Documentación

El primer paso para empezar el proyecto, como es lógico, fue leer las instrucciones, donde nos dimos cuenta que durante el proyecto debiamos trabajar en dos paquetes, `traveler_assistance_package` y `traveler_admin_package`.

Paquete **TRAVELER ASSISTANCE**.

Para empezar, creamos el *package header* del paquete **traveler_assistance_package**:

```
-- PACKAGE HEADER
CREATE OR REPLACE PACKAGE traveler_assistance_package
```

Así como también el *package body*:

```
-- PACKAGE BODY
CREATE OR REPLACE PACKAGE BODY traveler_assistance_package
```

El paquete `traveler_assistance_package` debe contener 6 procedimientos:

1. country_demographics
2. find_region_and_currency
3. countries_in_same_region
4. print_region_array
5. country_languages
6. print_language_array

Donde a continuación se documenta cada uno:

1. **country_demographics**

Se nos pide programar un *procedimiento* llamado **country_demographics**, que nos permita mostrar información específica acerca de un país. Además, en la siguiente instrucción, se nos pide pasar `COUNTRY_NAME` como parámetro de entrada.

El primer paso para empezar a programar cada procedimiento dentro de un paquete es colocarlo en el *package header*, por lo cual colocamos el nombre del procedimiento junto con el parámetro que recibe:

```
PROCEDURE country_demographics(v_country_name VARCHAR2); -- Procedimiento 1
// Esto va en el PACKAGE HEADER
```

Posterior a ello, empezamos a escribir el cuerpo del procedimiento dentro del *package body*:

```
PROCEDURE country_demographics(v_country_name VARCHAR2) IS
```

> Donde atendemos la instrucción que nos pide que enviemos como parámetro el `country_name`.

Después, la instrucción nos pide que mostremos las columnas `COUNTRY_NAME`, `LOCATION`, `CAPITOL`, `POPULATION`, `AIRPORTS` Y `CLIMATE`. Sin embargo, la instrucción **no** especifica directamente de cuál tabla es de la que tenemos que obtener todos estos datos, por lo cual el uso de la sentencia **DESCRIBE** nos puede ser útil para estos casos:

```
SQL>   DESCRIBE WF_countries;
Name    Null?   Type
COUNTRY_ID  NOT NULL    NUMBER(4)
REGION_ID   NOT NULL    NUMBER(3)
COUNTRY_NAME    NOT NULL VARCHAR2(70) -- COUNTRY_NAME
COUNTRY_TRANSLATED_NAME     VARCHAR2(40)
LOCATION        VARCHAR2(90) -- LOCATION
CAPITOL     VARCHAR2(50) -- CAPITOL
AREA        NUMBER(15)
COASTLINE       NUMBER(8)
LOWEST_ELEVATION        NUMBER(6)
LOWEST_ELEV_NAME        VARCHAR2(70)
HIGHEST_ELEVATION       NUMBER(6)
HIGHEST_ELEV_NAME       VARCHAR2(50)
DATE_OF_INDEPENDENCE        VARCHAR2(30)
NATIONAL_HOLIDAY_NAME       VARCHAR2(200)
NATIONAL_HOLIDAY_DATE       VARCHAR2(30)
POPULATION      NUMBER(12) -- POPULATION
POPULATION_GROWTH_RATE      VARCHAR2(10)
LIFE_EXPECT_AT_BIRTH        NUMBER(6,2)
MEDIAN_AGE      NUMBER(6,2)
AIRPORTS        NUMBER(6) -- AIRPORTS
CLIMATE     VARCHAR2(1000) -- CLIMATE
FIPS_ID     CHAR(2)
INTERNET_EXTENSION      VARCHAR2(3)
FLAG        BLOB
CURRENCY_CODE   NOT NULL    VARCHAR2(7)
```

> Donde podemos observar que dichas columnas se encuentran en la tabla `WF_COUNTRIES`.

Posteriormente, la instrucción nos pide algo bien específico: "Usa una estructura de registro definido por el Usuario para la cláusula INTO de tu sentencia Select", donde la palabra clave es "*esctructura de registro*", que nos insinúa que debemos utilizar alguna estructura de datos capaz de alojar toda la información obtenida. Dada la situación, el uso de un `RECORD` es de gran utilidad.

Teniendo en cuenta los datos que debemos obtener, creamor un `RECORD` que sea capaz de almacenar todo ello, esto primeramente debe estar definido en el *package header*:

```
```

```
TYPE country_record 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
);
```

Posteriormente, declaramos una variable de dicho tipo en el *package body*, más específicamente en la función que estamos programando:

```
v_country_information country_record;
```

Una vez con todas las variables definidas, solamente queda ejecutar la sentencia **SELECT** para recuperar los datos:

```
SELECT COUNTRY_NAME, LOCATION, CAPITOL, POPULATION, AIRPORTS, CLIMATE
INTO v_country_information
FROM WF_COUNTRIES
WHERE UPPER(COUNTRY_NAME) = UPPER(v_country_name); -- Uso del parámetro
v_country_name
```

> La palabra clave `INTO` guarda los datos obtenidos dentro de la variable `v_country_information`.

