PL/SQL: Oracle Express Edition

Marcelo Paz

Administración y Programación de Base de Datos

16 de julio de 2024

Versión: 1.1.0

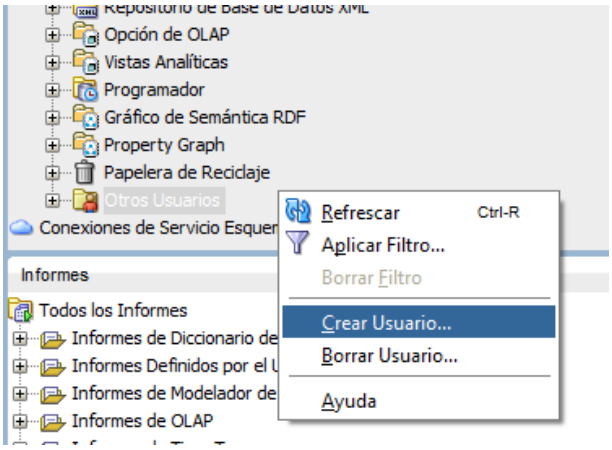
1. Anotaciones

1.1. Crear otro usuario para la base de datos

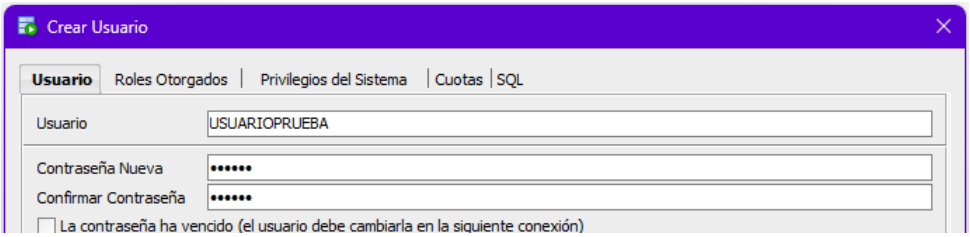
1. Lo primero es permitir el uso de scripts en la sesión actual.

```
1 ALTER SESSION SET "_ORACLE_SCRIPT" = TRUE;
```

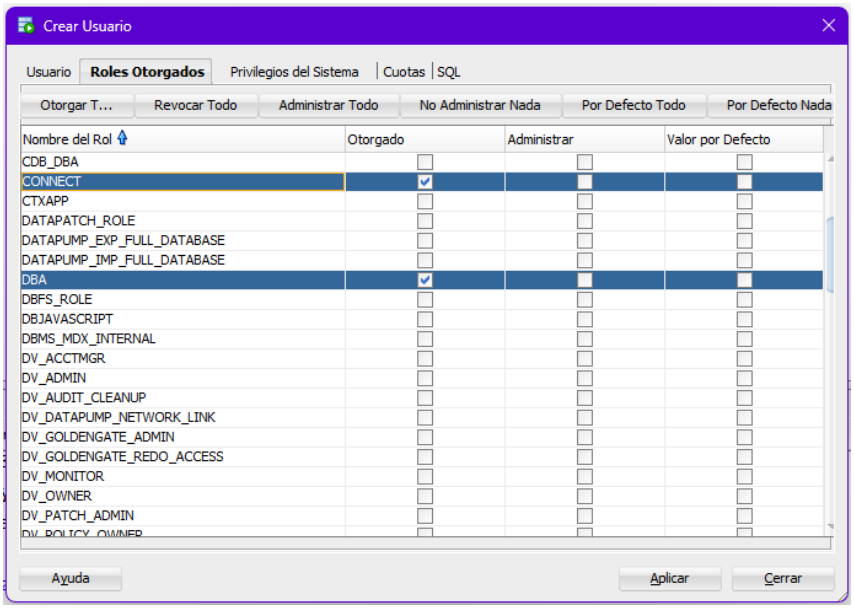
2. Luego en la conexión buscamos el directorio “Otros Usuarios”, le damos clic derecho y luego a “Crear Usuario. . . ”.



