



# SQL Dinámico

# Flujo de Sentencias SQL

Todas las sentencias pasan por los siguientes pasos:

- Análisis (Parsing): verificación de la sintaxis y validación de la sentencia. Se verifica la existencia de objetos referenciados
- Acoplamiento (Binding): Se reemplazan las variables del tipo de acoplamiento
- Ejecución (Execution): Con todos los datos necesarios, Oracle ejecuta la sentencia
- Lectura (Fetching): Si existen consultas, se seleccionan y recuperan las filas resultantes de las mismas

# **SQL Estático y SQL Dinámico**

**Por lo tanto, en PL/SQL ‘normal’ sólo se admiten instrucciones DML y de control de transacciones.**

**Esto se debe a que en el momento de compilación ya se realiza la comprobación de la existencia de objetos a los que se accede, y de los permisos sobre ellos**

# SQL DINÁMICO

El SQL dinámico permite crear sentencias SQL cuya estructura puede cambiar en el momento de la ejecución.

## Características:

- Se construye y almacena como un string en la aplicación
- Permite consultas con variaciones de columnas, objetos o condiciones
- Permite la ejecución de sentencias de definición de datos (DDL), de control de datos (DC) o de control de sesión

## Existen dos maneras de ejecutar sentencias dinámicamente en PL/SQL

- Utilizando SQL dinámico nativo
- Utilizando el paquete DBMS\_SQL



# **SQL DINÁMICO NATIVO**

- El SQL dinámico nativo provee la habilidad de ejecutar dinámicamente las sentencias de SQL de la siguiente manera:
  - EXECUTE IMMEDIATE para sentencias SQL y PL/SQL anónimos
  - Cursores variables (OPEN-FOR)

# Instrucción EXECUTE IMMEDIATE

La instrucción **EXECUTE IMMEDIATE** analiza y ejecuta inmediatamente una instrucción SQL dinámica.

## SINTAXIS:

```
EXECUTE IMMEDIATE sentencia_sql_dinamica  
[INTO {variable1 [, variable2]... |  
registro}]  
[USING [IN | OUT | IN OUT] variable_enlace1  
[, [IN | OUT | IN OUT]  
variable_enlace2]...]  
[{RETURNING | RETURN} INTO variable1[,  
variable2]...];
```



# Instrucción EXECUTE IMMEDIATE

- **sentencia sql dinamica**: Representa la sentencia SQL o un bloque PL/SQL
- **INTO variable1....**: variable1 almacena una columna seleccionada
- **USING IN | OUT variable enlace1**: variable\_enlace es un argumento que se pasa a la sentencia SQL dinámica
- **INTO variable1....**: variable1 recibe el valor retornado por la sentencia SQL (para sentencias DML)

## Ejemplo 1: Utilizar SQL dinámico para ejecutar sentencias DDL en PL/SQL

```
DECLARE
    v_string  VARCHAR2 (200) ;
BEGIN
    v_string := 'DROP TABLE
    B_PLAN_PAGO';
    EXECUTE IMMEDIATE v_string;
END;
```

**Observe que la sentencia que se ejecutará dinámicamente no se termina con el punto y coma (;), excepto cuando se ejecute un procedimiento**

## Ejemplo 2

### DECLARE

```
sentencia  VARCHAR2(200);  
bloque_plsql VARCHAR2(500);  
v_id number := 10;  
type t_emp is record  
(id number,  
  nombre varchar2(30));  
emp_rec t_emp;
```

### BEGIN

```
EXECUTE IMMEDIATE 'CREATE TABLE dept (id NUMBER, nombre  
VARCHAR2(30))';  
sentencia := 'INSERT INTO dept VALUES (:1, :2)';  
EXECUTE IMMEDIATE sentencia USING 10, 'Administración';  
sentencia := 'SELECT * FROM dept WHERE id = :v_id';  
EXECUTE IMMEDIATE sentencia INTO emp_rec USING v_id;
```

END;

/

**Atención: Para consultas que devuelven más de una fila use cursores variables (OPEN FOR ...)**

## Ejemplo 3: Invocando PL/SQL dinámicamente

```
CREATE OR REPLACE PROCEDURE P_INS_AREA (PID IN OUT NUMBER,  
    PNOMBRE VARCHAR2) IS  
    V_ID NUMBER;  
BEGIN  
    SELECT NVL(MAX(ID), 0)+1 INTO V_ID FROM B_AREAS;  
    INSERT INTO B_AREAS (ID, NOMBRE_AREA, FECHA_CREA, ACTIVA)  
    VALUES (V_ID, PNOMBRE, SYSDATE, 'S');  
END;  
/
```

```
DECLARE  
    V_ID NUMBER;  
    V_NOMBRE B_AREAS.NOMBRE_AREA%TYPE := 'AREA DE I+D';  
    V_SENTENCIA VARCHAR2(500);  
BEGIN  
    V_SENTENCIA:= 'BEGIN P_INS_AREA(:a, :b); END;';  
    EXECUTE IMMEDIATE V_SENTENCIA USING IN OUT V_ID, V_NOMBRE;  
END;
```



