

## PRÁCTICA 12

### PROGRAMACIÓN CON SQL PARTE 1

---

#### 1.1. OBJETIVO:

El alumno comprenderá los principales conceptos, así como la estructura básica para realizar la construcción de scripts SQL empleando las extensiones de programación SQL ofrecidas por los manejadores. Aplicará los conceptos referentes a disparadores para reflejar la integridad de la información y hará uso de funciones creadas por el usuario.

#### 1.2. ACTIVIDADES PREVIAS.

- Revisar el documento general de prácticas correspondiente a la práctica 12
- 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. Elementos de un programa PL/SQL***

```
--sección declarativa (opcional)
[declare]

--ordenes sql y pl/sql (obligatorio)
begin
    <instrucciones pl/sql>

--excepciones (opcional)
[exception]

--fin de bloque (obligatorio)
end;
/
```

##### ***1.3.2. Principales objetos que se pueden construir con un programa PL/SQL***

- Bloques anónimos
- Procedimientos almacenados.
- Funciones
- Paquetes PL/SQL
- Triggers

##### ***1.3.3. Variables, asignaciones y operadores.***

```
<identifier> [constant] <datatype> [not null] [:= | default <expression>];
```

Ejemplo:

```

set serveroutput on
declare
    v_uno number;
    v_dos number := 2;
    v_tres number not null := 3;
    v_cuatro number default 4;
    v_str varchar2(50);
begin
    v_uno := 1;
    v_str := v_uno||','||v_dos||','||v_tres||','||v_cuatro;
    dbms_output.put_line(v_str);
end;
/

```

- Uso de %type:

```
nombreVariable tabla.columna%type;
```

### Ejemplo:

Crear un Script PL/SQL que muestre en consola los datos del plan de estudios con id =1

```

declare
    v_id plan_estudios.plan_estudios_id%type;
    v_clave plan_estudios.clave%type;
    v_fecha plan_estudios.fecha_inicio%type;
begin
    select plan_estudios_id,clave,fecha_inicio
    into v_id,v_clave,v_fecha
    from plan_estudios
    where plan_estudios_id=1;

    dbms_output.put_line('id:      '||v_id);
    dbms_output.put_line('clave:  '||v_clave);
    dbms_output.put_line('fecha:  '||v_fecha);
end;
/

```

#### 1.3.3.1. Convenciones para el nombrado de tipos de variables.

| Estructura PL/SQL           | Convención       |
|-----------------------------|------------------|
| Variable                    | v_variable_name  |
| Constante                   | c_constant_name  |
| Parámetro en un subprograma | p_param_name     |
| Cursor                      | cur_cursor_name  |
| Excepción                   | e_exception_type |

### 1.3.4. Procedimientos almacenados.

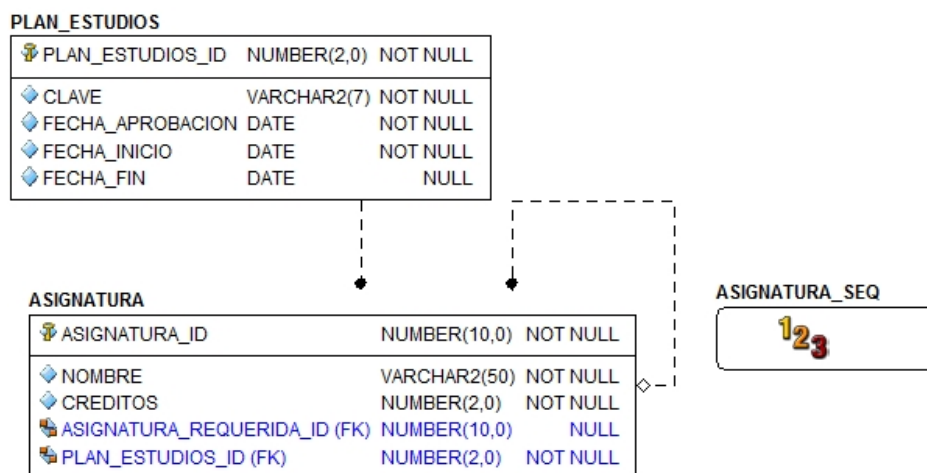
```
create [or replace] procedure [schema.]<procedure_name>
[
  (argument [{ in | out | in out }][nocopy] datatype [default expr]
  [,argument [{ in | out | in out }][nocopy] datatype [default expr]
  ]...
)
[ invoker_rights_clause ]
{ is | as }
{ pl/sql_subprogram_body | call_spec } ;
```

- El usuario que desee crear un procedimiento deberá contar con el privilegio `create procedure`.

