**Explicación del Documento sobre Paquetes en PL/SQL**

Los **paquetes en PL/SQL** permiten **agrupar** de manera lógica **procedimientos, funciones, variables, tipos de datos y cursores** en una sola unidad dentro de la base de datos. Esto mejora la organización y modularización del código, permitiendo ocultar la implementación interna y optimizando el rendimiento.

**¿Por qué el PACKAGE BODY no tiene BEGIN?**

* **Un paquete (PACKAGE) en PL/SQL es una colección de procedimientos, funciones, cursores y variables.**
* **El PACKAGE BODY solo implementa lo que se ha declarado en el PACKAGE.**
* **Como el PACKAGE BODY no ejecuta código directamente, no necesita un BEGIN.**
* **Solo los procedimientos y funciones dentro del PACKAGE BODY necesitan un BEGIN para ejecutar su lógica.**

**Ejemplo de estructura correcta de un paquete en PL/SQL**

**1️⃣ Especificación del paquete (PACKAGE)**

**Aquí solo declaramos los procedimientos y funciones, sin definir su lógica.**

**CREATE OR REPLACE PACKAGE pk AS**

**PROCEDURE datos\_cliente(p\_cod\_cliente IN clientes.cod\_cliente%TYPE);**

**END pk;**

**/**

**2️⃣ Implementación del paquete (PACKAGE BODY)**

**Aquí escribimos el código de los procedimientos.**

**CREATE OR REPLACE PACKAGE BODY pk IS -- No necesita BEGIN aquí**

**PROCEDURE datos\_cliente(p\_cod\_cliente IN clientes.cod\_cliente%TYPE) IS**

**-- Declaración de variables y cursor**

**TYPE t\_datos\_cliente IS RECORD (**

**cod\_cliente clientes.cod\_cliente%TYPE,**

**nombres clientes.nombres%TYPE,**

**distrito clientes.distrito%TYPE,**

**telefono clientes.telefono%TYPE**

**);**

**v\_aux t\_datos\_cliente;**

**CURSOR c\_datos\_cliente IS**

**SELECT cod\_cliente, nombres, distrito, telefono**

**FROM clientes**

**WHERE cod\_cliente = p\_cod\_cliente;**

**BEGIN -- Aquí empieza la ejecución del procedimiento**

**OPEN c\_datos\_cliente;**

**FETCH c\_datos\_cliente INTO v\_aux;**

**WHILE c\_datos\_cliente%FOUND LOOP**

**dbms\_output.put\_line(lpad('-', 70, '-'));**

**dbms\_output.put\_line('DATOS EMPLEADO' );**

**dbms\_output.put\_line(lpad('-', 70, '-'));**

**dbms\_output.put\_line(v\_aux.cod\_cliente||' ' || v\_aux.nombres);**

**dbms\_output.put\_line(v\_aux.distrito||' ' || v\_aux.telefono);**

**FETCH c\_datos\_cliente INTO v\_aux;**

**END LOOP;**

**CLOSE c\_datos\_cliente;**

**END datos\_cliente; -- Fin del procedimiento**

**END pk; -- Fin del paquete**

**/**

**✔ El PACKAGE BODY no tiene BEGIN porque solo define procedimientos y funciones.  
✔ Cada procedimiento y función dentro del PACKAGE BODY sí tiene su propio BEGIN y END.  
✔ El END pk; cierra el PACKAGE BODY.**

**💡 Si el PACKAGE BODY tuviera un BEGIN, significaría que queremos ejecutar código automáticamente al compilarlo, lo cual no es el propósito de un paquete. 🚀**

**📌 Conceptos Clave**

1. **División de un Paquete**
   * **Especificación (Obligatoria)**: Define los elementos públicos (funciones, procedimientos, variables, cursores) accesibles desde fuera del paquete.
   * **Cuerpo del Paquete (Opcional)**: Implementa las funciones y procedimientos declarados en la especificación. Contiene detalles internos ocultos a los usuarios externos.
2. **Beneficios de Usar Paquetes**
   * Modulariza el código, facilitando su mantenimiento.
   * Oculta detalles de implementación, protegiendo la integridad del código.
   * Mejora el rendimiento al cargar el paquete en memoria en la primera ejecución.
   * Permite la **sobrecarga de funciones** (varias funciones con el mismo nombre pero con diferentes parámetros).
3. **Cómo Crear un Paquete**
   * Se crean en **dos pasos**:
     1. **Cabecera del paquete** (PACKAGE).
     2. **Cuerpo del paquete** (PACKAGE BODY).