# **PROCESAMIENTO DINÁMICO DE CONSULTAS CON VARIABLES DE TIPO CURSOR**

# Qué son las variables de tipo **CURSOR**?

- Una variable cursor es una variable de referencia. A diferencia del cursor estático con consultas ya predefinidas, ellas permiten definir cursores dinámicos que pueden abrirse para varias consultas.
- Es semejante a un CURSOR normal, pero en lugar de almacenar un CURSOR en sí, apunta a la localización de memoria donde se encuentra el cursor
- De esa manera, en lugar de nombrar explícita y estáticamente el área del CURSOR, se define un puntero a dicha área.



# Beneficios de las variables de tipo **CURSOR**

- El beneficio más importante de la variable cursor es que proporciona un mecanismo para transmitir los resultados de las consultas (las filas recuperadas por un cursor) entre los diferentes programas PL / SQL - incluso entre cliente y servidor de programas PL/SQL.
- Así una aplicación puede declarar un cursor variable del lado cliente, el subprograma puede abrir el cursor y la aplicación cliente puede recuperar nuevamente el conjunto resultante.

# **PASOS PARA UTILIZAR UNA VARIABLE DE TIPO CURSOR:**

- Definir el TIPO DE DATO cursor
- Declarar una variable del tipo definido
- Abrir la variable de tipo cursor especificando la sentencia (QUERY) a ser utilizada en el momento de la apertura
- Hacer el FETCH (o pasarlo a otro subprograma para que realice el FETCH)
- Cerrar el cursor



# 1 y 2 Declaración del Tipo de Dato y de la Variable:

**DECLARE**

**TYPE** <nombretipo> **IS REF CURSOR**

**[RETURN** <tipo\_retorno>];

<nombrevariable>                      <nombretipo>;

Ejemplo:

**DECLARE**

**TYPE**    t\_cur    **IS REF CURSOR**

**RETURN** b\_empleados%rowtype;

v\_emp\_cur    t\_cur;

### 3- Apertura de la variable **CURSOR**

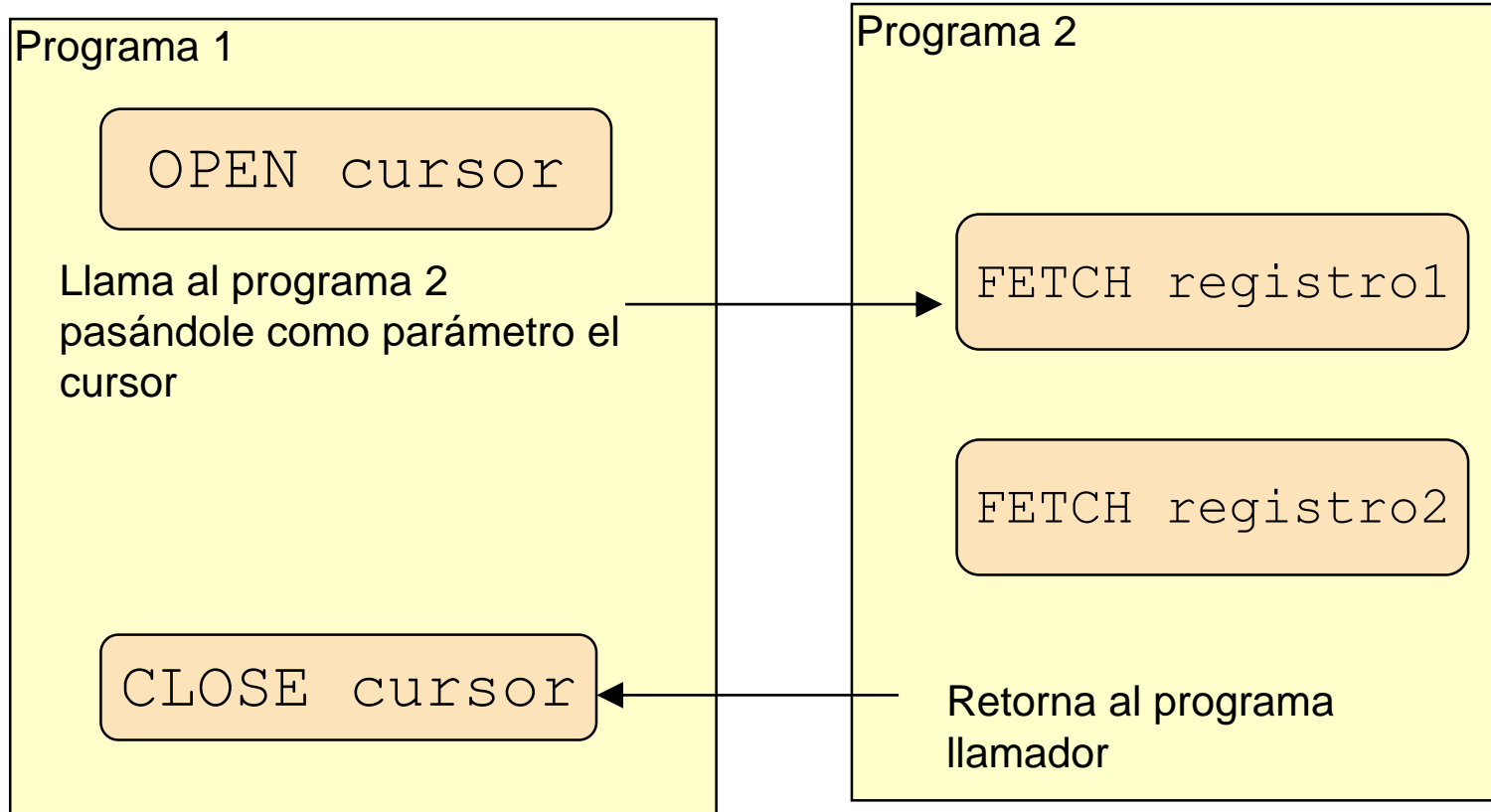
```
DECLARE  
    TYPE t_cur IS REF CURSOR  
        RETURN b_empleados%rowtype;  
    v_emp_cur t_cur;
```

```
BEGIN  
    OPEN v_emp_cur FOR  
    SELECT * FROM b_empleados;
```