Una vez con los datos almacenados, los imprimimos por consola con la función **PUT_LINE** del paquete **DBMS_OUTPUT**:

```
DBMS_OUTPUT.PUT_LINE('Country Name: ' ||
v_country_information.country_name);
DBMS_OUTPUT.PUT_LINE('Location: ' || v_country_information.location);
DBMS_OUTPUT.PUT_LINE('Capitol: ' || v_country_information.capitol);
DBMS_OUTPUT.PUT_LINE('Population: ' ||
TO_CHAR(v_country_information.population));
DBMS_OUTPUT.PUT_LINE('Airports: ' ||
TO_CHAR(v_country_information.airports));
DBMS_OUTPUT.PUT_LINE('Climate: ' || v_country_information.climate);
```

Finalmente, la instrucción nos pide una última cosa, que es lanzar una excepción si el país indicado **NO** existe. Para lograr este objetivo, recurrimos a la *excepción implícita* `NO_DATA_FOUND` para ello, y utilizamos la función `RAISE_APPLICATION_ERROR` para lanzar el error, donde estamos obviando que esto debe ser programado dentro del bloque **EXCEPTION**.

```

```

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'No se encontró información para el
país: ' || v_country_name);
```

2. **find_region_and_currency**

Está instrucción nos pide encontrar un procedimiento llamado `find_region_and_currency`, que lea y regrese la **moneda** y **región** en la cual un país está localizado. Además, también nos pide enviar `COUNTRY_NAME`, o el nombre del país.

Nuevamente, como primer paso, definimos la función en el *package header*:

```
PROCEDURE find_region_and_currency(COUNTRY_NAME IN VARCHAR2, country OUT
country_type); -- Procedimiento 2
```

Así como también en el *package body*:

```
 PROCEDURE find_region_and_currency(COUNTRY_NAME IN VARCHAR2)
```

También nos pide utilizar un registro **definido por el usuario** como parámetro de salida, que devuelva la información del **país**, **región** y **moneda**. Está última instrucción debemos tomarla por partes.

Primero, al igual que en el procedimiento anterior, se entiende que por **registro definido por el usuario**, debemos definir un *RECORD* que sea capaz de almacenar los datos de interés, para ello, definimos dicha estructura en el *package header*:

```
TYPE country_info IS RECORD(
    country_name -- ? Aún no conocemos el tipo, ni la ubicación de estas
columnas
    region -- ? Aún no conocemos el tipo, ni la ubicación de estas columnas
    currency -- ? Aún no conocemos el tipo, ni la ubicación de estas
columnas
);
```

Y modificamos el procedimiento para que regrese una variable de dicho tipo. Sin embargo, es conveniente recordar que un *procedimiento* **no** puede devolver valores, sin embargo, si podemos enviarlo como **OUT**, con el fin de que el valor pueda ser modificado dentro procedimiento.

```
PROCEDURE find_region_and_currency(COUNTRY_NAME IN VARCHAR2, country OUT
country_info)
```

Ahora viene lo complicado, ejecutar la sentencia **SELECT** completa. Nuevamente no nos dicen que tabla/tablas utilizar directamente, sin embargo, podemos hacer `DESCRIBE`:

```
SQL>   DESCRIBE WF_COUNTRIES;
Name    Null?   Type
COUNTRY_ID  NOT NULL    NUMBER(4)
REGION_ID   NOT NULL    NUMBER(3) -- IDENTIFICADOR DE CADA REGIÓN
COUNTRY_NAME    NOT NULL    VARCHAR2(70) -- COLUMNA CON LA CUAL COMPARAR EL
PARÁMETRO
COUNTRY_TRANSLATED_NAME     VARCHAR2(40)
LOCATION        VARCHAR2(90)
CAPITOL     VARCHAR2(50)
AREA        NUMBER(15)
COASTLINE       NUMBER(8)
LOWEST_ELEVATION        NUMBER(6)
LOWEST_ELEV_NAME        VARCHAR2(70)
HIGHEST_ELEVATION       NUMBER(6)
HIGHEST_ELEV_NAME       VARCHAR2(50)
DATE_OF_INDEPENDENCE        VARCHAR2(30)
NATIONAL_HOLIDAY_NAME       VARCHAR2(200)
NATIONAL_HOLIDAY_DATE       VARCHAR2(30)
POPULATION      NUMBER(12)
POPULATION_GROWTH_RATE      VARCHAR2(10)
LIFE_EXPECT_AT_BIRTH        NUMBER(6,2)
MEDIAN_AGE      NUMBER(6,2)
AIRPORTS        NUMBER(6)
CLIMATE     VARCHAR2(1000)
FIPS_ID     CHAR(2)
INTERNET_EXTENSION      VARCHAR2(3)
FLAG        BLOB
CURRENCY_CODE   NOT NULL    VARCHAR2(7) -- IDENTIFICADOR DE MONEDA