#### Ejemplo:

Considerando el siguiente modelo relacional, crear un procedimiento almacenado llamado `creaAsignatura` que será empleado para registrar una nueva asignatura. Las especificaciones del procedimiento son:

- Recibirá como valores de entrada: nombre de la asignatura, créditos, clave del plan de estudios, nombre de la asignatura requerida en caso que la materia esté seriada con otra.
- Adicional a los parámetros anteriores, el procedimiento recibirá un parámetro de salida. El procedimiento deberá inicializar este parámetro con el identificador de la asignatura asignado empleando la secuencia.
- Notar que el procedimiento recibe la clave del plan de estudios, por lo que deberá obtener el identificador correspondiente para ser asignado en el campo `plan_estudios_id`.
- De forma similar, el procedimiento recibe de forma opcional el nombre de la asignatura requerida. El procedimiento deberá obtener el identificador correspondiente.



El código del procedimiento se muestra a continuación.

```

create or replace procedure creaAsignatura (
    v_asignatura_id out number ,v_nombre in varchar2,
    v_creditos in number,v_clave_plan in varchar2,
    v_nombre_asignatura_requerida in varchar2 default null) is

    --declaracion de variables
    v_asignatura_req_id asignatura.asignatura_requerida_id%type;
    v_plan_id plan_estudios.plan_estudios_id%type;

begin

    --generando el siguiente id de asignatura
    select asignatura_seq.nextval into v_asignatura_id
    from dual;

    --obtiene el id del plan
    select plan_estudios_id into v_plan_id
    from plan_estudios
    where clave =v_clave_plan;

    --en caso de haber asignatura requerida, obtiene el id
    if v_nombre_asignatura_requerida is not null then
        select asignatura_id into v_asignatura_req_id
        from asignatura
        where lower(nombre) = lower(v_nombre_asignatura_requerida);
    else
        v_asignatura_req_id := null;
    end if;

    --insertando a la nueva asignatura
    insert into asignatura(asignatura_id,nombre,creditos, plan_estudios_id,
        asignatura_requerida_id)
    values (v_asignatura_id,v_nombre,v_creditos,v_plan_id,v_asignatura_req_id);
end;
/
show errors

```

#### 1.3.4.1. Ejecución de un procedimiento almacenado desde SQL \*Plus

En SQL \*Plus se puede invocar un procedimiento empleando la siguiente sintaxis:

```
exec nombreProcedimiento (<param1>,<param2>,...,<paramN>)
```

#### Ejemplo:

```

set serveroutput on
variable v_id number
exec creaAsignatura(:v_id,'Electronica',12,'PL-2009');
exec dbms_output.put_line(:v_id);

```

#### 1.3.4.2. Ejecución de un procedimiento almacenado desde un programa PL/SQL anónimo.

```
set serveroutput on
declare
    v_id number;

begin

    creaAsignatura(v_id, 'Electronica', 12, 'PL-2009');
    dbms_output.put_line('La asignatura Electronica creada con id: ' || v_id);

    creaAsignatura(v_id, 'Electronica Digital', 12, 'PL-2009', 'Electronica');
    dbms_output.put_line(
        'La asignatura Electronica Digital creada con id: ' || v_id);
end;
/
```

### 1.3.5. Disparadores (triggers).

- Programa PL/SQL que se almacena en la base de datos. Su principal diferencia con respecto a otros programas como procedimientos o funciones, es que un trigger no puede ser invocado por el programador o usuario, los triggers se ejecutan cuando ocurre un evento en la base de datos.
- El usuario que desee crear un trigger deberá contar con el privilegio `create trigger`.

#### 1.3.5.1. Tipos de triggers en Oracle.

- DDL triggers
- DML triggers
- Instead of triggers
- System or database event triggers

### 1.3.6. DML triggers

Representan el tipo de trigger más común y usado. El trigger se ejecuta cuando se realiza una operación `insert`, `update` o `delete` sobre un dato que pertenece a una tabla. El trigger se asocia a la tabla (no a los datos).

Un DML trigger puede ser de 2 tipos:

- Simple DML Trigger.
- Compound DML Trigger.

#### 1.3.6.1. Simple DML trigger.

El trigger se dispara cuando ocurre alguno de los siguientes eventos:

- Antes de ejecutar la sentencia DML que provoca la activación del trigger, conocido como ***before statement trigger*** o *statement-level before trigger*
- Después de ejecutar la sentencia DML que provoca la activación del trigger, conocido como ***after statement trigger*** o *statement-level after trigger*.

- Antes de aplicar la operación DML a cada uno de los registros afectados por la sentencia que provoca la activación del trigger, conocido como **before each row trigger** o *row-level before trigger*. Notar que el trigger se dispara por cada registro afectado.
- Después de aplicar la operación DML a cada uno de los registros afectados por la sentencia que provoca la activación del trigger, conocido como **after each row trigger** o *row-level after trigger*.

