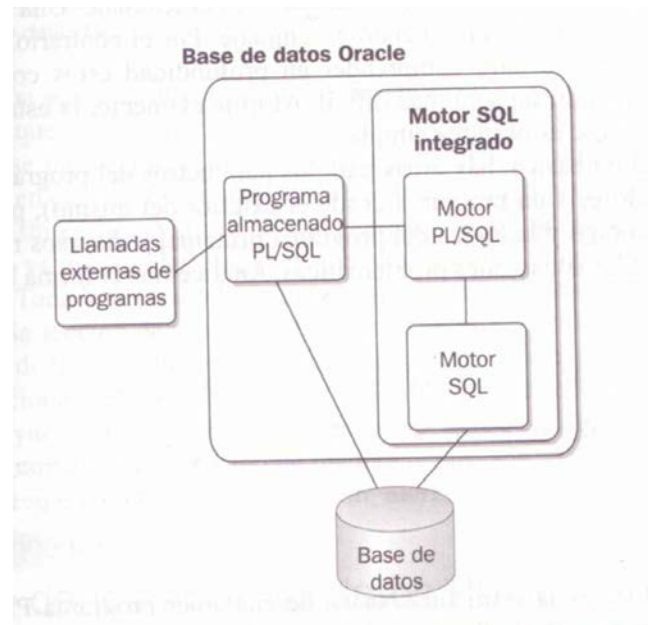


Introducción. PL/SQL



- SQL carece de control procedimental de la salida
 - No tiene mecanismos de gestión de matrices
 - No tiene estructuras de bucle ...
- PL/SQL (*Procedural Language for SQL*) extiende SQL mediante construcciones propias de los lenguajes procedimentales. La consecuencia es que se dispone de un lenguaje estructural más potente que SQL.
 - Lenguaje de programación propiedad de ORACLE
 - Entorno de programación reside directamente en la base de datos

Estructura Básica de un programa PL/SQL

```
[DECLARE]
    -- variables, constantes, ...
BEGIN
    -- órdenes SQL
    -- órdenes PL/SQL
[EXCEPTION]
    -- rutinas de tratamiento de excepciones
END;
/
```

- PL/SQL es un lenguaje de programación estructurado. La estructura básica es el **bloque**
- Un **bloque** PL/SQL está compuesto de tres partes principales
 - Sección declarativa (opcional)
 - Sección ejecutable (obligatoria)
 - Sección de excepciones (opcional). Especifica las acciones a realizar en caso de error o cuando se producen excepciones en la ejecución

Tipos de Datos PL/SQL

```
[DECLARE]
    -- variables, constantes, ...
BEGIN
    -- órdenes SQL
    -- órdenes PL/SQL
[EXCEPTION]
    -- rutinas de tratamiento de excepciones
END;
/
```

- Las **variables** se definen en la sección declarativa de los bloques donde también pueden inicializarse
 - PL/SQL no diferencia entre mayúsculas y minúsculas
- SINTAXIS:
`nombre_variable[CONSTANT] tipo_variable[NOT NULL] [:= valor];`
- También pueden definirse valores de las variables en las secciones de ejecución y de tratamiento de excepciones
`nombre_variable := valor;`

Tipos de Datos PL/SQL

```
DECLARE  
fecha DATE;  
dep_num NUMBER(2) NOT NULL := 10;  
Nombre VARCHAR2(10) := 'JUAN';  
Km_a_milla CONSTANT NUMBER := 1.4;  
nombreautor      autor.nombre%TYPE;
```

- A tener en cuenta:
 - Las variables declaradas NOT NULL siempre deben ser inicializadas
 - Si una variable no se inicializa contendrá el valor NULL
 - Las constantes debe ser inicializadas
- Las **variables** pueden utilizarse para almacenar valores devueltos o requeridos por una orden SQL
- PL/SQL soporta todos los tipos de SQL más algunos tipos especiales específicos para PL/SQL
- Las **variables** se pueden definir basándose en la definición de una columna de una TABLA
 - Utilizando el nombre de la tabla en la definición, la columna y la cadena especial de caracteres %TYPE
 - En caso de modificarse la definición de la columna, también lo hará la variable en el programa

PL/SQL. EJEMPLOS

```
SET SERVEROUTPUT ON -- se mostrarán las salidas por pantalla
```

