PRÁCTICA 13
PROGRAMACIÓN CON SQL PARTE 2

1.1. OBJETIVO:

El alumno pondrá en práctica los conceptos básicos de programación procedural en la construcción de cursores, funciones y manejo de LOBs.

1.2. ACTIVIDADES PREVIAS.

- Revisar el documento general de prácticas correspondiente a la práctica 13
- Para mayores detalles en cuanto a los temas que se revisan en esta práctica, revisar el archivo `tema10.pdf` que se encuentra en la carpeta compartida `BD/teoria/apuntes/tema10`

1.3. CONCEPTOS REQUERIDOS PARA REALIZAR ESTA PRÁCTICA.

1.3.1. Cursores

Hasta este momento, en todos los ejemplos PL/SQL la instrucción `select` devuelve exactamente registro. Si deseamos obtener más de un valor se requiere hacer uso de un **cursor** explícito para extraer individualmente cada fila.

Un cursor es un área de memoria privada utilizada para realizar operaciones con los registros devueltos tras ejecutar una sentencia `select`. Existen dos tipos:

- Implícitos
  - Creados y administrados internamente por el manejador para procesar sentencias SQL
- Explícitos
  - Declarados explícitamente por el programador.e

Su sintaxis es la siguiente:

```
cursor <cursor_name> is <select_statement>
```

El cursor explícito se abre mediante la siguiente sintaxis:

```
open <cursor_name>;
```

La extracción de los datos y su almacenamiento en variables PL/SQL se realiza utilizando la siguiente sintaxis:

```
fetch <cursor_name> into listavariables;
fetch <cursor_name> into registro pl/sql;
```

Importante: ¡Al terminar de usar un cursor se debe cerrar!

```
close <cursor_name>;
```

Un cursor define 6 atributos que se emplean para controlar el acceso a los datos, en especial los siguientes 4 atributos:

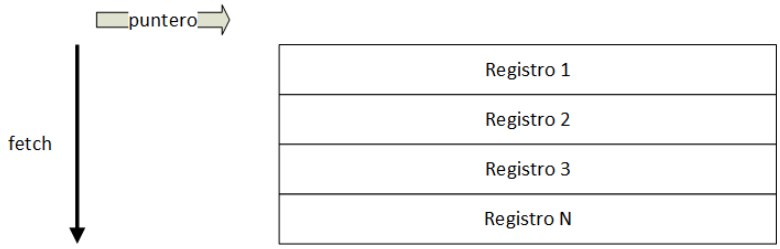
- `%found` y `%notfound` para controlar si la última orden `fetch` devolvió o no una fila.
- `%isopen` para saber si el cursor está abierto
- `%rowcount` que devuelve el número de filas extraídas por el cursor.

Existen 2 principales estrategias para procesar un cursor:

- Simple loop.
- For loop.

1.3.1.1. Simple loop

- Un Simple loop requiere que el programador controle el punto de inicio y el punto en el que el loop debe terminar. De no realizarse de forma correcta, se pueden provocar loop infinitos entre otros errores.
- En el ejemplo anterior, observar que únicamente se especifica la instrucción `loop` sin especificar alguna otra configuración para controlar su inicio y su fin. Esto implica un ciclo infinito, que debe ser roto en algún momento, al cumplirse cierta condición.
- La instrucción `fetch` provoca que el puntero asociado al cursor se mueva al siguiente renglón obtenido. Un cursor puede verse como una lista de renglones y un puntero que se va moviendo de arriba hacia abajo para recorrerlo:



- Observar que la siguiente línea del código es la instrucción `exit`. Si el puntero ha llegado a su fin, el valor del atributo `%notfound` será true, y por lo tanto el ciclo termina. Es importante validar esta condición antes de acceder a los valores de cada registro. De esta forma, la instrucción `exit` permite controlar el momento en el que el ciclo debe terminar.

