Final Project
Instructions for Instructors

The final project is divided into 2 parts. You can use them in various ways depending on how much time you wish to devote to this project or on the technical level of your students. Part one is all about handling data in normal user tables and part two is concentrating more on database object administration, so they are quite different. It is best if all students do both parts, either individually or you can split them into teams.

Project Setup: The Data

This project will use a case study called TRAVELER ASSISTANCE. A set of database tables is used to provide information and manage the world travelers' requests. Information is stored about all the countries in the world and their languages, currencies, demographics and the region in which they are located. This database will assist travelers to obtain specific information about the countries to which they wish to travel.

General Programming Guidelines:

- Code your SQL statements to query the data regardless of whether it is stored in upper or lower case (use the LOWER and/or UPPER functions).
- Include an exception handler to handle NO_DATA_FOUND and other relevant exceptions in all your procedures.

The Assignment and Deliverables:

Create the following programs:

- Part 1: traveler_assistance_package
- Part 2: traveler_admin_package

Part 1: Provide Basic Information to Travelers

Create a package called *traveler_assistance_package* that will contain the following seven procedures. Make all procedures public. Comment your procedures to explain their purpose and functionality.

Two procedures in this package (*countries_in_same_region* and *country_languages*) return their fetched data back to the calling environment as an OUT parameter which is an associative array (ie an INDEX BY table of records). The last two procedures (*print_region_array* and *print_language_array*) will accept and display the returned arrays.

1. Create a procedure called *country_demographics* to display specific information about a country.
   - Pass COUNTRY_NAME as an IN parameter. Display COUNTRY_NAME, LOCATION, CAPITOL, POPULATION, AIRPORTS, CLIMATE. Use a user-defined record structure for the INTO clause of your select statement. Raise an exception if the country does not exist.

   Hints:
   - In order to populate the record in the select statement without specifying the record structure components, the record structure must be identical to the column list on the select statement.

   Incorporated Topics: Declare global user-defined record TYPE in the package. Populate record structures in a select statement.

   Tables Used: WF_COUNTRIES
   To Test:
   ```
   BEGIN
     traveler_assistance_package.country_demographics('canada');
   END;
   ```

2. Create a procedure called *find_region_and_currency* to fetch and return the currency and region in which a country is located.
   - Pass COUNTRY_NAME as an IN parameter and use a user-defined record as an OUT parameter that returns the country name, its region and currency.

   Hints:
   - Declare a user-defined record TYPE in the package spec with appropriate components. Use this record type to declare record variables in your procedure.

   Incorporated Topics: Declare global user-defined record TYPE in the package, consisting of fields from different tables.

Tables Used: WF_COUNTRIES, WF_WORLD_REGIONS, WF_CURRENCIES
To test:

```
DECLARE
  v_region_record traveler_assistance_package.region_rec_type;
BEGIN
  traveler_assistance_package.find_region_and_currency
             ('Canada', v_region_record);
  DBMS_OUTPUT.PUT_LINE
    (v_region_record.country_name||'*** '
     ||v_region_record.region_name||'*** '
     ||v_region_record.currency_name);
END;
```

3. Create a procedure called *display_country_flag* to display a country name and the flag length. Pass COUNTRY_NAME as an IN parameter.

   Hints:
   - We cannot display the BLOB value in Application Express; instead, fetch and display the length in bytes of the FLAG column value.

   Incorporated Topics: Utilizing DBMS_LOB package and it's procedures.
   Tables used: WF_COUNTRIES
   Topics incorporated: working with BLOB data type.
   To test:

```
BEGIN
  traveler_assistance_package.display_country_flag('Canada');
END;
```

4. Create a procedure *countries_in_same_region* to fetch and return all the countries in the same region.
   - Pass REGION_NAME as an IN parameter and a PLSQL associative array of records (an INDEX BY table) as an OUT parameter. Return REGION_NAME, COUNTRY_NAME, and CURRENCY_NAME through the OUT parameter for all the countries in the requested region.