**📜 Reglas de Oro**

1. **Toda función o procedimiento debe estar declarado en la cabecera si se desea usar fuera del paquete.**
2. **Las variables y cursores pueden ser públicas (cabecera) o privadas (solo en el cuerpo).**
3. **Para llamar un procedimiento o función de un paquete, se usa:**

nombre\_paquete.nombre\_procedimiento(parámetros);

1. **Un paquete se ejecuta solo una vez por sesión cuando se usa por primera vez.**
2. **La sobrecarga de funciones permite reutilizar nombres de funciones con distintos parámetros.**
3. **Se puede usar una sección de inicialización en el paquete, que se ejecuta solo una vez.**
4. **Los paquetes de Oracle ofrecen funciones útiles (ej. DBMS\_OUTPUT.PUT\_LINE).**

**📌 Ejemplo de Creación de un Paquete en PL/SQL**

-- CABECERA DEL PAQUETE

CREATE OR REPLACE PACKAGE PK1 IS

-- Declaración de un procedimiento accesible desde fuera

PROCEDURE listar\_articulos(familia IN Articulos.cArtFml%TYPE);

END PK1;

/

-- CUERPO DEL PAQUETE

CREATE OR REPLACE PACKAGE BODY PK1 IS

PROCEDURE listar\_articulos(familia IN Articulos.cArtFml%TYPE) IS

codigo Articulos.cArtCdg%TYPE;

descripcion Articulos.cArtDsc%TYPE;

-- Cursor para seleccionar los artículos de una familia específica

CURSOR cArticulos IS

SELECT cArtCdg, cArtDsc FROM Articulos WHERE cArtFml = familia;

BEGIN

OPEN cArticulos;

LOOP

FETCH cArticulos INTO codigo, descripcion;

EXIT WHEN cArticulos%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(descripcion);

END LOOP;

CLOSE cArticulos;

END listar\_articulos;

END PK1;

/

Para ejecutar el procedimiento desde fuera del paquete:

BEGIN

PK1.listar\_articulos('ELEC'); -- Lista artículos de la familia "ELEC"

END;

/

**📌 Declaración de Cursores en Paquetes**

Podemos declarar cursores dentro de los paquetes de esta forma:

CREATE OR REPLACE PACKAGE empleados AS

CURSOR a RETURN empleado%ROWTYPE;

END empleados;

/

CREATE OR REPLACE PACKAGE BODY empleados AS

CURSOR a RETURN empleado%ROWTYPE IS

SELECT \* FROM empleado WHERE salario < 10000;

END empleados;

/

**📌 Paquetes de Oracle Útiles**

Algunos paquetes predefinidos en Oracle:

* **DBMS\_OUTPUT** → Imprimir mensajes en la consola (PUT\_LINE).
* **DBMS\_SQL** → Ejecutar SQL dinámicamente.
* **STANDARD** → Contiene funciones como TO\_CHAR y ABS.
* **DBMS\_STANDARD** → Manejo de errores con RAISE\_APPLICATION\_ERROR.

Con esta guía, puedes empezar a practicar la creación y uso de **paquetes en PL/SQL** con ejemplos claros y reglas esenciales. 🚀

✅ Cursor Público (en PACKAGE) → Lo pueden usar todos los procedimientos del paquete y se puede llamar desde fuera.

✅ Cursor Privado (en PACKAGE BODY) → Solo lo pueden usar los procedimientos dentro del PACKAGE BODY.

✅ Cursor Local (dentro de un PROCEDURE) → Solo lo puede usar ese procedimiento específico.