DECLARE

A NUMBER;

```
B  REAL;
```

D VARCHAR2(40);

BEGIN

```
A := 13.6;
```

```
B := 12;
```

```
DBMS_OUTPUT.PUT_LINE('A / B = ' || A/B);
```

```
D := 'primer programa pl/sql';
```

```
DBMS_OUTPUT.PUT_LINE(D);
```

END ;

/

A / B = 1.133

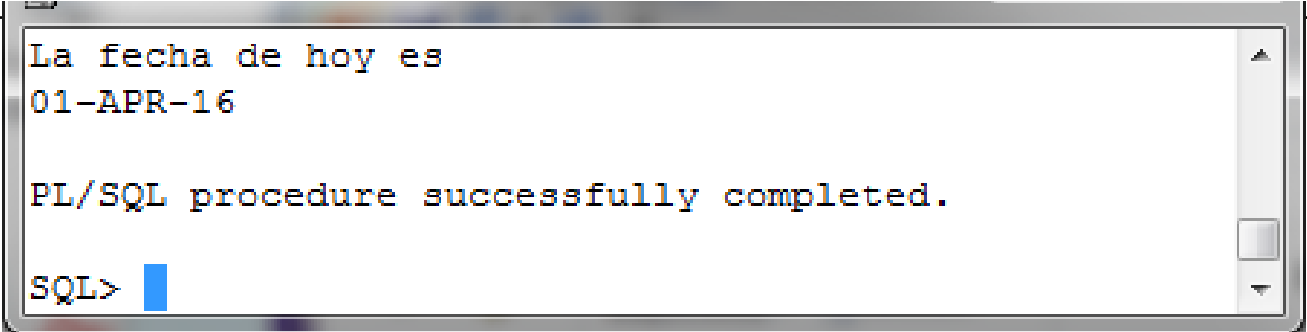
primer programa pl/sql

PL/SQL procedure successfully completed.

SQL>

PL/SQL. EJEMPLOS

```
/* Programa PL/SQL para sacar por pantalla la fecha actual*/  
SET SERVEROUTPUT ON -- se mostrarán las salidas por pantalla  
DECLARE  
    fecha_hoy DATE;  
BEGIN  
    fecha_hoy := SYSDATE;  
    DBMS_OUTPUT.PUT_LINE('La fecha de hoy es ');  
    DBMS_OUTPUT.PUT_LINE(fecha_hoy);  
END;  
/
```



La fecha de hoy es
01-APR-16

PL/SQL procedure successfully completed.

SQL>

ÓRDENES SQL EN PL/SQL. EJEMPLO

```
SET SERVEROUTPUT ON -- se mostrarán las salidas por pantalla

DECLARE

    poblacion_suc sucursal.poblacion%TYPE;
    provincia_suc  sucursal.provincia%TYPE;

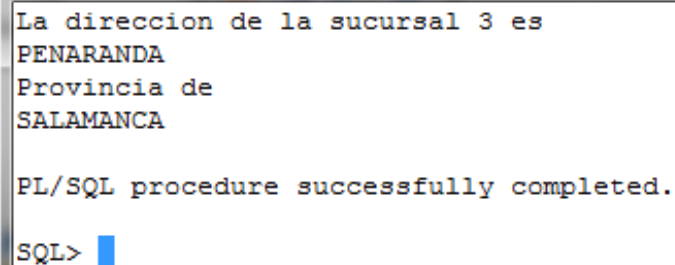
BEGIN

    SELECT poblacion, provincia
    INTO poblacion_suc, provincia_suc
    FROM sucursal
    WHERE codigo = 3;

    DBMS_OUTPUT.PUT_LINE('La direccion de la sucursal 3 es');
    DBMS_OUTPUT.PUT_LINE(poblacion_suc);
    DBMS_OUTPUT.PUT_LINE('Provincia de');
    DBMS_OUTPUT.PUT_LINE(provincia_suc);

END;

/
```



```
La direccion de la sucursal 3 es
PENARANDA
Provincia de
SALAMANCA

PL/SQL procedure successfully completed.

SQL> █
```

Tratamiento Condiciones de error en PL/SQL

```
[DECLARE]
    -- variables, constantes, ...
BEGIN
    -- Aquí va el programa
[EXCEPTION]
    -- rutinas de tratamiento de excepciones
END;
/
```

- RAISE_APPLICATION_ERROR permite que un programa PL/SQL pueda generar errores tal y como lo hace Oracle:
- SINTAXIS: RAISE_APPLICATION_ERROR(<NE> , <ME>)
 - <NE>: Número del Error, comprendido entre -20.000 y -20.999.
 - <ME>: Mensaje del Error, de longitud máxima 512 caracteres.