SQL>   DESCRIBE WF_WORLD_REGIONS;
Name    Null?   Type
REGION_ID   NOT NULL    NUMBER(3) -- IDENTIFICADOR DE REGIÓN
REGION_NAME NOT NULL    VARCHAR2(35)

SQL>   DESCRIBE WF_CURRENCIES;
Name    Null?   Type
CURRENCY_CODE   NOT NULL    VARCHAR2(7) -- IDENTIFICADOR DE MONEDA
CURRENCY_NAME   NOT NULL    VARCHAR2(40)
COMMENTS        VARCHAR2(150)
```

Donde, primero, para obtener el `CURRENCY_NAME`, notamos que tenemos que realizar el **JOIN** de la columna `WF_COUNTRIES.CURRENCY_CODE` y `WF_CURRENCIES.CURRENCY_CODE`. Por otro lado, para obtener el nombre de la región, podemos notar que podemos obtener el `REGION_NAME` del **JOIN** de la columna `WF_COUNTRIES.REGION_ID` y `WF_WORLD_REGIONS.REGION_ID`.

Una vez identificadas las columnas, podemos también completar la definición del **RECORD**:

```
TYPE country_info IS RECORD(
    country_name WF_COUNTRIES.country_name%TYPE,
    region WF_WORLD_REGIONS.REGION_NAME%TYPE,
    currency WF_CURRENCIES.currency_name%TYPE
);
```

Después, una vez con las columnas ubicadas, procedimos a realizar la sentencia `SELECT` y a guardar el resultado en el *RECORD*:

```
SELECT ctr.country_name, wr.region_name, cur.currency_name INTO country
FROM WF_COUNTRIES ctr, WF_WORLD_REGIONS wr, WF_CURRENCIES cur
WHERE LOWER(ctr.COUNTRY_NAME) = LOWER(COUNTRY_NAME) AND
ctr.REGION_ID = wr.REGION_ID AND
ctr.CURRENCY_CODE = cur.CURRENCY_CODE;
```

Finalmente, mostramos una excepción si no se encontraron resultados:

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'No se encontraron ciudades para '
|| COUNTRY_NAME);
```

3. **countries_in_same_region**

El procedimiento `countries_in_same_region` se encarga de **leer** y **devolver** todos los países que pertenecen a una misma región específica. Este procedimiento recibe `REGION_NAME` como un parámetro de entrada y utiliza un arreglo asociativo de registros (**INDEX BY**) como parámetro de salida, el cual devolverá los campos `REGION_NAME`, `COUNTRY_NAME`, y `CURRENCY_NAME` para todos los países que coincidan con la región solicitada.

Para comenzar, *definimos* el procedimiento en el *package header* con la siguiente declaración:

```
PROCEDURE countries_in_same_region(v_region_name IN VARCHAR2, countries OUT
countries_type);
```

Dentro del cuerpo del procedimiento, se define una variable `v_country` de tipo `country_type`, que almacenará temporalmente los datos de cada país mientras se procesan los resultados de la consulta. También se inicializa una variable `i` de tipo **PLS_INTEGER** con el valor **1**, que funcionará como índice para el arreglo asociativo `countries`:

```
v_country country_type;
i PLS_INTEGER := 1;
```

El procedimiento utiliza un **FOR LOOP** para iterar sobre los resultados de una consulta SQL que selecciona `country_name`, `region_name`, y `currency_name` de las tablas `WF_COUNTRIES`, `WF_WORLD_REGIONS`, y `WF_CURRENCIES`. Los datos se obtienen mediante un **JOIN** sobre las columnas `region_id` y `currency_code`, asegurando que se seleccionen *únicamente* los registros que coincidan con el nombre de la región proporcionado en el **argumento**. La comparación se realiza de manera ***insensible*** a mayúsculas o minúsculas utilizando la función `LOWER`:

```
FOR r IN (SELECT c.country_name, r.region_name, cu.currency_name
          FROM WF_COUNTRIES c
          JOIN WF_WORLD_REGIONS r ON c.region_id = r.region_id
          JOIN WF_CURRENCIES cu ON c.currency_code = cu.currency_code
          WHERE LOWER(r.region_name) = LOWER(v_region_name)) LOOP
```