🔹 Si necesitas que varios procedimientos compartan un cursor, usa un cursor público o privado. Si el cursor es solo para un procedimiento, decláralo dentro de él. 🎯🚀

🔍 Ejemplo para Verlo Claramente

-- CABECERA DEL PAQUETE: DECLARAMOS UN CURSOR PÚBLICO

CREATE OR REPLACE PACKAGE pk\_empleados AS

-- Cursor público: Puede ser usado en cualquier procedimiento dentro del paquete

CURSOR c\_publico IS

SELECT id\_empleado, nombre, salario

FROM empleados;

-- Procedimiento público

PROCEDURE listar\_empleados\_publico;

PROCEDURE listar\_empleados\_privado;

END pk\_empleados;

/

-- CUERPO DEL PAQUETE: IMPLEMENTAMOS LOS PROCEDIMIENTOS

CREATE OR REPLACE PACKAGE BODY pk\_empleados AS

-- Cursor privado: Puede ser usado por cualquier procedimiento dentro del PACKAGE BODY

CURSOR c\_privado IS

SELECT id\_empleado, nombre, salario

FROM empleados WHERE salario > 3000;

-- Procedimiento que usa el cursor público

PROCEDURE listar\_empleados\_publico IS

v\_id empleados.id\_empleado%TYPE;

v\_nombre empleados.nombre%TYPE;

v\_salario empleados.salario%TYPE;

BEGIN

OPEN c\_publico;

LOOP

FETCH c\_publico INTO v\_id, v\_nombre, v\_salario;

EXIT WHEN c\_publico%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('PUBLICO - ID: ' || v\_id || ' - Nombre: ' || v\_nombre || ' - Salario: ' || v\_salario);

END LOOP;

CLOSE c\_publico;

END listar\_empleados\_publico;

-- Procedimiento que usa el cursor privado

PROCEDURE listar\_empleados\_privado IS

v\_id empleados.id\_empleado%TYPE;

v\_nombre empleados.nombre%TYPE;

v\_salario empleados.salario%TYPE;

BEGIN

OPEN c\_privado;

LOOP

FETCH c\_privado INTO v\_id, v\_nombre, v\_salario;

EXIT WHEN c\_privado%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('PRIVADO - ID: ' || v\_id || ' - Nombre: ' || v\_nombre || ' - Salario: ' || v\_salario);

END LOOP;

CLOSE c\_privado;

END listar\_empleados\_privado;

-- Procedimiento con un cursor local (dentro del procedimiento)

PROCEDURE listar\_empleados\_local IS

CURSOR c\_local IS

SELECT id\_empleado, nombre, salario FROM empleados WHERE salario < 2000;

v\_id empleados.id\_empleado%TYPE;

v\_nombre empleados.nombre%TYPE;

v\_salario empleados.salario%TYPE;

BEGIN

OPEN c\_local;

LOOP

FETCH c\_local INTO v\_id, v\_nombre, v\_salario;

EXIT WHEN c\_local%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('LOCAL - ID: ' || v\_id || ' - Nombre: ' || v\_nombre || ' - Salario: ' || v\_salario);

END LOOP;

CLOSE c\_local;

END listar\_empleados\_local;

END pk\_empleados;

/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/////////////////

Aquí cursor con return

**🎯 Explicación del RETURN en Cursores de PL/SQL**

En PL/SQL, cuando declaramos un cursor **en la cabecera del paquete**, usamos RETURN para definir **qué tipo de datos devuelve el cursor**.

**📌 Sintaxis del RETURN en Cursores**

CURSOR nombre\_cursor RETURN tipo\_de\_registro IS

SELECT columna1, columna2 FROM tabla;

✅ RETURN **define la estructura de los datos** que devolverá el cursor.  
✅ Es útil cuando queremos **devolver un conjunto de filas** desde un procedimiento.

**✍ Ejemplo Completo con RETURN en un Cursor Público**

🔹 **Un procedimiento usa un cursor público declarado en la cabecera del paquete.**  
🔹 **Otro procedimiento usa un cursor privado dentro del PACKAGE BODY.**

**🟢 Paso 1: Crear el Paquete**

-- CABECERA DEL PAQUETE (PACKAGE)

CREATE OR REPLACE PACKAGE pk\_empleados AS

-- 1️⃣ Definimos un tipo de registro basado en la tabla empleados

TYPE t\_empleado IS RECORD (

id empleados.id\_empleado%TYPE,

nombre empleados.nombre%TYPE,

salario empleados.salario%TYPE

);

-- 2️⃣ Definimos un cursor público con RETURN usando el tipo de registro

CURSOR c\_empleados RETURN t\_empleado IS

SELECT id\_empleado, nombre, salario FROM empleados;

-- Procedimientos

PROCEDURE listar\_empleados\_publico;

PROCEDURE buscar\_empleado(p\_id IN empleados.id\_empleado%TYPE);

END pk\_empleados;

/

🔍 **Explicación**  
✅ Creamos el **tipo de registro t\_empleado**, que representa una fila de la tabla empleados.  
✅ El **cursor c\_empleados usa RETURN t\_empleado**, indicando que devuelve registros con esa estructura.  
✅ Declaramos dos procedimientos que lo usarán.