## 4 y 5 - Leer y cerrar el cursor

```
DECLARE
    TYPE t_cur IS REF CURSOR
        RETURN b_empleados%rowtype;
    v_emp_cur t_cur;
    v_emp b_empleados%rowtype;
BEGIN
    OPEN v_emp_cur FOR
    SELECT * FROM b_empleados;
    FETCH v_emp_cur INTO v_emp;
    DBMS_OUTPUT.PUT_LINE(v_emp.nombre);
    CLOSE v_emp_cur;
END;
```

# Compartiendo la referencia de un cursor entre dos programas



# Clases de variables de tipo cursor

- **Variables de tipo estricto o fuertes (strong type):**

Están definidas para que devuelvan un tipo de dato específico. Ejemplo:

```
TYPE c_emp IS REF CURSOR RETURN  
b_empleados%rowtype;
```

- **Variables de tipo débil (weak):** Pueden abrirse para cualquier consulta sin importar el tipo de dato. Ejemplo:


```
TYPE c_cursor IS REF CURSOR;
```

# Restricciones para las variables de tipo cursor

- No se pueden declarar variables de tipo cursor en una especificación de paquete.
- No se pueden utilizar operadores de comparación para evaluar variables de tipo cursor para la igualdad, desigualdad o la nulidad.
- Las columnas de tablas de Base de Datos NO pueden ser definidas como de tipo cursor variable
- No se puede utilizar un cursor variable en un bucle FOR LOOP de un cursor



# **SQL Dinámico utilizando el paquete DBMS\_SQL**



El paquete DBMS\_SQL permite ejecutar la sentencia que se le pasa como un 'string' .

El string que se pasa al paquete, origina dinámicamente una instrucción SQL en tiempo de ejecución.