   Hints:
   - The OUT parameter should be an associative array of records.
   - Declare an associative array of records TYPE in the package spec and use this type declaration to declare the OUT parameter.
   - The *print_region_array* procedure will display the contents of the array returned by the OUT parameter.

   Incorporated Topics: Create a global associative array of record TYPE in the package spec. Create a global record TYPE in the package spec. Populate the array of reords in a loop, Return multiple rows back to the calling environment through the OUT parameter of the procedure.

Tables Used: WF_COUNTRIES, WF_WORLD_REGIONS, WF_CURRENCIES

To test:

```
DECLARE
   v_region_array  traveler_assistance_package. countries_in_region_arr_type;
BEGIN
      traveler_assistance_package.countries_in_same_region
        ('Central america', v_region_array);
      traveler_assistance_package.print_region_array (v_region_array);
END;
```

5. Create a procedure *print_region_array* to display the content of an array of records that is passed to it.
   - Pass an associative array of records that was declared in procedure *countries_in_same_region,* as an IN Parameter. The procedure should display its content.

   Incorporated Topics: display the content of an array of records through a loop.
   To test: invoke *countries_in_same_region* procedure.

6. Create a procedure *country_languages* to fetch and return all the spoken language(s) and the official language(s) for a country.
   - Pass COUNTRY_NAME as an IN parameter. The OUT parameter is an associative array that will return COUNTRY_NAME, LANGUAGE_NAME and OFFICIAL.
     Note: A country may have multiple spoken languages. A country may also have more than one official language. Check the OFFICIAL field in WF_SPOKEN_LANGUAGES table to obtain the official languages for a country.

   Hints:
   - Create a PLSQL associative array of record TYPE in the package spec. You can use this data type to declare the OUT parameter in the procedure.
   - The *print_language_array* procedure will display the contents of the array returned by the OUT parameter.

   Incorporated Topics: Create a global associative array of record TYPE in the package spec. Create a global record TYPE in the package spec. Populate the array of reords in a loop, Return multiple rows back to the calling environment through the OUT parameter of the procedure.

   Tables Used: WF_COUNTRIES, WF_WORLD_REGIONS, WF_CURRENCIES

To test:

```
DECLARE
   v_lang_array  traveler_assistance_package. country_lang_arr_type;
BEGIN
      traveler_assistance_package.country_languages
        ('argentine republic', v_lang_array);
      traveler_assistance_package.print_language_array (v_lang_array);
END;
```

7. Create a procedure *print_language_array* to display the content of an array of records that is passed to it.
    - Pass an associative array of records that was declared in procedure *countries_languages*, as an IN Parameter. The procedure should display its content.

Incorporated Topics: display the content of an array of records through a loop.
To test: invoke *country_languages* procedure.

Part 2: Traveler System Administration

Create a package called *traveler_admin_package*, which can be used to maintain the system.

1. Create a procedure *display_disabled_triggers* that displays list of all disabled triggers in your schema.

   Incorporated Topics: Read trigger information from the data dictionary.
   Tables Used: USER_TRIGGERS;
   To test:

   ```
   BEGIN
      traveler_admin_package.display_disabled_triggers;
   END;
   ```

2. Create a function *all_dependent_objects* that returns all the dependent objects for a particular object.
   - Pass OBJECT_NAME as an IN parameter and return an array that contains the NAME , TYPE, REFERENCED_NAME AND REFERENCED_TYPE values.

   Hints:
   - Query the data dictionary and RETURN an associative array of records from the body of the function.
   - If a function returns an empty array, an ORA-06502 exception will be raised. Include code to test whether the associative array contains at least one record; if it does not, populate the first field of the first record with a suitable message.