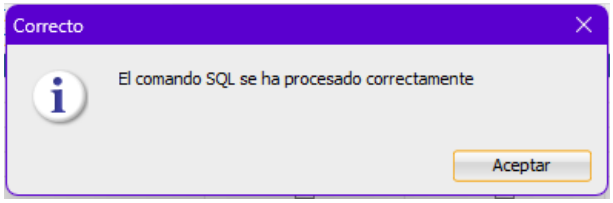
3. Le damos un nombre a nuestro nuevo usuario “USUARIOPRUEBA” y creamos una contraseña.



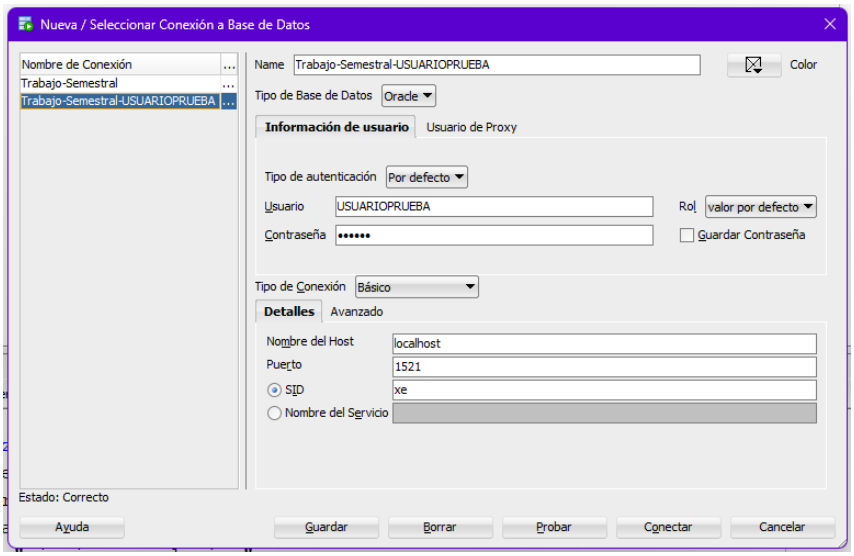
4. Nos vamos a la sección de “Roles Otorgados”, le damos a “CONNECT y DBA” y luego a “Aplicar”.



5. Debería aparecernos una ventana para confirmarnos el proceso.



6. Para finalizar al crear una nueva conexión a la BD, debemos ingresar el usuario y darle a “Probar” para validar que la conexión se realiza correctamente (Borde Inferior Izquierdo nos muestra el “Estado: Correcto”).



1.2. [Extra] Tipos de la base de datos:

- Number(tamaño[,cantidadDecimales])
- CHAR(tamaño), VARCHAR2(tamaño)
- DATE
- LONG: bloque de texto (hasta 2GB)
- LONG RAW: Datos multimedia
- ROWID: Identificador único para cada fila.

1.3. [Extra] Tipos de datos propios de PL/SQL:

- DEC, DECIMAL, REAL
- DOUBLE PRECISION, INTEGER
- BOOLEAN
- TABLE: similar a un arreglo
- RECORD: datos compuestos
- ...

1.4. [Extra] Operadores

- Asignación: :=
- Aritméticos: +, -, *, /, MOD, ** (potencia)
- Relacionales: =, <>, >, <, >=, <=, ...
- Booleanos: AND, OR, NOT, ...
- Concatenación: ||
- ...

1.5. [Extra] Palabras reservadas

- **BETWEEN:** Nos permite buscar valores entre dos valores.
- **ADD_MONTHS(FECHA, N):** Nos permite sumar/restar 'N' meses a una FECHA.
- **COALESCE(COUNT(1), 0):** Nos permite contar los valores de una columna, si no hay valores, retorna 0.
- **EXTRACT(DAY/MONTH/YEAR FROM FECHA):** Nos permite extraer el día/mes/año de una FECHA.

