## Clase: Sesión 6 – Paquetes y Buenas Prácticas en PL/SQL

### 1. ¿Qué es un paquete en PL/SQL?

Un **paquete (package)** es un contenedor que agrupa **procedimientos, funciones, variables, cursores y tipos** que pertenecen al mismo módulo funcional.
Es similar a una *librería* o un *módulo* en otros lenguajes.
**¿Por qué existen?**
Porque en una base de datos grande no puedes tener cientos de procedimientos sueltos.
Los paquetes:

- Organizan el código.
- Ocultan detalles internos.
- Mejoran el rendimiento.
- Facilitan el mantenimiento.

### 2. Estructura de un paquete

Un paquete tiene **dos partes obligatorias**:

**A) Especificación del paquete (SPEC)**
Es la *parte pública*: lo que otros programas pueden usar.

Ejemplo:

```
CREATE OR REPLACE PACKAGE pkg_empleados AS
  PROCEDURE contratar(p_id NUMBER, p_nombre VARCHAR2, p_salario NUMBER);
  FUNCTION calcular_bonus(p_salario NUMBER) RETURN NUMBER;
END pkg_empleados;
/
```

Aquí **solo declaras**, no implementas.

**Cuerpo del paquete (BODY)**
Contiene la implementación real.
Ejemplo:
CREATE OR REPLACE PACKAGE BODY pkg_empleados AS

```
  PROCEDURE contratar(p_id NUMBER, p_nombre VARCHAR2, p_salario
NUMBER) IS
  BEGIN
    INSERT INTO empleados(id, nombre, salario)
    VALUES (p_id, p_nombre, p_salario);
  END;

  FUNCTION calcular_bonus(p_salario NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN p_salario * 0.1;
  END;

END pkg_empleados;
/
```

Es la parte privada.
Puede tener variables internas invisibles para el resto del sistema.
Puede incluir funciones privadas no declaradas en el SPEC.

## Ventajas de usar paquetes (muy importantes en empresas)

**1. Modularidad**
Agrupa código relacionado: empleados, pedidos, inventario…

**2. Encapsulación**
La lógica interna queda oculta.
El programador solo ve la interfaz pública.

**3. Rendimiento**
Oracle carga el paquete completo en memoria la primera vez:
**ejecución más rápida**.

**4. Persistencia de variables**
Las variables del paquete mantienen su valor **durante toda la sesión**.

**5. Mantenimiento más fácil**

Cambiar el cuerpo no rompe a los programas que lo usan si el SPEC no cambia.

## Buenas prácticas profesionales:

**1. Centraliza la lógica en paquetes**
Evita tener procedimientos sueltos.
Ejemplos recomendados:

- pkg_clientes
- pkg_pedidos
- pkg_rrhh
- pkg_inventario

**2. No pongas lógica de negocio en triggers**
Los triggers deben ser ligeros.

Ejemplo correcto:

```
CREATE OR REPLACE TRIGGER trg_auditoria
AFTER UPDATE OF salario ON empleados
FOR EACH ROW
BEGIN
  pkg_auditoria.log_cambio_salario(:OLD.id, :OLD.salario, :NEW.salario);
END;
/
```

**3. Documenta la especificación**
Cada procedimiento debe incluir una descripción clara.

**4. Maneja excepciones dentro del paquete**
Nunca dejes excepciones sin capturar.
Ejemplo:

```
WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR(-20001, 'Empleado no encontrado');
```

## Mini-proyecto: Auditoría de cambios de salarios

**Objetivo:**
Crear un sistema que registre automáticamente cualquier cambio de salario.

**Paso 1: Crear tabla de auditoría**

```
CREATE TABLE auditoria_salarios (
  empleado_id NUMBER,
  salario_antiguo NUMBER,
  salario_nuevo NUMBER,
  fecha_cambio DATE
);
```

**Paso 2: Crear el paquete**
**Especificación:**

```
CREATE OR REPLACE PACKAGE pkg_auditoria AS
  PROCEDURE log_cambio_salario(
    p_id NUMBER,
    p_salario_ant NUMBER,
    p_salario_nue NUMBER);
END pkg_auditoria;
/
```

**Cuerpo:**

```
CREATE OR REPLACE PACKAGE BODY pkg_auditoria AS
  PROCEDURE log_cambio_salario(
    p_id NUMBER,
    p_salario_ant NUMBER,
    p_salario_nue NUMBER) IS
  BEGIN
    INSERT INTO auditoria_salarios
    VALUES(p_id, p_salario_ant, p_salario_nue, SYSDATE);
  END;
END pkg_auditoria;
/
```

**Paso 3: Crear el trigger**

```
CREATE OR REPLACE TRIGGER trg_auditoria_salario
```

```
AFTER UPDATE OF salario ON empleados
FOR EACH ROW
BEGIN
  pkg_auditoria.log_cambio_salario(:OLD.id, :OLD.salario, :NEW.salario);
END;
/
```

Ahora cualquier cambio de salario queda registrado automáticamente.

## En MySQL no hay paquetes, pero sí podemos imitarlos

En Oracle (PL/SQL) tienes *packages* (pkg_empleados, pkg_auditoria, etc.).

En **MySQL** no existen los paquetes como tal, pero sí tienes:

- **Procedimientos almacenados** (CREATE PROCEDURE)
- **Funciones almacenadas** (CREATE FUNCTION)
- **Triggers**
- **Schemas/Bases de datos** y **nombres con prefijo** (por ejemplo, auditoria_log_cambio_salario)

En MySQL "simulamos" un paquete usando **nombres organizados por prefijo**
(por ejemplo, todo lo de auditoría empieza por auditoria_...).
Ejemplo de "paquete simulado" para auditorías:

- auditoria_log_cambio_salario → procedimiento
- auditoria_log_cambio_cliente → procedimiento
- trg_auditoria_salario → trigger que llama al procedimiento

### Mini-proyecto en MySQL: Auditoría de cambios de salario

Vamos a replicar la idea de la sesión 6, pero en MySQL:
Cada vez que cambie el salario de un empleado, se registrará automáticamente
un registro con: empleado, salario antiguo, salario nuevo y fecha de cambio.

**Paso 0: contexto (tabla empleados)**
Si ya tienes una tabla empleados úsala; si no, te propongo una muy simple:

```
CREATE TABLE empleados (
  id INT PRIMARY KEY,
  nombre VARCHAR(100),
  salario DECIMAL(10,2)
```

);

**Paso 1: Crear la tabla de auditoría**
Esta tabla guardará el historial de cambios de salario.

```
CREATE TABLE auditoria_salarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  empleado_id INT,
  salario_antiguo DECIMAL(10,2),
  salario_nuevo DECIMAL(10,2),
  fecha_cambio DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

- empleado_id : referencia al empleado.
- salario_antiguo / salario_nuevo : los valores antes y después.
- fecha_cambio : se rellena sola con la fecha y hora del cambio.

**Paso 2: Crear el "procedimiento del paquete" en MySQL**

En Oracle teníamos pkg_auditoria.log_cambio_salario.
En MySQL lo haremos como un **procedimiento** llamado, por ejemplo: auditoria_log_cambio_salario

Recuerda que en MySQL, para crear procedimientos, se suele cambiar el **DELIMITER**.

```
DELIMITER //

CREATE PROCEDURE auditoria_log_cambio_salario(
   IN p_id INT,
   IN p_salario_ant DECIMAL(10,2),
   IN p_salario_nue DECIMAL(10,2)
)
BEGIN
   INSERT INTO auditoria_salarios(
      empleado_id,
      salario_antiguo,
      salario_nuevo
   ) VALUES (
```

```
        p_id,
        p_salario_ant,
        p_salario_nue
    );
END;
//

DELIMITER ;
```

Explicación:

- IN p_id : parámetro de entrada con el id del empleado.
- INSERT INTO auditoria_salarios : guarda el cambio.
- La fecha se pone sola con DEFAULT CURRENT_TIMESTAMP.

**Paso 3: Crear el trigger que llama al procedimiento**

Ahora hacemos que **MySQL dispare el procedimiento** cada vez que se modifique el salario de un empleado.

```
DELIMITER //

CREATE TRIGGER trg_auditoria_salario
AFTER UPDATE ON empleados
FOR EACH ROW
BEGIN
    -- Solo registrar si cambia realmente el salario
    IF OLD.salario <> NEW.salario THEN
        CALL auditoria_log_cambio_salario(
            OLD.id,
            OLD.salario,
            NEW.salario
        );
    END IF;
END;
//

DELIMITER ;
```

Detalles importantes:

- AFTER UPDATE ON empleados : se ejecuta tras cada actualización de la tabla empleados.
- FOR EACH ROW : se ejecuta por cada fila modificada.
- OLD.salario : el salario antes del cambio.
- NEW.salario : el salario después del cambio.
- Comprobamos IF OLD.salario <> NEW.salario para que no se dispare si no cambia el salario.

**Paso 4: Probar que funciona**

1. Insertamos un empleado:

```
INSERT INTO empleados (id, nombre, salario)
VALUES (1, 'Luis', 1500.00);
```

2. Actualizamos su salario:

```
UPDATE empleados
SET salario = 1800.00
WHERE id = 1;
```

3. Miramos la auditoría:

```
SELECT * FROM auditoria_salarios;
```

Deberías ver algo así:

| id | empleado_id | salario_antiguo | salario_nuevo | fecha_cambio |
|---|---|---|---|---|
| 1 | 1 | 1500.00 | 1800.00 | 2025-11-13 10:23:45 |

**Diferencias:**

- En PL/SQL:
  - Usas **packages** (pkg_auditoria) con **spec + body**.
  - El trigger llama a pkg_auditoria.log_cambio_salario.
- En MySQL:
  - No hay packages, pero:
    - Usas **procedimientos** (auditoria_log_cambio_salario).
    - Usas **nombres organizados** por prefijo: auditoria_....
    - El trigger llama al procedimiento con CALL ....
    -

La **filosofía es la misma**:
Centralizar la lógica en procedimientos (tu "pseudo-paquete")
y dejar los triggers como simples disparadores que solo llaman a esa lógica.

**Buenas prácticas en MySQL**

- Usa **prefijos coherentes**:
  - auditoria_..., empleados_..., clientes_...
- No metas lógica gigante dentro de triggers : mejor llamas a procedimientos.
- Documenta tus procedimientos con comentarios al principio del código.
- Agrupa por dominio funcional (igual que los paquetes):
  - Todo de RRHH con prefijo rrhh_...
  - Todo de auditoría con auditoria_...