   Incorporated Topics: Object Dependency. Looking up dependency information from the data dictionary.
   Tables Used: USER_DEPENDENCIES

   To test:

   ```
   DECLARE
      v_dep_arr  traveler_admin_package.dep_obj_arr_type;
   BEGIN
      v_dep_arr :=  traveler_admin_package.all_dependent_objects('wf_countries');
      traveler_admin_package.print_dependent_objects(v_dep_arr);
   END;
   ```

3. Create a procedure *print_dependent_objects* that displays the array of dependent objects returned by the *all_dependent_objects* function.

Suggested Solutions

Part 1: Package traveler_assistance_package

Package Specification

```
CREATE OR REPLACE PACKAGE traveler_assistance_package
IS
TYPE region_rec_type IS RECORD
          (country_name    wf_countries.country_name%TYPE,
           region_name    wf_world_regions.region_name%TYPE,
           currency_name  wf_currencies.currency_name%TYPE);

TYPE lang_rec_type IS RECORD
          (country_name   wf_countries.country_name%TYPE,
           language_name  wf_languages.language_name%TYPE,
           official       wf_spoken_languages.official%TYPE);

TYPE countries_in_region_arr_type
  IS TABLE OF region_rec_type INDEX BY PLS_INTEGER;

TYPE country_lang_arr_type
  IS TABLE OF lang_rec_type INDEX BY PLS_INTEGER;

---- PROCEDURES ----
PROCEDURE country_demographics
   (p_country_name         wf_countries.country_name%TYPE);

PROCEDURE find_region_and_currency
   (p_country_name         wf_countries.country_name%TYPE,
    p_region_record   OUT  region_rec_type);

PROCEDURE display_country_flag
   (p_country_name         wf_countries.country_name%TYPE);

PROCEDURE countries_in_same_region
   (p_region_name          wf_world_regions.region_name%TYPE,
    p_country_region_array OUT countries_in_region_arr_type);

PROCEDURE country_languages
   (p_country_name           wf_countries.country_name%TYPE,
    p_country_lang_arr  OUT  country_lang_arr_type);

PROCEDURE print_region_array
   (p_country_region_array   countries_in_region_arr_type);

PROCEDURE print_language_array
```

```
   (p_lang_array            country_lang_arr_type);

END traveler_assistance_package;
```

