file name.' );
  print_upload_form;
  print_page_trailer( TRUE );
  return;
END IF;

-- -- Get the length, MIME type and the BLOB of the new photo from the -- upload table. --
SELECT doc_size,
       mime_type,
       blob_content
INTO   upload_size,
       upload_mime_type,
       upload_blob
```
FROM photos_upload
WHERE name = new_photo;

--
-- Make sure we have a valid file.
--
IF upload_size = 0
THEN
    print_page_header;
    print_heading( 'Error message' );
    htp.print( '<hr size="-1"><p>Please supply a valid image file.</p>' );
    print_upload_form;
    print_page_trailer( TRUE );
    return;
END IF;

--
-- If the description is blank, then use the file name.
--
IF c_description IS NULL
THEN
    c_description := new_photo;
    pos := INSTR( c_description, '/', -1 );
    IF pos > 0
    THEN
        c_description := SUBSTR( c_description, pos + 1 );
    END IF;
    c_description := SUBSTR( 'Image from file: ' ||
        c_description || '.', 1, 40 );
END IF;

--
-- Insert a new row into the table, returning the newly allocated sequence
-- number.
INSERT INTO photos ( id, description, metaExif, metaIPTC, metaXMP, image, thumb )
VALUES ( photos_sequence.nextval, c_description, NULL, NULL, NULL, ORDSYS.ORDIMAGE.INIT(), ORDSYS.ORDIMAGE.INIT() )
RETURN id
INTO new_id;

--
-- Fetch the newly initialized full-size and thumb-nail image objects.
--
SELECT image,
    thumb
INTO new_image,
    new_thumb
FROM photos
WHERE id = new_id
FOR UPDATE;

--
-- Load the photo from the upload table into the image object.
--
DBMS_LOB.COPY( new_image.source.localData, upload_blob, upload_size );
new_image.setLocal();
BEGIN
    new_image.setProperties();
EXCEPTION
    WHEN OTHERS THEN
        new_image.contentLength := upload_size;
        new_image.mimeType := upload_mime_type;
END;

--
-- Some image formats are supported by interMedia but may not be able
-- to be displayed in-line by a browser. The BMP format is one example.
-- Convert the image to a GIF or JPEG based on number of colors in the
-- image.
--
IF new_image.contentFormat IS NOT NULL AND
    ( new_image.mimeType = 'image/bmp' OR
      new_image.mimeType = 'image/x-bmp' )
THEN
    BEGIN
        new_image.process(
            'fileFormat=' ||
            get_preferred_format( new_image.contentFormat ) );
        EXCEPTION
            WHEN OTHERS THEN
                NULL;
        END
    END IF;

--
-- Try to copy the full-size image and process it to create the thumb- 
-- nail.
-- This may not be possible if the image format is not recognized.
--
BEGIN
    new_image.processCopy( thumb_scale, new_thumb );
EXCEPTION
    WHEN OTHERS THEN
        new_thumb.deleteContent();
        new_thumb.contentLength := 0;
    END;
END;

--
-- fetch the metadata and sort the results
--
BEGIN
    metav := new_image.getMetadata( 'ALL' );
    FOR i IN 1..metav.count() LOOP
        meta_root := metav(i).getRootElement();
        CASE meta_root
            WHEN 'ordImageAttributes' THEN xmlORD := metav(i);
            WHEN 'xmpMetadata'  THEN xmlXMP  := metav(i);
            WHEN 'iptcMetadata' THEN xmlIPTC := metav(i);
        END CASE;
    END LOOP;
END;
WHEN 'exifMetadata' THEN xmlEXIF := metav(i);
ELSE NULL;
END CASE;
END LOOP;
EXCEPTION
WHEN OTHERS THEN
  NULL;
END;

-- Update the full-size and thumb-nail images in the database.
-- Update metadata columns
--
UPDATE photos
SET image = new_image,
    thumb = new_thumb,
    metaORDImage = xmlORD,
    metaEXIF = xmlEXIF,
    metaIPTC = xmlIPTC,
    metaXMP = xmlXMP
WHERE id = new_id;

-- -- update the text indexes
-- sync_indexes;

-- Delete the row from the upload table.
--
DELETE FROM photos_upload WHERE name = new_photo;
COMMIT;

-- Redirect browser to display full album.
--
print_page_header(
    '<meta http-equiv="refresh" content="2;url=PHOTO_ALBUM.VIEW_ALBUM">'
);
print_heading( 'Photo successfully uploaded into photo album' );
Searching for Images by Keyword or Phrase

You can use the View album and Search album pages to perform a keyword or phrase search of the metadata stored in the photo album. On either of these pages, enter the keyword or phrase in the Full text search: text entry field and click Search. The photo album application uses the CONTEXT text index to locate images that have metadata containing the text you entered. If the search is successful, the thumbnail versions of the matching images are displayed in a four-column table. Select the thumbnail image to view the full-size version, or select the description link below the thumbnail image to view the metadata for the image. If the search fails, the message "No photos were found" is displayed.

At the top of the Search album page, there is a navigation bar, which includes links to other photo album pages. From the Search album page, you can return to the View album page or select the Search metadata or Upload photo pages. These pages are described in Section 3.1.2.1, Section 3.1.2.7, and Section 3.1.2.2, respectively.

Figure 3-3 shows a Search album page that contains the results of a successful search operation.

Figure 3-3 Search album Page Showing Results
Full text searching of the photo album is implemented by the `view_album` and `print_album` procedures. See Section 3.1.2.1 for a discussion of these procedures.