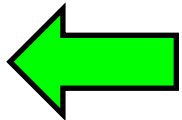
```
raise_application_error( -20123, 'Esto es un error. Ha hecho algo incorrecto' );
```
- Hay Excepciones Predefinidas que controlan errores particulares (excepto OTHERS que controla cualquier tipo de error)
- Hace que requieran el mismo tratamiento los errores definidos por el usuario (excepción declarada por el usuario) y los errores predefinidos.

Tratamiento Condiciones de error en PL/SQL

Nombre de la Excepción	Explicación	Error Oracle
NO_DATA_FOUND	Cuando una orden de selección no devuelve ninguna fila. (e.g. SELECT..INTO)	ORA-01403
TOO_MANY_ROWS	Cuando sólo debería devolverse una fila y se devuelven múltiples filas	ORA-01422
DUP_VAL_ON_INDEX	Cuando se intenta un registro en una tabla que tiene una clave principal y se viola la integridad de la entidad (Ya está insertado)	ORA-00001
VALUE_ERROR	Se genera cada vez que se produce un error aritmético, de conversión, de truncamiento o de restricciones en una orden procedimental (si es una orden SQL se produce la excepción INVALID_NUMBER)	ORA-06502
ZERO_DIVIDE	Surge al intentar dividir por cero	ORA-01476
Others	Esta excepción captura todos los errores no tratados por rutinas específicas de tratamiento de errores	No específico

Rutina de tratamiento de errores. EJEMPLO

```
DECLARE
    poblacion_suc sucursal.poblacion%TYPE;
    provincia_suc  sucursal.provincia%TYPE;
BEGIN
    SELECT poblacion, provincia
    INTO poblacion_suc, provincia_suc
    FROM sucursal
    WHERE codigo = 40;
    DBMS_OUTPUT.PUT_LINE('La dirección de la sucursal 3 es ' || poblacion_suc);
    DBMS_OUTPUT.PUT_LINE('Provincia de ' || provincia_suc);
EXCEPTION
    -- rutina genérica de tratamiento de cualquier tipo de error
    WHEN others then raise_application_error (-20100,'error#'||sqlcode||' desc#: ' || sqlerrm);
END;
/
```

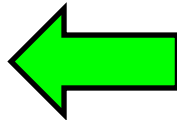


```
DECLARE
*
ERROR at line 1:
ORA-20100: error#100 desc#: ORA-01403: no data found
ORA-06512: at line 13

SQL>
```

Rutina de tratamiento de errores. EJEMPLO

```
DECLARE
    poblacion_suc sucursal.poblacion%TYPE;
    provincia_suc  sucursal.provincia%TYPE;
BEGIN
    SELECT poblacion, provincia
    INTO poblacion_suc, provincia_suc
    FROM sucursal
    WHERE codigo = 40;
    DBMS_OUTPUT.PUT_LINE('La dirección de la sucursal 3 es ' || poblacion_suc);
    DBMS_OUTPUT.PUT_LINE('Provincia de ' || provincia_suc);
EXCEPTION
    WHEN no_data_found then raise_application_error (-20052, 'No hay datos en la tabla para esa
sucursal');
    WHEN others then raise_application_error (-204582, 'Algo va pero que muy mal...');
END;
/
```



```
DECLARE
*
ERROR at line 1:
ORA-20052: No hay datos en la tabla para esa sucursal
ORA-06512: at line 12

SQL>
```

PL/SQL. Estructuras de Control

1. CONDICIONAL

- IF/THEN/END IF;
- IF/THEN/ELSE/END IF;
- IF/THEN/ELSEIF/END IF;

```
IF apellido = 'GIL' then
    sueldo := sueldo * 1.20;
END IF;
```

2. Bucle LOOP

```
LOOP
[EXIT WHEN condición]
END LOOP;
```

```
LOOP
    IF balance_cuenta >= 0 then EXIT;
    ELSE anotaciones := 'DESCUBIERTO';
    END IF;
END LOOP;
```