Package Body

```
CREATE OR REPLACE PACKAGE BODY traveler_assistance_package
IS

/* This procedure fetches and displays information about a single
country */

PROCEDURE country_demographics
  (p_country_name IN wf_countries.country_name%TYPE)
IS
  TYPE country_record_type IS RECORD
    (country_name   wf_countries.country_name%TYPE,
     location       wf_countries.location%TYPE,
     capitol        wf_countries.capitol%TYPE,
     population     wf_countries.population%TYPE,
     airports       wf_countries.airports%TYPE,
     climate        wf_countries.climate%TYPE);

  country_rec       country_record_type;
  v_country_name    wf_countries.country_name%TYPE;

BEGIN
  v_country_name:= LOWER(p_country_name);
  SELECT country_name, location, capitol,
         population, airports, climate
    INTO country_rec
    FROM wf_countries
    WHERE LOWER(country_name) = v_country_name;

  DBMS_OUTPUT.PUT_LINE('**Country Name:
'||Country_rec.country_name);
  DBMS_OUTPUT.PUT_LINE('**Location: ' ||country_rec.location  ||'
'||
'**Capitol: ' ||country_rec.capitol ||' '||'**Population:
'||country_rec.population||' '||
'**Airports: '||country_rec.airports||' '||'**Climate:
'||country_rec.climate);
EXCEPTION
  WHEN NO_DATA_FOUND then
    RAISE_APPLICATION_ERROR(-20003, 'Country Not Found...');

END country_demographics;
```

```
----------------------------

/* This procedure fetches collection data about a single country
from multiple tables, returning the fetched data in an array
structure */

PROCEDURE find_region_and_currency
  ( p_country_name  IN  wf_countries.country_name%TYPE,
    p_region_record OUT region_rec_type)
IS
  v_region_rec    region_rec_type;
  v_country_name  wf_countries.country_name%TYPE;

BEGIN
  v_country_name := LOWER(p_country_name);
  SELECT country_name, region_name, currency_name
    INTO p_region_record
    FROM wf_countries c, wf_world_regions r, wf_currencies cur
    WHERE c.region_id = r.region_id
    AND c.currency_code = cur.currency_code
    AND LOWER(c.country_name) = v_country_name;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, 'Country Not Found...');

END find_region_and_currency;
-------------------------------

/* This procedure uses DBMS_LOB to display the length of a
country’s flag */

PROCEDURE display_country_flag
  (p_country_name  IN wf_countries.country_name%TYPE)
IS
  v_country_name      wf_countries.country_name%TYPE;
  v_flag              wf_countries.flag%TYPE;
  v_flag_length       NUMBER(9);

BEGIN
  v_country_name := LOWER(p_country_name);
  SELECT flag INTO v_flag
    FROM wf_countries
    WHERE LOWER(country_name) = v_country_name;
  v_flag_length := DBMS_LOB.GETLENGTH(v_flag);
  DBMS_OUTPUT.PUT_LINE
     (p_country_name||'  Flag length: '||v_flag_length);
```

```
END display_country_flag;
---------------------------

/* This procedure fetches the country name, region name and
currency for all countries in a region, returning the fetched data
as a collection (an INDEX BY table) */

PROCEDURE countries_in_same_region
  (p_region_name           IN   wf_world_regions.region_name%TYPE,
   p_country_region_array  OUT  countries_in_region_arr_type)
IS
  v_region_name     wf_world_regions.region_name%TYPE;
  i pls_integer:=1;
  CURSOR country_region_cur IS
    SELECT country_name, region_name, currency_name
      FROM   wf_countries c, wf_world_regions r, wf_currencies cur
      WHERE c.region_id = r.region_id
      AND c.currency_code = cur.currency_code
      AND LOWER(r.region_name) = v_region_name;

BEGIN
  v_region_name := LOWER(p_region_name);
  FOR v_region_rec IN country_region_cur LOOP
       p_country_region_array(i) := v_region_rec;
       i:= i+1;
  END LOOP;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, 'Country Not Found...');

END countries_in_same_region;

/* This procedure returns an array of records with all the
languages spoken in a country and whether each language is official
or not */

PROCEDURE country_languages
  (p_country_name     IN   wf_countries.country_name%TYPE,
   p_country_lang_arr OUT  country_lang_arr_type)

IS
  v_country_name      wf_countries.country_name%TYPE;
  i pls_integer:=1;
  CURSOR country_lang_cur IS
    SELECT c.country_name,l.language_name, sl.official
```

```
      FROM   wf_countries c, wf_spoken_languages sl, wf_languages l
      WHERE  c.country_id = sl.country_id
      AND sl.language_id = l.language_id
      AND LOWER(c.country_name) = v_country_name;

BEGIN
  v_country_name := LOWER(p_country_name);
  FOR v_lang_rec IN country_lang_cur LOOP
    p_country_lang_arr(i) := v_lang_rec;
    i:= i+1;
  END LOOP;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, 'Country Not Found...');
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20004, 'Error Found. Program
terminated...');

END country_languages;

/* This procedure displays the array of countries in a region which
was passed back by the COUNTRIES_IN_SAME_REGION procedure */

PROCEDURE print_region_array
   (p_country_region_array   IN   countries_in_region_arr_type)
IS

BEGIN
  DBMS_OUTPUT.PUT_LINE ('Country name             Region name
Currency name');
  FOR i IN
p_country_region_array.FIRST..p_country_region_array.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE
      (RPAD(p_country_region_array(i).country_name,25) ||
       RPAD(p_country_region_array(i).region_name,20)  ||
       p_country_region_array(i).currency_name);
  END LOOP;

END print_region_array;

/* This procedure displays the array of languages for a country
which was passed back by the COUNTRY_LANGUAGES procedure */

PROCEDURE print_language_array
   (p_lang_array   IN   country_lang_arr_type)
```

```
IS

BEGIN
  DBMS_OUTPUT.PUT_LINE ('Country name             Language name
Official');
  FOR i IN p_lang_array.FIRST..p_lang_array.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE
      (RPAD(p_lang_array(i).country_name,25) ||
       RPAD(p_lang_array(i).language_name,20) ||
       p_lang_array(i).official);
  END LOOP;

END print_language_array;

END traveler_assistance_package;
```