**🟠 Paso 2: Crear el Cuerpo del Paquete**

-- CUERPO DEL PAQUETE (PACKAGE BODY)

CREATE OR REPLACE PACKAGE BODY pk\_empleados AS

-- Cursor privado: Solo se usa dentro del PACKAGE BODY

CURSOR c\_empleado\_unico(p\_id empleados.id\_empleado%TYPE) IS

SELECT id\_empleado, nombre, salario

FROM empleados WHERE id\_empleado = p\_id;

-- Procedimiento que usa el cursor público

PROCEDURE listar\_empleados\_publico IS

v\_empleado t\_empleado; -- Variable del tipo de registro del cursor

BEGIN

OPEN c\_empleados;

LOOP

FETCH c\_empleados INTO v\_empleado;

EXIT WHEN c\_empleados%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_empleado.id ||

' - Nombre: ' || v\_empleado.nombre ||

' - Salario: ' || v\_empleado.salario);

END LOOP;

CLOSE c\_empleados;

END listar\_empleados\_publico;

-- Procedimiento que usa el cursor privado

PROCEDURE buscar\_empleado(p\_id IN empleados.id\_empleado%TYPE) IS

v\_id empleados.id\_empleado%TYPE;

v\_nombre empleados.nombre%TYPE;

v\_salario empleados.salario%TYPE;

BEGIN

OPEN c\_empleado\_unico(p\_id);

FETCH c\_empleado\_unico INTO v\_id, v\_nombre, v\_salario;

IF c\_empleado\_unico%FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Empleado encontrado: ' || v\_id || ' - ' || v\_nombre || ' - ' || v\_salario);

ELSE

DBMS\_OUTPUT.PUT\_LINE('No se encontró ningún empleado con el ID: ' || p\_id);

END IF;

CLOSE c\_empleado\_unico;

END buscar\_empleado;

END pk\_empleados;

/

**🚀 Ejecutar los Procedimientos**

-- Llamar al procedimiento que usa el cursor público

BEGIN

pk\_empleados.listar\_empleados\_publico;

END;

/

-- Llamar al procedimiento que usa el cursor privado pasando un ID de empleado

BEGIN

pk\_empleados.buscar\_empleado(101);

END;

/

**📜 Explicación del Uso del RETURN en el Cursor Público**

| **Parte del Código** | **Explicación** |
| --- | --- |
| TYPE t\_empleado IS RECORD (...) | Crea un tipo de registro que almacena id, nombre y salario. |
| CURSOR c\_empleados RETURN t\_empleado IS ... | Declara un cursor que devuelve registros con la estructura de t\_empleado. |
| v\_empleado t\_empleado; | Declara una variable para almacenar registros del cursor. |
| FETCH c\_empleados INTO v\_empleado; | Extrae una fila y la guarda en la variable v\_empleado. |

**🎯 Resumen Final**

✅ **El RETURN en un cursor se usa para definir la estructura de los datos que devuelve.**  
✅ **Los cursores con RETURN permiten trabajar con registros de forma más clara y estructurada.**  
✅ **El cursor público (c\_empleados) se puede usar en cualquier procedimiento del paquete.**  
✅ **El cursor privado (c\_empleado\_unico) solo se puede usar dentro del PACKAGE BODY.**

🔹 **Con esto ya puedes usar cursores públicos con RETURN de forma correcta en tus paquetes PL/SQL. 🚀**

¡Buena pregunta, Bryan! 🔥

Si quieres almacenar **muchos registros** en un paquete, necesitas usar **colecciones** en lugar de una sola variable de tipo RECORD o %ROWTYPE.

**📌 ¿Cómo Almacenar Múltiples Registros con un Cursor en un Paquete?**

Tanto RECORD como %ROWTYPE **solo almacenan un registro a la vez**.  
Si necesitas manejar **varios registros**, debes usar una **tabla PL/SQL (colección)**.