Ejemplo:

Construir un programa PL/SQL que imprima el nombre, apellidos, nombre de la asignatura y el número de cursos que imparte cada profesor de la universidad.

```
declare
  --declaración del cursor
  cursor cur_datos_profesor is
  select p.nombre,p.apellido_paterno, p.apellido_materno,
    a.nombre, count(*) cursos
  from profesor p, curso c, asignatura a
  where p.profesor_id=c.profesor_id
  and c.asignatura_id = a.asignatura_id
  group by p.nombre,p.apellido_paterno,p.apellido_materno,a.nombre;

  --declaración de variables
```

```
  v_nombre profesor.nombre%type;
  v_ap_pat profesor.apellido_paterno%type;
  v_ap_mat profesor.apellido_materno%type;
  v_asignatura asignatura.nombre%type;
  v_num_cursos number;

begin
 open cur_datos_profesor;
 dbms_output.put_line('resultados obtenidos');
 dbms_output.put_line(
    'nombre  apellido paterno apellido materno  asignatura #cursos');
 loop
    fetch cur_datos_profesor into
      v_nombre,v_ap_pat,v_ap_mat,v_asignatura,v_num_cursos;
    exit  when cur_datos_profesor%notfound;
    dbms_output.put_line(
      v_nombre||' , '||v_ap_pat||' , '||v_ap_mat
      ||' , '||v_asignatura||' , '||v_num_cursos);
  end loop;
  --Importante: Cerrar el cursor al terminar para liberar recursos.
  close cur_datos_profesor;
end;
/
```

1.3.1.2. For loop

- Representa una forma simplificada de un simple loop. Las operaciones recurrentes del cursor como son: abrirlo, el uso de `fetch` y cerrarlo se hacen de forma implícita.
- El programa anterior empleando un for loop se muestra a continuación:

```
set serveroutput on
declare
  --declaración del cursor
 cursor cur_datos_profesor is
 select p.nombre as nombre_profesor,p.apellido_paterno, p.apellido_materno,
    a.nombre as nombre_asignatura, count(*) cursos
 from profesor p, curso c, asignatura a
 where p.profesor_id=c.profesor_id
 and c.asignatura_id = a.asignatura_id
 group by p.nombre,p.apellido_paterno,p.apellido_materno,a.nombre;

begin
 dbms_output.put_line('resultados obtenidos');
 dbms_output.put_line(
    'nombre  apellido paterno apellido materno  asignatura #cursos');
 for p in cur_datos_profesor loop
    dbms_output.put_line(
      p.nombre_profesor||' , '||p.apellido_paterno
      ||' , '||p.apellido_materno
      ||' , '||p.nombre_asignatura||' , '||p.cursos);
  end loop;
end;
/
```

La variable 'p' representa a cada uno de los renglones que obtiene el cursor. Por lo tanto, para acceder a los valores de las columnas se puede emplear la sintaxis `p.<nombre_columna>`

1.3.2. Funciones

- La principal diferencia de una función con respecto a un procedimiento es que una función si puede regresar un valor.
- Lo anterior implica que una función puede ser invocada desde una sentencia SQL mientras que un procedimiento almacenado no.

```
select functionName(<param1>,<param2>,…,<paramN>) from dual
```

- Las funciones también pueden ser empleados como operando derecho en una expresión.

```
v_valor = functionName(<param1>,<param2>,…,<paramN>)
```

- Las funciones no pueden contener código DML, DDL a menos que se utilice una transacción autónoma.

Sintaxis:

```
create [or replace] function [schema.]<fnction_name>
   [(argument [{ in | out | in out }][nocopy] datatype [default expr]
     [,argument [{ in | out | in out }][nocopy] datatype [default expr]
     ]...)
   ]
   return {datatype}
   [ authid [definer | current_user]]
   [ deterministic | parallel_enabled ]
   [ pipelined ]
   [ result cache [relies on <table_name>]]
   is { pl/sql_subprogram_body | call_spec } ;
```

- El usuario que desee crear una función deberá contar con el privilegio `create procedure` (el mismo empleado para los procedimientos).