PL/SQL. Estructuras de Control

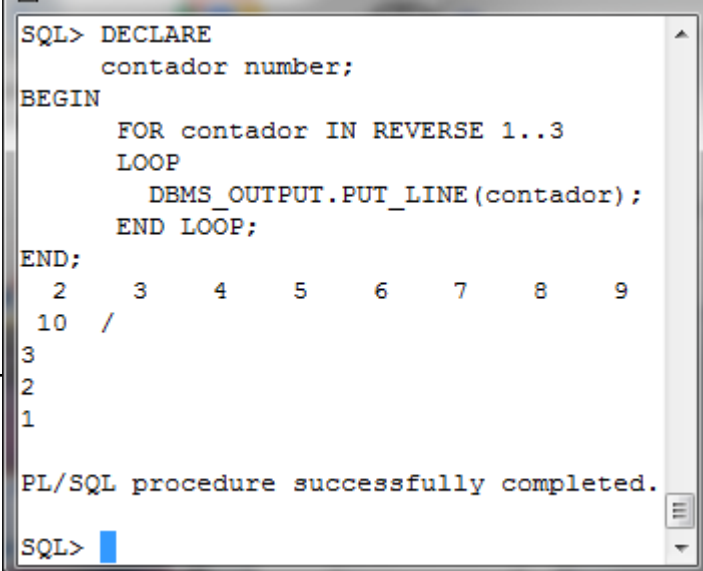
3. Bucle FOR

```
FOR contador IN valor_min..valor_max  
LOOP  
    Instrucciones;  
END LOOP;
```

```
DECLARE  
    contador number;  
BEGIN  
    FOR contador IN REVERSE 1..3  
    LOOP  
        DBMS_OUTPUT.PUT_LINE(contador);  
    END LOOP;  
END;  
/
```

4. Bucle WHILE

```
WHILE condición  
LOOP  
    Instrucciones;  
END LOOP;
```



```
SQL> DECLARE  
    contador number;  
BEGIN  
    FOR contador IN REVERSE 1..3  
    LOOP  
        DBMS_OUTPUT.PUT_LINE(contador);  
    END LOOP;  
END;  
2      3      4      5      6      7      8      9  
10 /  
3  
2  
1  
  
PL/SQL procedure successfully completed.  
SQL>
```

DISPARADORES y PLSQL

```
CREATE OR REPLACE TRIGGER historico_proyectos
AFTER DELETE OR INSERT OR UPDATE on PROYECTO
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        insert into registro (USUARIO, TABLA, COD_ITEM, ACCION) VALUES (user,
        'PROYECTO', :new.cod_proy, 'I');
    END IF;
    IF DELETING THEN
        ...
    END IF;
    IF UPDATING THEN
        ...
    END IF;
EXCEPTION
    -- rutina genérica de tratamiento de cualquier tipo de error
    WHEN others then raise_application_error (-20100,'error#'||sqlcode||'
desc#: '|| sqlerrm);

END;
```

DISPARADORES. Tablas mutantes

- Una tabla que está “mutando” es una tabla que está siendo modificada por un INSERT, un DELETE o un UPDATE o una tabla que podría modificarse por los efectos de un ON DELETE CASCADE (integridad referencial).
- Una tabla está “restringiendo” (tabla de restricción o tabla padre) si una sentencia activadora podría necesitar leerla directamente, por una sentencia SQL, o indirectamente, por una restricción de integridad referencial.
- Una tabla no se considera ni mutando ni restringiendo para los triggers. Sin embargo, hay dos restricciones importantes con respecto a las tablas que mutan o que restringen:
 - Las instrucciones de un row trigger no pueden ni leer ni modificar una tabla que está mutando.
 - Las instrucciones de un row trigger no pueden cambiar ni la clave primaria, ni claves foráneas, ni atributos únicos de tablas que estén restringiendo. Esta restricción tiene una excepción: un before row trigger disparado por un INSERT de una sola fila de una tabla con una clave foránea, puede modificar cualquier columna de la tabla primaria siempre que no se viole ninguna de las restricciones de integridad referencial.
- Los ERRORES por Tablas Mutantes se detectan y se generan en Tiempo de Ejecución y no de Compilación.