### Sintaxis:

```
create [or replace] trigger <trigger_name>
{before | after}
{delete | insert | update} of <column_name1,...,> on <table_name>
[for each row]
[declare]
  [<nombre_variable> <tipo_de_dato>[:= <valor_inicial>] ]
begin
  <instrucciones pl-sql>
end;
/
```

### *Predicados condicionales:*

Permiten identificar en el código del trigger el evento particular que provocó su activación. Por ejemplo, determinar si el evento fue una operación insert, update, delete, e inclusive saber qué columna fue actualizada en el caso de una operación update.

Para identificar el evento se emplean los siguientes predicados condicionales:

- Inserting: El trigger fue activado por una operación de inserción.
- Updating: El trigger fue activado por una operación de actualización.
- Updating('<nombre\_columna>') El trigger fue activado al actualizar una determinada columna.
- Deleting: El trigger fue activado por una operación de eliminación.

### Ejemplo:

```
create or replace trigger t
  before
    insert or
    update of nombre, status_libro_id or
    delete
  on libro
begin
  case
    when inserting then
      dbms_output.put_line('insertando libro');
    when updating('nombre') then
      dbms_output.put_line('Actualizando el nombre');
    when updating('status_libro_id') then
      dbms_output.put_line('Actualizando status');
    when deleting then
      dbms_output.put_line('Eliminando');
  end case;
end;
/
```

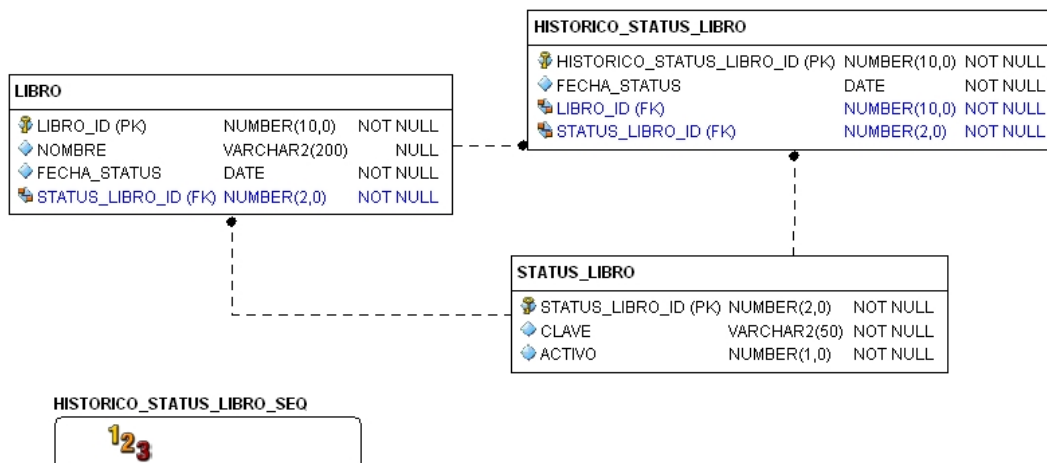
- Para triggers DML tipo Row Level existen 2 nuevas variables . :new y :old
- A nivel general, estas 2 variables hacen referencia a el registro que se está actualizando, insertando o eliminando. Sus valores dependen de la operación DML y también dependen del momento en el que el trigger se dispara: after o before. Esto se puede ilustrar en la siguiente tabla:

| Variable | Before Insert                                                 | Before update                                                                     | Before Delete                                                |
|----------|---------------------------------------------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------|
| :new     | Contiene los valores del nuevo registro que se va a insertar. | Contiene los nuevos valores del registro que se va a actualizar                   | NULL. Para una operación delete no existe un nuevo registro. |
| :old     | NULL. En una operación Insert no existe registro anterior.    | Contiene los valores viejos o anteriores del registro que se va a actualizar      | Contiene los valores del registro que se va a eliminar.      |
| Variable | After Insert                                                  | After update                                                                      | After Delete                                                 |
| :new     | Contiene los valores del registro que se acaba de insertar.   | Contiene los valores nuevos del registro que se acaba de actualizar.              | NULL. Para una operación delete no existe un nuevo registro. |
| :old     | NULL. En una operación Insert no existe registro anterior.    | Contiene los valores viejos o anteriores del registro que se acaba de actualizar. | Contiene los valores del registro que se acaba de eliminar.  |

- En resumen:
  - :old solo tiene valores para update y delete.
  - :new solo tiene valores para insert y update.
- En memoria el manejador almacena tanto el valor nuevo (:new) como el valor anterior(:old) del registro en turno. Dentro del código del trigger es posible hacer referencia a estos valores.