Ejemplo:

- Crear una función que realice la unión de hasta 5 cadenas empleando un carácter de unión. Por ejemplo, para las cadenas "A", "B", "C," "D", "E", generar una cadena empleando el carácter "#" como carácter de unión. La función deberá regresar la cadena "A#B#C#D#E".
- Como mínimo la función deberá recibir las primeras 2 cadenas y el carácter de unión. El código es el siguiente:

```
create or replace function joinStrings(
  join_string varchar2,
  str_1 varchar2,
  str_2 varchar2,
  str_3 varchar2 default null,
  str_4 varchar2 default null,
  str_5 varchar2 default null
) return varchar2 is
```

```
  v_str varchar2(4000);
  begin
    v_str :=  str_1||join_string||str_2;
    if str_3 is not null then
      v_str := v_str||join_string||str_3;
    end if;
    if str_4 is not null then
      v_str := v_str||join_string||str_4;
    end if;
    if str_5 is not null then
      v_str := v_str||join_string||str_5;
    end if;
    return v_str;
  end;
/
show errors
```

1.3.2.1. Ejecución de funciones.

- Notación posicional:

```
select joinStrings('#','A','B',null,null,'E') as "join_string"
from dual;
```

- Notación por nombre:

```
select joinStrings(
 str_1 => 'A',
 str_5 => 'E',
 str_2 => 'B',
 join_string => '#') as "join_string"
from dual;
```

1.3.3. Large Objects (LOBs)

- LOBs son estructuras empleadas para almacenar objetos como texto, archivos binarios (imágenes, videos, documentos), etc.
- A partir de Oracle 11g, se hace uso de un nuevo concepto llamado ***Secure files*** que permite incrementar considerablemente el desempeño para realizar el manejo de este tipo de objetos.
- Se emplean los tipos de datos `clob`, `nclob` y `blob` para almacenar LOBs en la base de datos, y `bfile` para almacenar objetos fuera de ella.
- `Bfile` almacena una especie de 'puntero' que permite ubicar al archivo fuera de la BD.

1.3.3.1. Objetos CLOB

- Se emplean los tipos de datos `clob` y `nclob`
- Este tipo de dato permite almacenar hasta 128 terabytes de texto en un solo registro, por ejemplo, un capítulo completo de un libro, un documento xml, etc.
- Las columnas declaradas con tipo de dato `clob` típicamente se almacenan de forma separada con respecto al resto del registro. En su lugar se almacena un 'descriptor' o 'locator' el cual puede visualizarse como un puntero que lleva al lugar físico donde se encuentra el texto.
- Existen diversas formas de inicializar un objeto `clob`:

```
v_texto1 clob; --declara un objeto clob nulo
v_texto2 clob := empty_clob(); --declara e inicializa un objeto clob vacio.
v_texto3 clob := 'texto'; --inicializa un objeto clob con una cadena.
```

Ejemplo:

Suponer que existe la siguiente tabla y secuencia.

Crear un procedimiento encargado de registrar un nuevo libro. El procedimiento recibirá los siguientes parámetros:

- `p_articulo_id`: Parámetro de salida en el que el procedimiento actualizará su valor con el identificador asignado empleando la secuencia.
- `p_titulo`: el título del libro
- `p_nombre_archivo`: El nombre del archivo donde se encuentra el texto del libro. Su contenido deberá ser insertado en la columna texto cuyo tipo de dato es CLOB.
- Suponer que los archivos de texto se encuentran en el directorio `/tmp/data_dir`

```
create or replace procedure crea_libro(p_libro_id out number,
  p_titulo in varchar2, p_nombre_archivo in varchar2) is

  v_bfile bfile;
  v_src_offset number := 1;
  v_dest_offset number:= 1;
  v_dest_clob clob;
  v_src_length number;
  v_dest_length number;
  v_lang_context number := dbms_lob.default_lang_ctx;
  v_warning number;

  begin

    v_bfile := bfilename('DATA_DIR',p_nombre_archivo);
    if dbms_lob.fileexists(v_bfile) = 1 and not
      dbms_lob.isopen(v_bfile) = 1 then
        dbms_lob.open(v_bfile,dbms_lob.lob_readonly);
    else
      raise_application_error(-20001,'El archivo '
        ||p nombre archivo
        ||' no existe en el directorio DATA_DIR'
        ||' o el archivo esta abierto');
    end if;

    select libro_seq.nextval into p_libro_id
    from dual;

    insert into libro(libro_id,titulo,texto)
    values(p_libro_id,p_titulo,empty_clob());

    select texto into v_dest_clob
```

```
    from libro
    where libro_id = p_libro_id;

    dbms_lob.loadclobfromfile(
      dest_lob     => v_dest_clob,
      src_bfile    => v_bfile,
      amount       => dbms_lob.getlength(v_bfile),
      dest_offset  => v_dest_offset,
      src_offset   => v_src_offset,
      bfile_csid   => dbms_lob.default_csid,
      lang_context => v_lang_context,
      warning      => v_warning
    );
    dbms_lob.close(v_bfile);

    v_src_length := dbms_lob.getlength(v_bfile);
    v_dest_length := dbms_lob.getlength(v_dest_clob);

    if v_src_length = v_dest_length then
      dbms_output.put_line('Escritura correcta, bytes escritos: '
        || v_src_length);
    else
      raise_application_error(-20002,'Error al escribir datos.\n'
        ||' Se esperaba escribir '||v_src_length
        ||' Pero solo se escribio '||v_dest_length);
    end if;
  end;
/
show errors
```

- Observar el nombre de un objeto llamado 'DATA_DIR'. Se trata de un objeto tipo `directory` que representa (mapea) a una ruta en el servidor.
- El objeto deberá existir en la BD antes de ejecutar el procedimiento. Esta instrucción deberá ser ejecutada por el usuario SYS:

```
create or replace directory data_dir as '/tmp/data_dir';
```

- No olvidar otorgar el privilegio correspondiente para que el usuario que ejecute el procedimiento pueda leer archivos del directorio.
- Notar que en el procedimiento se especifica en mayúsculas ya que los nombres de los objetos se guardan en mayúsculas en el diccionario de datos.

```
grant read on directory data_dir to <usuario>;
```

- El siguiente código invoca al procedimiento anterior.

```
variable v_libro_id number
exec crea_libro(:v_libro_id,'Mi primer libro','prueba.txt');
```

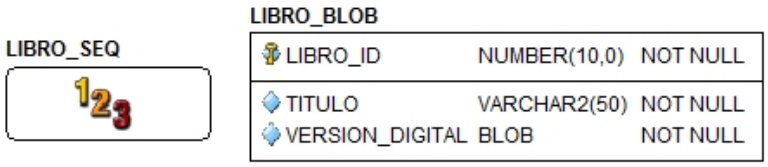
1.3.3.2. Objetos BLOB.

La forma de trabajar con objetos BLOB es similar a la técnica anterior.

```
v_bin1 blob; --declara un objeto blob nulo
v_bin2 blob := empty_blob(); --declara e inicializa un objeto blob vacío.
v_bin3 blob := '43'||'41'||'52'; --inicializa un objeto BLOB con una cadena
                                 --en formato hexadecimal que representa a
```

```
--la palabra CAR.
```

En el siguiente ejemplo se muestra una variante del ejemplo anterior.
Suponer que el contenido del libro ahora se guarda en un documento PDF (archivo binario) en el campo `version_digital`.



La variante del procedimiento anterior para guardar objetos BLOB en la tabla anterior se muestra a continuación. Considerar que existe un archivo llamado prueba.pdf en el directorio configurado.