**💡 Opciones para Almacenar Múltiples Registros**

| **Método** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| **Colecciones con TABLE y RECORD** | Se usa un tipo de tabla para almacenar múltiples registros personalizados. | TYPE t\_empleado\_tab IS TABLE OF t\_empleado; |
| **Colecciones con TABLE y %ROWTYPE** | Se usa una tabla PL/SQL con %ROWTYPE para guardar filas completas. | TYPE t\_empleado\_tab IS TABLE OF empleados%ROWTYPE; |

**🔥 Ejemplo 1: Cursor Público con RETURN y Colección en la Cabecera**

CREATE OR REPLACE PACKAGE pk\_empleados AS

-- 1️⃣ Definimos un tipo de registro con solo algunas columnas

TYPE t\_empleado IS RECORD (

id empleados.id\_empleado%TYPE,

nombre empleados.nombre%TYPE

);

-- 2️⃣ Definimos una tabla (colección) de empleados

TYPE t\_empleado\_tab IS TABLE OF t\_empleado;

-- 3️⃣ Declaramos un cursor público con `RETURN`

CURSOR c\_empleados RETURN t\_empleado IS

SELECT id\_empleado, nombre FROM empleados;

-- 4️⃣ Procedimiento que llena la colección con empleados

PROCEDURE listar\_empleados (p\_lista OUT t\_empleado\_tab);

END pk\_empleados;

/

**🎯 Ejemplo 2: Implementación en el PACKAGE BODY**

CREATE OR REPLACE PACKAGE BODY pk\_empleados AS

PROCEDURE listar\_empleados (p\_lista OUT t\_empleado\_tab) IS

v\_empleado t\_empleado;

v\_lista t\_empleado\_tab := t\_empleado\_tab(); -- Inicializa la colección

i PLS\_INTEGER := 1; -- Índice para la tabla

BEGIN

OPEN c\_empleados;

LOOP

FETCH c\_empleados INTO v\_empleado;

EXIT WHEN c\_empleados%NOTFOUND;

v\_lista.EXTEND; -- Aumenta el tamaño de la colección

v\_lista(i) := v\_empleado; -- Guarda el registro

i := i + 1;

END LOOP;

CLOSE c\_empleados;

-- Devuelve la lista completa

p\_lista := v\_lista;

END listar\_empleados;

END pk\_empleados;

/

**🛠 Ejemplo 3: Llamando al Procedimiento y Mostrando Múltiples Registros**

DECLARE

v\_empleados pk\_empleados.t\_empleado\_tab; -- Variable para almacenar la lista de empleados

BEGIN

pk\_empleados.listar\_empleados(v\_empleados);

-- Recorrer la colección e imprimir cada registro

FOR i IN 1 .. v\_empleados.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_empleados(i).id || ' - Nombre: ' || v\_empleados(i).nombre);

END LOOP;

END;

/

**🚀 Resumen**

✅ **Si quieres almacenar muchos registros**, usa **colecciones** (TABLE OF RECORD).  
✅ **Puedes devolver la colección desde un procedimiento** para usarla en PL/SQL.  
✅ **Las colecciones permiten recorrer y manejar múltiples registros fácilmente.**

📌 **Cuando tienes un cursor que devuelve muchas filas, la mejor práctica es almacenarlas en una colección para poder manipularlas.** 🔥

No puedes declarar el cursor en la **cabecera** del paquete y luego **no redeclararlo en el body**. Esto se debe a que **PL/SQL requiere que el cursor tenga su implementación en el cuerpo del paquete**.

Sin embargo, sí puedes **declararlo solo en la cabecera** si lo conviertes en un **cursor explícito con retorno de un registro (RETURN RECORD)**. Pero **no puedes usar RETURN T\_AUX (colección de registros) en la cabecera del paquete** porque los cursores solo pueden devolver **una fila a la vez**.

**🔥 SOLUCIÓN: Declarar solo en la cabecera y usar RETURN RECORD**

Si quieres que el cursor **solo esté en la cabecera del paquete** y que en el body solo lo abras, entonces usa **un cursor con RETURN RECORD** en la cabecera.