### Ejemplo:

Suponer las siguientes tablas:



- Generar un trigger, de tal forma que al crear o al modificar el status de un libro, se genere un registro en su histórico.

```
create or replace trigger hist_status_trigger
after insert or update of status_libro_id on libro
for each row
declare
v_status_id number(2,0);
v_fecha_status date;
v_hist_id number(10,0);
v_libro_id number(10,0);
begin
    -- obtiene el consecutivo de la secuencia
    select historico_status_libro_seq.nextval into v_hist_id from dual;

    --asigna valores a las variables con el nuevo status y fecha
    v_status_id := :new.status_libro_id;
    v_fecha_status := :new.fecha_status;
    v_libro_id := :new.libro_id;

    dbms_output.put_line('status anterior: ' || :old.status_libro_id);
    dbms_output.put_line('status nuevo: ' || :new.status_libro_id);

    dbms_output.put_line('insertando en historico, libro_id: '
        || v_libro_id || ', status_id: ' || v_status_id
        || ', fecha: ' || v_fecha_status || ', hist_id: '
        || v_hist_id);

    -- inserta en el histórico

    insert into historico_status_libro
        (historico_status_libro_id,status_libro_id,fecha_status,libro_id)
    values(v_hist_id,v_status_id,v_fecha_status,v_libro_id);
end;
```

### 1.3.6.2. Compound DML Trigger.

A diferencia de un simple DML trigger, en este caso el trigger puede dispararse cuando ocurra uno, más, o inclusive en todos los eventos mencionados anteriormente. Algunos beneficios:

- Permite realizar acciones distintas en diferentes eventos.
- Permite compartir datos. Por ejemplo, las variables que hayan sido inicializadas en un evento pueden ser leídas cuando el trigger se ejecute en un evento posterior.

### Sintaxis:



```

create [or replace] trigger <trigger_name>
for {insert | update | update of <column_name>[,<column_name>..] | delete}
on <table_name>
compound trigger

--declarative, common section

[before statement is
[declaration_statement;]
begin
    begin_statement;
end before statement;
]

[before each row is
[declaration_statement;]
begin
    begin_statement;
end before each row;
]

[after each row is
[declaration_statement;]
begin
    begin_statement;
end after each row;
]

[after statement is
[declaration_statement;]
begin
    begin_statement;
end after statement;
]
end [<trigger_name>];
/



```

### 1.3.6.3. Cuidado con tablas mutantes en DML triggers.

Este error se refiere a que una tabla no puede ser modificada o consultada dentro del código del trigger mientras esta se está actualizando (la tabla está mutando).

#### Ejemplo:

Suponer la siguiente tabla y los siguientes datos:

| LIBRO                                                                               |          |                       |
|-------------------------------------------------------------------------------------|----------|-----------------------|
|  | LIBRO_ID | NUMBER(10,0) NOT NULL |
|  | NOMBRE   | VARCHAR2(50) NOT NULL |

```

insert into libro(libro_id,nombre) values (1,'L1');
insert into libro(libro_id,nombre) values (2,'L2');
insert into libro(libro_id,nombre) values (3,'L3');

```

Suponer que se define el siguiente trigger

```
create or replace trigger trg_libro
after delete on libro
for each row
declare
  v_count number;
begin
  select count(*) into v_count
  from libro;
  dbms_output.put_line('Numero de registros en la tabla: '||v_count);
end;
/
```

Considerar que se intenta ejecutar la siguiente sentencia:

```
delete from libro where libro_id in(1,2);
```

Ocurre el siguiente error:

```
ERROR at line 1:
ORA-04091: la tabla JORGE.LIBRO esta mutando, puede que el
disparador/la
funcion no puedan verla
ORA-06512: en "JORGE.TRG_LIBRO", linea 4
ORA-04088: error durante la ejecucion del disparador
'JORGE.TRG_LIBRO'
```

- Observar que el trigger intenta obtener el número de registros de la misma tabla. Esta instrucción es invalida ya que se trata de un after insert DML trigger. El registro ha sido insertado pero el trigger pudiera no percatarse aun de que hay un nuevo registro, por que el conteo de registros puede ser incorrecto.
- Algunas opciones para corregir el problema:
  - En este caso, emplear un before insert DML trigger para que el conteo se realice antes de realizar la inserción.
  - En otros escenarios, el uso de :new y :old puede ayudar a consultar datos del registro en cuestión
  - Uso de un compound DML trigger.

### 1.3.7. 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.]<function_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
```

### 1.3.7.1. 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;
```

## 1.4. PRÁCTICA COMPLEMENTARIA.

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