## Funciones incorporadas en el package

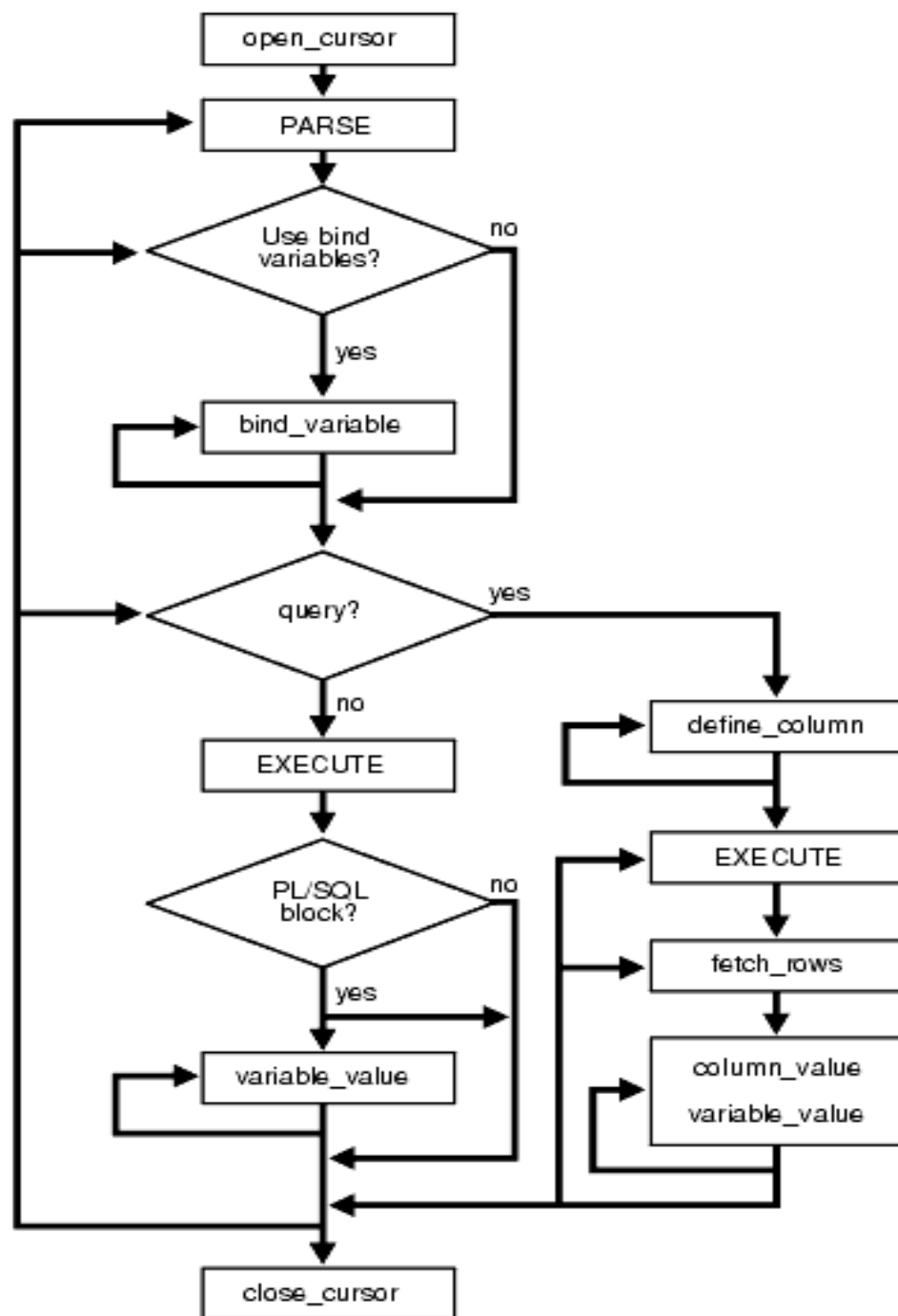
<b>OPEN_CURSOR</b>	Devuelve el identificador de un cursor
<b>PARSE</b>	Permite verificar la sintaxis de la instrucción. También ejecuta las instrucciones DDL
<b>BIND_VARIABLE</b>	Permite asociar un 'contenedor' a una variable real, y de informar el tipo y tamaño de la variable
<b>DEFINE_COLUMN</b>	Identifica el tipo y el tamaño de las variables PL/SQL que reciben los datos.
<b>COLUMN_VALUE</b>	Permite devolver datos en las variables definidas por DEFINE_COLUMN
<b>EXECUTE</b>	Ejecuta la instrucción y devuelve el número de filas procesadas
<b>FETCH_ROWS</b>	Extrae las filas
<b>CLOSE_CURSOR</b>	Cierra el cursor al finalizar el procesamiento

# Pasos generales a seguir:

- Abrir un cursor (***OPEN\_CURSOR***)
- Se analiza la cadena utilizando ***DBMS\_SQL.PARSE***
- Llevar a cabo el acoplamiento de cualquier variable de entrada mediante ***DBMS\_SQL.BIND\_VARIABLE***
- Si la instrucción es DML o DDL, ejecutarla mediante ***DBMS\_SQL.EXECUTE***

# Pasos generales a seguir:

- En caso de sentencias ***SELECT***, previamente se definen los elementos con una lista de selección.
- Se devuelve los resultados en las variables PL/SQL (***COLUMN\_VALUE***)
- Se cierra el cursor (***CLOSE\_CURSOR***)