**📌 CABECERA DEL PAQUETE (Declarar solo una vez el cursor)**

CREATE OR REPLACE PACKAGE pk\_facturas AS

-- 📌 1️⃣ Tipo RECORD para una fila de datos

TYPE TD\_C\_VER\_FACTURAS IS RECORD (

COD\_FACTURA FACTURAS.COD\_FACTURA%TYPE,

FECHA\_EMISION FACTURAS.FECHA\_EMISION%TYPE,

COD\_CLIENTE FACTURAS.COD\_CLIENTE%TYPE,

COD\_PRODUCTO DETALLE\_FACTURAS.COD\_PRODUCTO%TYPE,

CANTIDAD DETALLE\_FACTURAS.CANTIDAD%TYPE,

SUBTOTAL DETALLE\_FACTURAS.SUBTOTAL%TYPE

);

-- 📌 2️⃣ Cursor que devuelve una sola fila (pero se puede recorrer con WHILE)

CURSOR c\_ver\_facturas(p\_cod\_factura FACTURAS.COD\_FACTURA%TYPE)

RETURN TD\_C\_VER\_FACTURAS;

-- 📌 3️⃣ Procedimientos

PROCEDURE ver\_facturas (p\_cod\_factura IN FACTURAS.COD\_FACTURA%TYPE);

PROCEDURE ver\_facturas;

PROCEDURE ver\_facturas (p\_cod\_cliente IN FACTURAS.COD\_CLIENTE%TYPE, p\_mostrar\_detalles IN BOOLEAN);

END pk\_facturas;

/

✅ **El cursor está solo en la cabecera**  
✅ **El cursor devuelve una fila (RETURN RECORD)**

**📌 IMPLEMENTACIÓN DEL PAQUETE (Solo se usa el cursor, no se redeclara)**

CREATE OR REPLACE PACKAGE BODY pk\_facturas AS

-- 📌 1️⃣ Procedimiento para ver una factura con WHILE

PROCEDURE ver\_facturas (p\_cod\_factura IN FACTURAS.COD\_FACTURA%TYPE) IS

v\_factura TD\_C\_VER\_FACTURAS; -- 📌 Variable para almacenar la fila del cursor

BEGIN

OPEN c\_ver\_facturas(p\_cod\_factura);

FETCH c\_ver\_facturas INTO v\_factura;

WHILE c\_ver\_facturas%FOUND LOOP

dbms\_output.put\_line('Factura: ' || v\_factura.cod\_factura);

dbms\_output.put\_line('Fecha: ' || v\_factura.fecha\_emision);

dbms\_output.put\_line('Cliente: ' || v\_factura.cod\_cliente);

dbms\_output.put\_line('Producto: ' || v\_factura.cod\_producto ||

', Cantidad: ' || v\_factura.cantidad ||

', Subtotal: ' || v\_factura.subtotal);

dbms\_output.put\_line('------------------------------------------------');

FETCH c\_ver\_facturas INTO v\_factura; -- Siguiente fila

END LOOP;

CLOSE c\_ver\_facturas;

END ver\_facturas;

-- 📌 2️⃣ Procedimiento para ver todas las facturas

PROCEDURE ver\_facturas IS

CURSOR c\_todas\_facturas IS

SELECT cod\_factura FROM facturas;

v\_cod\_factura FACTURAS.COD\_FACTURA%TYPE;

BEGIN

OPEN c\_todas\_facturas;

FETCH c\_todas\_facturas INTO v\_cod\_factura;

WHILE c\_todas\_facturas%FOUND LOOP

pk\_facturas.ver\_facturas(v\_cod\_factura);

FETCH c\_todas\_facturas INTO v\_cod\_factura;

END LOOP;

CLOSE c\_todas\_facturas;

END ver\_facturas;

-- 📌 3️⃣ Procedimiento para ver facturas de un cliente

PROCEDURE ver\_facturas (p\_cod\_cliente IN FACTURAS.COD\_CLIENTE%TYPE, p\_mostrar\_detalles IN BOOLEAN) IS

CURSOR c\_facturas\_cliente IS

SELECT cod\_factura FROM facturas WHERE cod\_cliente = p\_cod\_cliente;

v\_cod\_factura FACTURAS.COD\_FACTURA%TYPE;

BEGIN

OPEN c\_facturas\_cliente;

FETCH c\_facturas\_cliente INTO v\_cod\_factura;

WHILE c\_facturas\_cliente%FOUND LOOP

IF p\_mostrar\_detalles THEN

pk\_facturas.ver\_facturas(v\_cod\_factura);

ELSE

dbms\_output.put\_line('Factura: ' || v\_cod\_factura);

END IF;

FETCH c\_facturas\_cliente INTO v\_cod\_factura;

END LOOP;

CLOSE c\_facturas\_cliente;

END ver\_facturas;

END pk\_facturas;

/

**🔥 Beneficios de esta solución**

✅ **El cursor c\_ver\_facturas solo se declara en la cabecera.**  
✅ **No necesitas redeclararlo en el body, solo lo usas con OPEN, FETCH y CLOSE.**  
✅ **Puedes recorrerlo con WHILE o LOOP sin problemas.**  
✅ **No necesitas colecciones (TABLE OF RECORD), solo un RECORD normal.**

**⛔ Lo que NO se puede hacer**

❌ **No puedes hacer RETURN T\_AUX (una colección) en la cabecera del paquete.**  
❌ **No puedes declarar un cursor sin RETURN RECORD en la cabecera y evitar redeclararlo en el body.**  
❌ **No puedes devolver una colección directamente desde un cursor en un paquete.**

**🚀 Conclusión**

Si **quieres que el cursor solo esté en la cabecera**, usa **RETURN RECORD** y **un WHILE para recorrerlo** en el body.  
Este es el método **correcto y permitido en PL/SQL**. 🎯

Aquí tienes una tabla que resume las reglas sobre el uso de cursores en un paquete PL/SQL:

| **Regla** | **¿Se puede hacer?** | **Explicación** |
| --- | --- | --- |
| **Declarar un cursor en la cabecera y no redeclararlo en el body** | ✅ Sí, pero con RETURN RECORD | Si el cursor devuelve una sola fila a la vez (RETURN RECORD), puedes declararlo en la cabecera y usarlo en el body sin redeclararlo. |
| **Declarar un cursor en la cabecera con RETURN TABLE OF RECORD** | ❌ No | Los cursores en la cabecera solo pueden devolver **una fila a la vez** (RECORD), no una colección (TABLE OF RECORD). |
| **Usar OPEN cursor\_name, FETCH, CLOSE en los procedimientos del body** | ✅ Sí | Puedes recorrer los resultados usando un WHILE con %FOUND. |
| **Recorrer los resultados de un cursor sin usar un FETCH dentro de un loop** | ❌ No | Para leer cada fila del cursor, **debes hacer FETCH manualmente** dentro de un loop (WHILE o LOOP). |
| **Hacer BULK COLLECT INTO desde un cursor declarado en la cabecera** | ❌ No | BULK COLLECT INTO solo funciona con colecciones (TABLE OF RECORD), pero los cursores en la cabecera no pueden devolver colecciones. |
| **Devolver una colección (TABLE OF RECORD) desde un cursor en un paquete** | ❌ No | Solo se pueden devolver **una fila a la vez** (RETURN RECORD). Para colecciones, hay que definirlas y usarlas dentro del body. |
| **Llamar a un procedimiento que usa un cursor sin abrirlo manualmente** | ❌ No | Si defines un cursor en la cabecera, **debes abrirlo (OPEN) y recorrerlo (FETCH) manualmente** en el body. |
| **Usar cursores implícitos en procedimientos dentro de un paquete** | ✅ Sí | Puedes hacer SELECT INTO sin declarar un cursor, pero solo si esperas **una sola fila**. |

Con esto ya sabes exactamente qué se puede hacer y qué no dentro de un paquete PL/SQL con cursores. 🚀

* **Un cursor es un conjunto de filas** que se puede procesar **una fila a la vez**.
* **No puede almacenar colecciones**. El cursor solo maneja una fila en su memoria a la vez mientras se va recorriendo.
* **Sí puede recorrer colecciones**. Puedes usar un cursor para recorrer una colección, pero el cursor en sí solo procesa una fila a la vez, no almacena todas las filas de una colección de forma simultánea.

Esto es cierto tanto si usas el cursor dentro de un **paquete**, **procedimiento** o **función**, o si lo usas directamente en un bloque anónimo. El comportamiento es el mismo: el cursor recorre las filas de la base de datos una a una, y no puede almacenar todas esas filas en su memoria a la vez.

**Ejemplo más claro:**