Observar las diferencias del procedimiento resaltadas en negritas.

```
create or replace procedure crea_libro_blob(p_libro_id out number,
  p_titulo in varchar2, p_nombre_archivo in varchar2) is

  v_bfile bfile;
  v_src_offset number := 1;
  v_dest_offset number:= 1;
  v_dest_blob blob;
  v_src_length number;
  v_dest_length number;

  begin

    v_bfile := bfilename('DATA_DIR',p_nombre_archivo);
    if dbms_lob.fileexists(v_bfile) = 1 and not
      dbms_lob.isopen(v_bfile) = 1 then
        dbms_lob.open(v_bfile,dbms_lob.lob_readonly);
    else
      raise_application_error(-20001,'El archivo '
        ||p_nombre_archivo
        ||' no existe en el directorio DATA_DIR'
        ||' o el archivo esta abierto');
    end if;

    select libro_seq.nextval into p_libro_id
    from dual;

    insert into libro_blob(libro_id,titulo,version_digital)
    values(p_libro_id,p_titulo,empty_blob());

    select version_digital into v_dest_blob
    from libro_blob
    where libro_id = p_libro_id;

    dbms_lob.loadblobfromfile(
      dest_lob       => v_dest_blob,
      src_bfile      => v_bfile,
      amount         => dbms_lob.getlength(v_bfile),
      dest_offset    => v_dest_offset,
      src_offset     => v_src_offset);
```

```
    dbms_lob.close(v_bfile);

    v_src_length := dbms_lob.getlength(v_bfile);
    v_dest_length := dbms_lob.getlength(v_dest_blob);

    if v_src_length = v_dest_length then
      dbms_output.put_line('Escritura correcta, bytes escritos: '
        || v_src_length);
    else
      raise_application_error(-20002,'Error al escribir datos.\n'
        ||' Se esperaba escribir '||v_src_length
        ||' Pero solo se escribio '||v_dest_length);
    end if;
  end;
/
show errors
```

1.3.3.3. Escritura de objetos BLOB en disco.

Revisar el siguiente código encargado de realizar la operación contraria: a partir de un objeto blob almacenado en la base de datos, leer su contenido y crear un archivo en disco (exportar un dato binario). En el ejemplo, se guarda un archivo PDF en el directorio configurado por el objeto directory.

```
create or replace procedure guarda_libro_en_archivo(
  p_nombre_directorio in varchar2,
  p_nombre_archivo    in varchar2,
  p_libro_id          in number,
  p_longitud          out number
) is
v_blob  blob;

v_file utl_file.FILE_TYPE;
v_buffer_size number :=32767;
v_buffer RAW(32767);
v_position number := 1;

begin
  --ejecuta la consulta para extraer el blob
  select version_digital into v_blob
  from libro_blob
  where libro_id = p_libro_id;

  --abre el archivo para escribir
  v_file := utl_file.fopen(upper(p_nombre_directorio),
    p_nombre_archivo,'wb',v_buffer_size);

  p_longitud := dbms_lob.getlength(v_blob);

  --Escribe en el archivo hasta completar
  while v_position < p_longitud loop
    dbms_lob.read(v_blob,v_buffer_size,v_position,v_buffer);
    utl_file.put_raw(v_file,v_buffer,true);
    v_position := v_position+v_buffer_size;
  end loop;
  utl_file.fclose(v_file);
```

```
  -- cierra el archivo en caso de error y relanza la excepción.
  exception
    when others then
      --cerrar v_file en caso de error.
      if utl_file.is_open(v_file) then
          utl_file.fclose(v_file);
      end if;
      --muestra detalle del error
      dbms_output.put_line(dbms_utility.format_error_backtrace);
      --relanza la excepcion para que sea manejada por el
      --programa que invoque a este procedimiento.
      p_longitud := -1;
      raise;
end;
/
show errors
```

1.4. PRÁCTICA COMPLEMENTARIA.

- Continuar con las actividades de la práctica complementaria e incluir los resultados en el reporte.