2. PL/SQL

De sus siglas en inglés **Procedural Language/Structured Query Language**, es un lenguaje de programación que añade funcionalidades a SQL, talez como programación por procedimientos.
Ventajas:

- Se administra centralmente dentro de la base de datos.
- Se comunica en forma nativa con objetos de la base de datos.
- Fácil de modularizar y manejar.
- Modularidad (Reusabilidad, esconder la complejidad)
  - **Procedimientos:** Aceptan y retornan cero o más valores.
  - **Funciones:** Aceptan y retornan un valor.
  - **Triggers:** Sentencias asociadas a una tabla.
  - **Paquetes:** Colección de procedimientos y funciones (definición y cuerpo).

Desventajas:

- Oracle.

2.1. Estructura de un bloque PL/SQL

Un bloque se compone de tres secciones:

1. **Declaración de variables:**
   a) Tipos Oracle
   b) Tipo exclusivo PL/SQL

PL/SQL

```
DECLARE
    -- Declaración de variables
    nombre_variable tipo_dato;
    nombre_variable tipo_dato := valor;
```

2. **Sección ejecutable:**
    - Instrucciones a ejecutar dentro del bloque.

PL/SQL
```
BEGIN
    -- Instrucciones
    nombre_variable := valor;
    nombre_variable := nombre_variable + 1;
```

3. (OPCIONAL) **Sección de manejo de excepciones:**
    - Se definen los manejadores de errores que soportará el bloque.

PL/SQL
```
EXCEPTION
    -- Manejo de excepciones
    WHEN nombre_excepcion THEN
        -- Instrucciones
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
```

4. **Importante terminar con la palabra clave:**

PL/SQL
```
END;
```

El bloque completo se ve de la siguiente manera:

PL/SQL
```
DECLARE
    -- Declaración de variables
BEGIN
    -- Instrucciones
EXCEPTION
    -- Manejo de excepciones
END;
```

2.2. Estructuras de control

- **IF-THEN-ELSE:**

PL/SQL

```
IF (condicion) THEN
    -- Instrucciones
ELSIF (condicion) THEN
    -- Instrucciones
ELSE
    -- Instrucciones
END IF;
```

- **GOTO:**

PL/SQL

```
GOTO <<etiqueta>>;
<<etiqueta>>:
    -- Instrucciones
```

- **LOOP:**

PL/SQL

```
LOOP
    -- Instrucciones
    EXIT WHEN (condicion);
END LOOP;
```

- **WHILE:**

PL/SQL

```
WHILE (condicion) LOOP
    -- Instrucciones
END LOOP;
```

- **FOR:**

PL/SQL

```
FOR variable IN rango LOOP
    -- Instrucciones
END LOOP;
```

2.3. Procedimientos almacenados con varios argumentos

Un procedimiento almacenado es un programa PL/SQL que tiene un nombre y ademas admite argumentos/parametros.

PL/SQL

```
CREATE [OR REPLACE] PROCEDURE <PR_nom>(<pa1> [IN/OUT/IN OUT] <type>,
↪  <pa2> [IN/OUT/IN OUT] <type>,...)
RETURN tipo_dato
IS
    -- Declaración de variables locales
BEGIN
    -- Instrucciones
    RETURN valor;
[EXCEPTION]
    -- Manejo de excepciones
END;
```

OBS:

- Las palabras en corchetes son opcionales.
- Las palabras en <> son las variables/tipo de datos.
- Un parametro puede ser de entrada 'IN', salida 'OUT' o ambos 'IN OUT'.

2.3.1. Ejecutar un procedimiento almacenado

PL/SQL

```
EXECUTE <PR_nom>(<valor1>, <valor2>, ...);
EXEC <PR_nom>(<valor1>, <valor2>, ...); -- Alternativa

BEGIN -- Alternativa
    <PR_nom>(<valor1>, <valor2>, ...);
END;
```

2.3.2. Borrar un procedimiento almacenado

PL/SQL

```
DROP PROCEDURE <PR_nom>;
```

2.4. Manipulación de datos

- **INSERT:** para agregar datos.

PL/SQL

```
INSERT INTO nombre_tabla (columna1, columna2, ...)
VALUES (valor1, valor2, ...);
```

*OBS: en Oracle no se puede multiples filas con un solo insert.

- **UPDATE:** para modificar datos.

PL/SQL

```
UPDATE nombre_tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condicion;
```

- **DELETE:** para eliminar datos.

PL/SQL

```
DELETE FROM nombre_tabla
WHERE condicion;
```

- **DROP:** para eliminar tablas/trigger/procedimientos/indices/vistas.

PL/SQL

```
DROP TABLE nombre_tabla;
DROP TRIGGER nombre_trigger;
DROP PROCEDURE nombre_procedimiento;
DROP INDEX nombre_indice;
DROP VIEW nombre_vista;
```

2.5. Cursores

Controladores de áreas de memoria que almacenan los resultados de una instrucción SELECT.

2.5.1. Tipos de cursores

- **Explícitos:** Se declara que es un cursor y luego se manipula para obtener 1 o varios valores.

PL/SQL

```
DECLARE
    CURSOR nombre_cursor IS
        SELECT columna1, columna2, ...
        FROM nombre_tabla
        WHERE condicion;
BEGIN
    OPEN nombre_cursor;
    FETCH nombre_cursor INTO variable1, variable2, ...;
    CLOSE nombre_cursor;
END;
```

PL/SQL: *Abrir el cursor.*

```
    OPEN nombre_cursor;
```

PL/SQL: *Recuperar los datos del Buffer.*

```
FETCH nombre_cursor INTO variable1, variable2, ...;
```

PL/SQL: *Cerrar el cursor.*

```
CLOSE nombre_cursor;
```

OBS:

- Un cursor explícito se debe abrir, recuperar y cerrar.
- Un cursor explícito admite parámetros (variables) para filtrar los datos.
- Si el cursor explícito recupera más de un valor, se debe usar un bucle.

- **Implícitos:** Solo se declara la consulta y se recupera el un solo valor, por lo que si la consulta retorna más de un valor, se generará un error/excepción.

PL/SQL: *Cerrar el cursor.*

```
DECLARE
    m amigos%ROWTYPE;
BEGIN
    SELECT * INTO m
    FROM amigos
    WHERE id = 1020;
    DBMS_OUTPUT.PUT_LINE('Mi amigo se llama'||m.nombre);
END;
```

OBS:

- Un cursor implícito no se necesita abrir, recuperar o cerrar.
- En el ejemplo anterior, se recupera un solo valor pues suponemos que el ID es la clave primaria y por lo tanto hay un unico valor posible dentro de esta.

2.5.2. Recorrer un cursor explícito

PL/SQL: *Recorrer cursor con DO-WHILE.*

```
DECLARE
    CURSOR misAmigos IS
        SELECT * FROM amigos;
    elAmigo amigos%ROWTYPE;
BEGIN
    OPEN misAmigos;
    LOOP
        FETCH misAmigos INTO elAmigo;
        EXIT WHEN misAmigos%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Mi amigo se llama -->'||elAmigo.nombre);
    END LOOP;
END;
```

PL/SQL: *Recorrer cursor con WHILE.*

```
DECLARE
    CURSOR misAmigos IS
        SELECT * FROM amigos;
    elAmigo amigos%ROWTYPE;
BEGIN
    OPEN misAmigos;
    WHILE misAmigos%FOUND LOOP
        FETCH misAmigos INTO elAmigo;
        DBMS_OUTPUT.PUT_LINE('Mi amigo se llama -->'||elAmigo.nombre);
    END LOOP;
END;
```

PL/SQL: *Recorrer cursor con FOR.*

```
DECLARE
    CURSOR misAmigos IS
        SELECT * FROM amigos;
    elAmigo amigos%ROWTYPE;
BEGIN
    FOR elAmigo IN misAmigos LOOP
        DBMS_OUTPUT.PUT_LINE('Mi amigo se llama -->'||elAmigo.nombre);
    END LOOP;
END;
```

*OBS: No se necesita abrir, recuperar o cerrar. Por lo que el utilizar un FOR es una forma implicita de recorrer el CURSOR.

2.5.3. Ejemplo

Supongamos que tenemos la siguiente tabla:

VEHICULO			
ID	Marca	Tipo	Color
1	Toyota	Auto	Rojo
2	Nissan	Auto	Verde
3	Mercedes	Camion	Blanco
4	Mercedes	Camion	Rojo

Programa 1: Mostrar los datos del vehícolo con ID = '3'.

PL/SQL

```
DECLARE
    vehiculoID VEHICULO.ID%ROWTYPE;
    vehiculoM VEHICULO.Marca%ROWTYPE;
    vehiculoT VEHICULO.Tipo%ROWTYPE;
    vehiculoC VEHICULO.Color%ROWTYPE;
BEGIN
    SELECT ID, Marca, Tipo, Color
    INTO vehiculoID, vehiculoM, vehiculoT, vehiculoC
    FROM VEHICULO
    WHERE ID = 3;
    DBMS_OUTPUT.PUT_LINE('Datos');
    DBMS_OUTPUT.PUT_LINE('      ID: ' || vehiculoID);
    DBMS_OUTPUT.PUT_LINE('    Marca: ' || vehiculoM);
    DBMS_OUTPUT.PUT_LINE('     Tipo: ' || vehiculoT);
    DBMS_OUTPUT.PUT_LINE('    Color: ' || vehiculoC);
END;
```

Resultado

```
Datos
        ID: 3
        Marca: Mercedes
        Tipo: Camion
        Color: Blanco
```

VEHICULO			
ID	Marca	Tipo	Color
1	Toyota	Auto	Rojo
2	Nissan	Auto	Verde
3	Mercedes	Camion	Blanco
4	Mercedes	Camion	Rojo

Programa 2: Mostrar la marca de los vehículos que son de color 'Rojo'.

PL/SQL

```
DECLARE
    CURSOR Cur_colorVehiculo (colorV IN VARCHAR2) IS
        SELECT Marca
        FROM VEHICULO
        WHERE Color = colorV;
    vehiculoC VEHICULO.Marca%ROWTYPE;
BEGIN
    OPEN Cur_colorVehiculo('Rojo');
    LOOP
        FETCH Cur_colorVehiculo INTO vehiculoC;
        EXIT WHEN Cur_colorVehiculo%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Marca: ' || vehiculoC);
    END LOOP;
    CLOSE Cur_colorVehiculo;
END;
```

Resultado

```
Marca: Toyota
Marca: Mercedes
```

VEHICULO			
ID	Marca	Tipo	Color
1	Toyota	Auto	Rojo
2	Nissan	Auto	Verde
3	Mercedes	Camion	Blanco
4	Mercedes	Camion	Rojo

2.6. Manejo de excepciones

En PL/SQL una advertencia o condición de error es llamada una excepción.

PL/SQL

```
DECLARE
-- Declaración de variables
BEGIN
-- Instrucciones
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No se encontraron datos');
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('División por cero');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

2.6.1. Algunas ya definidas

- `NO_DATA_FOUND (-1403)`
- `ZERO_DIVIDE (-1476)`
- `ACCESS_INTO_NULL (-6530)`
- `STORAGE_ERROR (-6500)`
- `SUBSCRIPT_BEYOND_COUNT (-6533)`

[Extra]

- **SQLCODE:** Código de error.
- **SQLERRM:** Mensaje de error.

2.7. Triggers

Es un procedimiento invocado automáticamente por el gestor de base de datos en respuesta a un evento en una tabla o vista.

2.7.1. Estructura

- **Evento:** Un cambio en la BD que dispara el trigger (INSERT, UPDATE, DELETE).
- **Condición:** Una condición que debe ser verdadera para que el trigger se ejecute.
- **Acción:** Un procedimiento que es ejecutado cuando el trigger es activado y la condición es verdadera.

En codigo:

PL/SQL

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
{BEFORE | AFTER} {INSERT | UPDATE | DELETE [OF <columna>, ...]
[OR {INSERT | UPDATE | DELETE [OF <columna>, ...]}]
ON <nombre_tabla>
[FOR EACH ROW [WHEN (<condicion>)]]
DECLARE
    -- Declaración de variables locales
BEGIN
    -- Acción
[EXCEPTION]
    -- Manejo de excepciones
END <nombre_trigger>;
```