---- END OF PACKAGE ----

Part2: Package traveler_admin_package

Package Spec:

```
CREATE OR REPLACE PACKAGE traveler_admin_package
IS

TYPE dep_obj_rec_type IS RECORD
       (name             USER_DEPENDENCIES.name%TYPE,
        type             USER_DEPENDENCIES.type%TYPE,
        referenced_name  USER_DEPENDENCIES.referenced_name%TYPE,
        referenced_type   USER_DEPENDENCIES.referenced_type%TYPE);

TYPE dep_obj_arr_type
  IS TABLE OF dep_obj_rec_type INDEX BY PLS_INTEGER;

PROCEDURE display_disabled_triggers;

FUNCTION all_dependent_objects
  (p_object_name   IN  VARCHAR2)
RETURN  dep_obj_arr_type;

PROCEDURE print_dependent_objects
  (p_dep_obj_arr   IN   dep_obj_arr_type);

END traveler_admin_package;
```

Package Body:

```
CREATE OR REPLACE PACKAGE BODY traveler_admin_package
IS

/* This procedure displays a list of all disabled triggers in your
schema */

PROCEDURE display_disabled_triggers
IS
  CURSOR trigger_curs IS
    SELECT trigger_name
      FROM user_triggers
      WHERE status = 'DISABLED';

BEGIN
  FOR v_trigger_rec IN trigger_curs LOOP
    DBMS_OUTPUT.PUT_LINE
      (v_trigger_rec.trigger_name||':  '||'is disabled. ');
  END LOOP;
END display_disabled_triggers;

/* This function returns all objects directly dependent on a
requested object */

FUNCTION all_dependent_objects
  (p_object_name      VARCHAR2)
RETURN dep_obj_arr_type
IS
  CURSOR dep_obj_curs IS
    SELECT name, type, referenced_name, referenced_type
      FROM USER_DEPENDENCIES
      WHERE referenced_name = UPPER(p_object_name)
      ORDER BY type;
  v_dep_obj_arr  dep_obj_arr_type;
  i              pls_integer:= 1;
  v_data_exists  BOOLEAN;
BEGIN
  FOR v_dep_obj_rec IN dep_obj_curs LOOP
    v_dep_obj_arr(i):= v_dep_obj_rec;
    i := i + 1;
  END LOOP;
  v_data_exists := (v_dep_obj_arr.COUNT > 0);
  IF NOT v_data_exists THEN
    v_dep_obj_arr(1).name := 'No Dependent Objects Found... ';
  END IF;
  RETURN v_dep_obj_arr;

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred');
    RETURN v_dep_obj_arr ;

END all_dependent_objects;

/* This procedure displays the data returned by the
ALL_DEPENDENT_OBJECTS function */

PROCEDURE print_dependent_objects
  (p_dep_obj_arr   IN   dep_obj_arr_type)
IS

BEGIN
  DBMS_OUTPUT.PUT_LINE ('Name                           Type
Referenced Name                Referenced Type');
  FOR i IN p_dep_obj_arr.FIRST..p_dep_obj_arr.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE
      (RPAD(p_dep_obj_arr(i).name,31) ||
       RPAD(p_dep_obj_arr(i).type,20) ||
       RPAD(p_dep_obj_arr(i).referenced_name,31) ||
       p_dep_obj_arr(i).referenced_type);
  END LOOP;

END print_dependent_objects;

END traveler_admin_package;
```

---- END OF PACKAGE ----