## Ejemplo 1 (Recibe una sentencia DDL por parámetro)

```
CREATE OR REPLACE PROCEDURE EJECUTA_DDL
(p_sentencia IN VARCHAR2) IS
v_cursor      NUMBER;
BEGIN
    v_cursor := DBMS_SQL.OPEN_CURSOR;
    BEGIN
        DBMS_SQL.PARSE(v_cursor, p_sentencia,
DBMS_SQL.NATIVE);
    EXCEPTION
        WHEN OTHERS THEN
            IF SQLCODE <> -942 THEN
                RAISE_APPLICATION_ERROR(-20000, SQLERRM);
            END IF;
    END;
    DBMS_SQL.CLOSE_CURSOR(v_cursor);
END;
```

```
SQL> EXEC EJECUTA_DDL('DROP TABLE X PURGE');
```

## Ejemplo 2: DBMS\_SQL con parámetros

```
CREATE TABLE PRUEBA_PERSONA  
(ID NUMBER(7),  
APELLIDO VARCHAR2(60));
```

## Ejemplo 2: DBMS\_SQL con parámetros

```
CREATE OR REPLACE PROCEDURE P_INSERTAR_FILA(nom_tab
  VARCHAR2, pid NUMBER, pnom VARCHAR2) IS
  v_cursor INTEGER;
  v_sentencia VARCHAR2(200);
  v_cantidad NUMBER;
BEGIN
  v_sentencia := 'INSERT INTO ' || nom_tab || ' (ID,
  APELLIDO) VALUES (:cid, :cnom) ';
  v_cursor := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cursor, v_sentencia,
  DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(v_cursor, ':cid', pid);
  DBMS_SQL.BIND_VARIABLE(v_cursor, ':cnom', pnom);
  v_cantidad := DBMS_SQL.EXECUTE(v_cursor);
  DBMS_SQL.CLOSE_CURSOR(v_cursor);
  DBMS_OUTPUT.PUT_LINE(v_cantidad);
END;
```

```
SQL> EXEC P_INSERTAR_FILA ('PRUEBA_PERSONA', '3523123', 'Ruíz');
```

### Ejemplo 3: DBMS\_SQL con parámetros y retorno de variables

```
CREATE OR REPLACE PROCEDURE P_VER_TABLAS (P_ID NUMBER, P_TABLA VARCHAR2)
IS
  V_CURSOR NUMBER;
  V_SENTENCIA VARCHAR2(1000);
  V_FILAS INTEGER;
  V_NOMBRE VARCHAR2(40);
BEGIN
  V_CURSOR := DBMS_SQL.OPEN_CURSOR;
  IF P_TABLA = 'B_AREAS' THEN
    V_SENTENCIA := 'SELECT NOMBRE_AREA FROM B_AREAS WHERE ID = :PID';
  ELSIF P_TABLA = 'B_LOCALIDAD' THEN
    V_SENTENCIA := 'SELECT NOMBRE FROM B_LOCALIDAD WHERE ID = :PID';
  END IF;
  DBMS_SQL.PARSE(V_CURSOR, V_SENTENCIA, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(V_CURSOR, ':PID', P_ID);
  DBMS_SQL.DEFINE_COLUMN(V_CURSOR, 1, V_NOMBRE, 40);
  V_FILAS := DBMS_SQL.EXECUTE(V_CURSOR);
  IF DBMS_SQL.FETCH_ROWS(V_CURSOR) > 0 THEN
    DBMS_SQL.COLUMN_VALUE(V_CURSOR, 1, V_NOMBRE);
    DBMS_OUTPUT.PUT_LINE(V_NOMBRE);
  END IF;
  DBMS_SQL.CLOSE_CURSOR(V_CURSOR);
END;
```

```
SQL> EXEC P_VER_TABLAS(11, 'B_AREAS');
```



# Derechos del invocador

- El SQL Dinámico permite escribir procedimientos que relicen acciones en un esquema, pero que pueden ser llamados desde otros esquemas y operar sobre los objetos de dichos esquemas.
- Con la cláusula AUTHID, el procedimiento puede ser invocado desde otro esquema ejecutándose con los privilegios del usuario que lo invoca, de manera a no tener que calificar con prefijos los objetos referenciados.

# Derechos del invocador

```
CREATE OR REPLACE PROCEDURE BORRAR  
  (tipo_objeto IN VARCHAR2, nombre IN  
   VARCHAR2)  
  AUTHID CURRENT_USER AS  
BEGIN  
  EXECUTE IMMEDIATE 'DROP ' ||  
    tipo_objeto || ' ' || nombre;  
END;  
/
```