Dentro del bucle, se asignan los valores recuperados a la variable `v_country`:

```
v_country.country_name := r.country_name;
v_country.region := r.region_name;
v_country.currency := r.currency_name;
```

Estos valores se almacenan luego en el arreglo countries utilizando el índice i, que se incrementa con cada iteración para asegurar que cada registro se almacene en la posición correcta dentro del arreglo:

```
countries(i) := v_country;
i := i + 1;
```

Una vez que el bucle ha procesado todos los registros, se verifica si `i` sigue siendo igual a **1**. Si este es el caso, significa que **no se encontraron registros** que coincidan con el nombre de la región, por lo que se lanza la excepción `NO_DATA_FOUND` para indicar que *no hay resultados* disponibles para la región solicitada:

```
IF i = 1 THEN
    RAISE NO_DATA_FOUND;
END IF;
```

Finalmente, se maneja la excepción `NO_DATA_FOUND` utilizando `RAISE_APPLICATION_ERROR` para generar un mensaje de error que notifique al usuario que **no** se encontraron ciudades para la región especificada:

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'No se encontraron ciudades para '
|| v_region_name);
```

4. **print_region_array**

El procedimiento `print_region_array` está diseñado para **mostrar el contenido de un arreglo de registros** que se pasa como *parámetro de entrada*. Este procedimiento es especialmente útil para *visualizar* la información previamente almacenada en un arreglo asociativo de registros, como el que se genera en el procedimiento `countries_in_same_region`.

Para comenzar, definimos el procedimiento en el *package header* del paquete `traveler_assistance_package` con la siguiente declaración:

```
PROCEDURE print_region_array(countries countries_type); -- Procedimiento 4
```

El propósito de este procedimiento es iterar a través del arreglo asociativo countries, el cual contiene registros que incluyen los campos `country_name`, `region`, y `currency`. El procedimiento *recorrerá* cada uno de estos registros y **mostrará** sus contenidos utilizando la función `DBMS_OUTPUT.PUT_LINE`, que permite imprimir el mensaje por consola.

Dentro del cuerpo del procedimiento, se utiliza un **FOR LOOP** para recorrer el arreglo countries. Este bucle se extiende desde el *primer índice* (`countries.FIRST`) hasta el *último* (`countries.LAST`), asegurando que todos los registros almacenados en el arreglo se procesen:

```
FOR i IN countries.FIRST .. countries.LAST LOOP
```

Dentro del bucle, para cada índice `i`, se imprimen los valores de `country_name`, `region`, y `currency`, correspondientes al registro almacenado en esa posición del arreglo. La impresión se realiza en un formato **claro** y **legible** para el usuario:

```
DBMS_OUTPUT.PUT_LINE('Country Name : ' || countries(i).country_name);
DBMS_OUTPUT.PUT_LINE('Region       : ' || countries(i).region);
DBMS_OUTPUT.PUT_LINE('Currency     : ' || countries(i).currency);
```

Además, para **mejorar la legibilidad** de la salida, se incluye una línea de separación entre los registros utilizando una serie de guiones:

```
DBMS_OUTPUT.PUT_LINE('-----------------------------------');
```

5. **country_languages**

El procedimiento `country_languages` tiene como objetivo **leer** y **devolver** todos los idiomas hablados en un *país determinado*, así como identificar **cuál** de esos idiomas es el idioma oficial. Este *procedimiento* recibe `COUNTRY_NAME` como un *parámetro de entrada* y utiliza un arreglo asociativo de registros como

parámetro de salida. El arreglo devuelto contiene los campos `COUNTRY_NAME`, `LANGUAGE_NAME`, y `OFFICIAL` para cada idioma asociado con el país especificado.

El procedimiento se define en el `package header` del paquete `traveler_assistance_package` con la siguiente declaración:

```
PROCEDURE country_languages(v_country_name IN VARCHAR2, country_lang OUT
country_languages_type); -- Procedimiento 5
```

Dentro del cuerpo del procedimiento, se inicia declarando una variable `v_language` de tipo `country_language_type`, que se utilizará para **almacenar temporalmente** los datos de cada idioma recuperado. También se inicializa una variable `i` de tipo **PLS_INTEGER** con el valor 1, que funcionará como índice para el arreglo asociativo `country_lang`.

El procedimiento utiliza un FOR LOOP que itera sobre los resultados de una consulta SQL. Esta consulta selecciona los campos country_name, language_name, y official de las tablas WF_COUNTRIES, WF_LANGUAGES, y WF_SPOKEN_LANGUAGES. Para obtener estos datos, se realiza un JOIN entre las tablas utilizando las columnas country_id y language_id, lo que asegura que se recuperen únicamente los registros que coincidan con el nombre del país proporcionado. La comparación del nombre del país se realiza de manera insensible a mayúsculas o minúsculas utilizando la función UPPER: