¿Qué es un Trigger?

Un Trigger es un bloque de código PL/SQL que se ejecuta **implícitamente** (automáticamente) en respuesta a un evento en la base de datos. No se le llama directamente; simplemente "se dispara".

Componentes Clave

1. **Evento:** La acción que lo dispara (INSERT, UPDATE, DELETE, o eventos de DDL/Database como CREATE TABLE o LOGON).
2. **Momento:** Cuándo se dispara (BEFORE o AFTER del evento).
3. **Nivel:**
    - **Fila (FOR EACH ROW):** Se ejecuta una vez por cada fila afectada.
    - **Sentencia (Por defecto):** Se ejecuta una vez por la sentencia SQL completa, sin importar cuántas filas afecte.

Guía Paso a Paso: Creación de un Trigger de Auditoría

El caso de uso más común es la **auditoría**: registrar quién, cuándo y qué cambió en una tabla sensible. Usaremos un *Trigger a nivel de fila* que se dispara **antes** de una actualización.

1. Preparación: La Tabla de Auditoría

Necesitamos una tabla donde registrar los cambios.

```
CREATE TABLE auditoria_empleados (
  usuario_cambio VARCHAR2(50),
  fecha_cambio DATE,
  tipo_operacion VARCHAR2(10),
  columna_modificada VARCHAR2(50),
  valor_anterior VARCHAR2(100),
  valor_nuevo VARCHAR2(100)
);
```

2. Creación del Trigger

Este trigger se dispara **antes** de actualizar el salario de la tabla empleados.

```
CREATE OR REPLACE TRIGGER trg_auditoria_salario
BEFORE UPDATE OF salario ON empleados -- 1. Momento y Evento
FOR EACH ROW                  -- 2. Nivel: Fila por Fila
```

```
WHEN (NEW.salario <> OLD.salario)    -- 3. Condición de Disparo (Opcional)
DECLARE
  v_usuario auditoria_empleados.usuario_cambio%TYPE;
BEGIN
  -- Obtener el usuario actual
  SELECT USER INTO v_usuario FROM dual;

  -- Insertar el registro de auditoría
  INSERT INTO auditoria_empleados (
    usuario_cambio,
    fecha_cambio,
    tipo_operacion,
    columna_modificada,
    valor_anterior,
    valor_nuevo
  )
  VALUES (
    v_usuario,
    SYSDATE,
    'UPDATE',
    'SALARIO',
    :OLD.salario, -- Referencia al valor ANTES del cambio
    :NEW.salario  -- Referencia al valor DESPUÉS del cambio
  );

END;
/
```

Nota Clave: Dentro de un trigger a nivel de fila, se usan las pseudo-registros **:OLD** y **:NEW**:

- **:OLD**: Los datos de la fila **antes** del UPDATE o DELETE.
- **:NEW**: Los datos de la fila **después** del INSERT o UPDATE.

3. Prueba y Verificación

Ahora, si actualizamos un salario, el trigger se disparará automáticamente.

```
-- 1. Verificar el salario actual de Ana Gerente (ID 2)
SELECT salario FROM empleados WHERE emp_id = 2; -- Salario: 8000

-- 2. Ejecutar la actualización (El trigger se dispara antes)
UPDATE empleados
SET salario = 8800
WHERE emp_id = 2;

COMMIT;

-- 3. Verificar el registro en la tabla de auditoría
SELECT * FROM auditoria_empleados ORDER BY fecha_cambio DESC;
```

Deberías ver una fila en auditoria_empleados que dice que el salario pasó de **8000** a **8800**.

Casos de Uso y Limitaciones

Usos Comunes de Triggers

Objetivo	Evento	Momento	Ejemplo
Auditoría	UPDATE, DELETE	BEFORE o AFTER	Registrar valores antiguos en una tabla de LOG.
Validación de Datos	INSERT, UPDATE	BEFORE	Asegurar que el salario no se puede reducir a menos del 80% del valor anterior.
Integridad Referencial	DELETE	AFTER	Si se borra un cliente, actualizar el estado de sus pedidos a "cancelado".
Generación de ID	INSERT	BEFORE	Asignar automáticamente un ID a una columna usando una secuencia si el ID viene nulo (obsoleto con IDENTITY).

Exportar a Hojas de cálculo

Riesgos y Limitaciones

1. **"Mutating Table" Error:** Es el error más común. Un trigger no puede consultar o modificar la tabla que lo disparó. Esto sucede porque la tabla está en un estado transitorio durante la ejecución.

2. **Complejidad y Mantenimiento:** Los triggers ocultan la lógica. Es difícil de depurar; un error en un UPDATE simple podría ser causado por un trigger en una tabla secundaria que nadie recuerda.
3. **Rendimiento:** Un trigger a nivel de fila en una tabla que recibe miles de INSERTs por minuto añade sobrecarga. Debe ser lo más eficiente posible.

Triggers DDL y de Base de Datos

Además de los eventos DML (datos), Oracle permite triggers para eventos del sistema:

- **DDL (Data Definition Language):** Disparadores cuando se ejecutan CREATE, DROP o ALTER. (Ej: Impedir que se borren tablas en horario laboral).
- **Database Events:** Disparadores en LOGON (cuando un usuario se conecta), LOGOFF (cuando se desconecta) o SERVERERROR (cuando ocurre un error de la BD).

Estructura Básica: El Bloque EXCEPTION

Cualquier bloque de código PL/SQL (un procedimiento, una función o un bloque anónimo BEGIN...END) puede incluir una sección opcional EXCEPTION al final.

La estructura es:

```
[Declaraciones de variables/cursores (DECLARE)]
BEGIN
  -- Lógica del negocio: Aquí es donde puede ocurrir el error.
  ...
EXCEPTION
  -- Manejador 1: Para errores específicos (Ej. NO_DATA_FOUND)
  WHEN nombre_excepcion_o_codigo_error THEN
    -- Código para manejar ese error
    ...
  -- Manejador 2: Para cualquier otro error no capturado
  WHEN OTHERS THEN
    -- Código para manejar errores inesperados
    ...
    RAISE; -- Opcional: relanza la excepción
END;
/
```

Tipos de Excepciones

Oracle clasifica las excepciones en tres categorías principales:

1. Predefinidas (Nombradas)

Son los errores más comunes de Oracle que ya tienen un nombre asociado en PL/SQL. Son las más fáciles de manejar.

Excepción Predefinida	**Código de Error**	**Descripción Típica**
NO_DATA_FOUND	ORA-01403	Un SELECT INTO no devolvió ninguna fila.
TOO_MANY_ROWS	ORA-01422	Un SELECT INTO devolvió más de una fila.
DUP_VAL_ON_INDEX	ORA-00001	Intentaste insertar un valor que ya existe en una restricción **UNIQUE** o **PRIMARY KEY**.
ZERO_DIVIDE	ORA-01476	Intentaste dividir por cero (x / 0).
VALUE_ERROR	ORA-06502	Error de conversión de datos (ej. intentar meter 'ABC' en un campo NUMBER).

2. No Predefinidas (Nombradas por el Usuario)

Son errores de Oracle que no tienen un nombre asociado en PL/SQL (ej. la violación de un *Foreign Key* es ORA-02291). Puedes asociarles un nombre usando una PRAGMA (directiva de compilador).

3. Definidas por el Usuario (Custom)

Son excepciones que tú creas y lanzas manualmente cuando tu lógica de negocio se rompe (ej. Un cliente intenta comprar más de 10 productos en un solo pedido).

Ejemplo Práctico: Capturando Errores Comunes

Usaremos un bloque de ejemplo para manejar dos situaciones: no encontrar datos y un error matemático.

```
DECLARE
  v_salario empleados.salario%TYPE;
  v_divisor NUMBER := 0; -- Lo usaremos para causar ZERO_DIVIDE
BEGIN
  -- 1. Intento de división por cero (Causará ZERO_DIVIDE)
  -- DBMS_OUTPUT.PUT_LINE('Resultado: ' || (100 / v_divisor));
```

```
  -- 2. Intento de buscar un ID que no existe (Causará NO_DATA_FOUND)
  SELECT salario INTO v_salario
  FROM empleados
  WHERE emp_id = 99999; -- ID inexistente

  -- Si el código llega aquí, el proceso fue exitoso
  DBMS_OUTPUT.PUT_LINE('Salario encontrado: ' || v_salario);

EXCEPTION
  -- Manejador A: Captura la falta de datos
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('⚠️ Aviso: Empleado no encontrado. No se puede continuar.');
    v_salario := 0; -- Asigna un valor seguro

  -- Manejador B: Captura el error matemático
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('❌ Error Crítico: Se intentó dividir por cero.');

  -- Manejador C: Captura todos los demás errores inesperados
  WHEN OTHERS THEN
    -- SQLCODE: Devuelve el código de error numérico de Oracle (-01476, -01403, etc.)
    -- SQLERRM: Devuelve el mensaje de error de Oracle asociado al código
    DBMS_OUTPUT.PUT_LINE('🚨 Error inesperado (' || SQLCODE || '): ' || SQLERRM);
    -- Opcional: RAISE; para relanzar la excepción

END;
/
```

Consejo: El manejador WHEN OTHERS siempre debe ser el **último** en la lista.

Lanzamiento de Excepciones Personalizadas

A menudo, el error no es técnico, sino lógico (violación de una regla de negocio). Para esto, declaras tu propia excepción y la lanzas con la sentencia RAISE.

Ejemplo de Excepción de Negocio

```
DECLARE
  e_salario_demasiado_alto EXCEPTION; -- 1. Declara la excepción
  v_salario_propuesto NUMBER := 15000;
BEGIN
  IF v_salario_propuesto > 10000 THEN
    RAISE e_salario_demasiado_alto; -- 2. Lanza la excepción
  END IF;

  -- Si no se lanza, el código continúa
  DBMS_OUTPUT.PUT_LINE('Salario Aceptado.');

EXCEPTION
  WHEN e_salario_demasiado_alto THEN -- 3. Captura la excepción nombrada
    DBMS_OUTPUT.PUT_LINE(' 🚫 ERROR DE NEGOCIO: Salario propuesto excede el límite de 10.000.');

  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error general inesperado.');
END;
/
```

Funciones Útiles para Excepciones

- **SQLCODE**: Devuelve el código de error numérico de la última excepción (ej. -1403 para NO_DATA_FOUND).
- **SQLERRM**: Devuelve el mensaje de texto asociado a la última excepción (ej. ORA-01403: no data found).
- **RAISE_APPLICATION_ERROR**: Se usa típicamente en procedimientos y triggers para lanzar una excepción personalizada que el usuario pueda ver, utilizando un rango de código de error definido por el usuario (-20000 a -20999).

La gestión de excepciones asegura que un pequeño error no detenga todo un proceso, permitiendo que tu programa tome decisiones de recuperación inteligentes (ej. registrar el error, notificar al usuario, o asignar un valor por defecto).