2.7.2. Manejo de variables :OLD y :NEW

- **:OLD:** Representa el valor de la columna antes de que se realice la operación.
  - **INSERT:** No definido; todos los campos toman valor NULL.
  - **UPDATE:** Valores originales de la fila, antes de la actualización.
  - **DELETE:** Valores, antes del borrado de la fila.
- **:NEW:** Representa el valor de la columna después de que se realice la operación.
  - **INSERT:** Valores que serán insertados cuando se complete la orden.
  - **UPDATE:** Nuevos valores que serán escritos cuando se complete
  - **DELETE:** No definidos; todos los campos toman el valor NULL.

2.7.3. Estado de un trigger

- **ENABLE:** Trigger activo.

PL/SQL

```
ALTER TRIGGER <nombre_trigger> ENABLE;
```

- **DISABLE:** Trigger inactivo.

PL/SQL

```
ALTER TRIGGER <nombre_trigger> DISABLE;
```

2.7.4. Borrar un trigger

PL/SQL

```
DROP TRIGGER <nombre_trigger>;
```

2.7.5. Codigo personalizado

Oracle nos permite crear excepciones personalizadas con codigos desde -20000 hasta -20999 incluidos.

PL/SQL

```
RAISE_APPLICATION_ERROR(-20001, 'Mensaje de error');
```

2.8. Indices

El tema de los indices es optimizar la busqueda de datos en una tabla, por lo que se debe tener en cuenta que no se debe abusar de ellos, pues si se crean muchos indices, la inserción de datos se verá afectada.

2.8.1. Tipos de indices

- **Índices Implicitos:** Son creados por Oracle de forma automática, se menciono como índice implicito la 'PRIMARY KEY'.
- **Índices Explícitos:** Son creados por el usuario.

Índice Explícito

1. **TIPO I: Datos únicos (UNIQUE INDEX)**

PL/SQL

```
CREATE UNIQUE INDEX <nombre_indice> ON <nombre_tabla>(<columna>);
```

2. **TIPO 2: Datos muy repetidos (BITMAP INDEX)**

PL/SQL

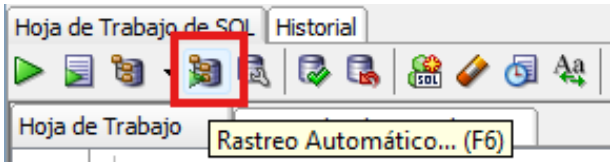
```
CREATE BITMAP INDEX <nombre_indice> ON <nombre_tabla>(<columna>);
```

3. **TIPO III: Datos normales/comunes (INDEX)**

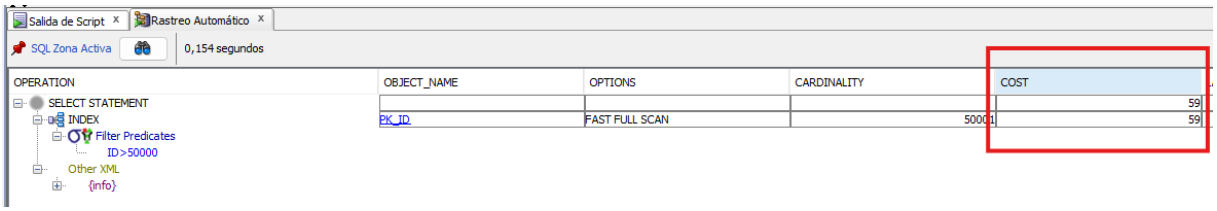
PL/SQL

```
CREATE INDEX <nombre_indice> ON <nombre_tabla>(<columna>);
```

2.8.2. Ver el costo de un índice con rastreo automatico



Ejemplo ventana de rastreo:



2.8.3. Ver los índices de una tabla

PL/SQL

```
SELECT * FROM ALL_INDEXES WHERE TABLE_NAME='nombre_tabla';
```

2.8.4. Borrar un índice

PL/SQL

```
DROP INDEX <nombre_indice>;
```