Los **disparadores de la base de datos** (triggers) son procedimientos **PL/SQL** que se asocian a tablas de Oracle y se ejecutan automáticamente cuando se realiza una operación específica sobre esa tabla (como una inserción, eliminación o actualización). Estos se usan para garantizar que se cumplan ciertas reglas o restricciones en la base de datos de manera automática y transparente para el usuario. Aquí te explico los conceptos clave de los triggers, cómo usarlos y las reglas a seguir para implementarlos correctamente:

**Conceptos clave de los triggers**

1. **Qué son los triggers**:
   * Un trigger es un conjunto de acciones que se ejecutan **automáticamente** en respuesta a un evento (como **INSERT**, **DELETE** o **UPDATE**) en una tabla de la base de datos.
   * Se usan para implementar **restricciones de integridad**, **reglas de seguridad** y **acciones administrativas**.
   * No se deben usar para duplicar funcionalidades ya existentes en la base de datos (como la validación de integridad).
2. **Tipos de triggers**:
   * **BEFORE**: El trigger se ejecuta **antes** de que se ejecute la operación en la base de datos.
   * **AFTER**: El trigger se ejecuta **después** de que se haya realizado la operación.
   * **INSTEAD OF**: Usado principalmente para vistas, reemplaza la operación DML en lugar de ejecutarla.
3. **Eventos que disparan un trigger**:
   * **INSERT**, **UPDATE** y **DELETE** son los eventos más comunes.
   * También se pueden especificar las columnas afectadas (por ejemplo, un trigger para un UPDATE que solo se dispare si cambia una columna específica).
4. **Niveles de ejecución del trigger**:
   * **FOR EACH STATEMENT**: El trigger se ejecuta una sola vez para toda la operación, independientemente de cuántas filas afecte.
   * **FOR EACH ROW**: El trigger se ejecuta una vez por cada fila afectada. Se usa para triggers que deben trabajar con los valores de cada fila específica.
5. **Uso de NEW y OLD**:
   * En triggers de **fila**, se puede usar :NEW (para los valores **nuevos**) y :OLD (para los valores **anteriores**) para acceder a los datos antes y después de la operación.
   * Ejemplo: IF :NEW.salario < :OLD.salario para verificar si el salario ha disminuido.
6. **Cláusula WHEN**:
   * Se usa para especificar una condición que debe cumplirse para que el trigger se ejecute, pero solo en triggers a nivel de fila. La condición debe ser una **expresión SQL**.

**Sintaxis para crear un trigger**

CREATE [OR REPLACE] TRIGGER nombre\_trigger

{BEFORE | AFTER}

{INSERT | UPDATE | DELETE [OF columnas...]}

ON tabla

[REFERENCING OLD AS alias NEW AS alias]

[FOR EACH {STATEMENT | ROW [WHEN condición]}]

DECLARE

-- Declaraciones opcionales

BEGIN

-- Sentencias del trigger

END;

**Ejemplo de uso de triggers**

**1. Crear una tabla para controlar qué usuarios modifican una tabla:**

CREATE TABLE controla\_usuarios (

usuario VARCHAR2(17),

operacion VARCHAR2(6),

fecha DATE

);

**2. Disparadores para controlar las operaciones sobre la tabla emp:**

* **Trigger para actualizar un empleado**:

CREATE TRIGGER actualiza\_emple

BEFORE UPDATE ON EMP

DECLARE

nom\_usuario VARCHAR2(30);

BEGIN

nom\_usuario := USER;

INSERT INTO controla\_usuarios

VALUES (nom\_usuario, 'UPDATE', sysdate);

END;

/

* **Trigger para insertar un nuevo empleado**:

CREATE TRIGGER inserta\_emp

BEFORE INSERT ON EMP

DECLARE

nom\_usuario VARCHAR2(30);

BEGIN

nom\_usuario := USER;

INSERT INTO controla\_usuarios

VALUES (nom\_usuario, 'INSERT', sysdate);

END;

/

* **Trigger para borrar un empleado**:

CREATE TRIGGER borra\_emp

BEFORE DELETE ON EMP

DECLARE

nom\_usuario VARCHAR2(30);

BEGIN

nom\_usuario := USER;

INSERT INTO controla\_usuarios

VALUES (nom\_usuario, 'DELETE', sysdate);

END;

/

**3. Tipos de trigger y su orden de ejecución:**

* **BEFORE** se ejecuta antes de que la operación en la tabla se realice.
* **AFTER** se ejecuta después de que la operación en la tabla haya sido completada.
* Si hay varios triggers, se ejecutan en este orden:
  1. BEFORE de tipo **statement**.
  2. BEFORE de tipo **row**.
  3. La operación **DML** (INSERT, UPDATE, DELETE).
  4. AFTER de tipo **row**.
  5. AFTER de tipo **statement**.

**4. Múltiples eventos en un solo trigger:**

Puedes especificar múltiples eventos para que un único trigger se dispare por diferentes operaciones, como **INSERT**, **UPDATE** o **DELETE**. Ejemplo:

CREATE TRIGGER ejemplo

BEFORE DELETE OR UPDATE ON tabla

FOR EACH ROW

BEGIN

-- Acciones

END;

**5. Restricciones sobre los triggers:**

* No puedes usar sentencias de control de transacciones (COMMIT, ROLLBACK) dentro de un trigger.
* Los triggers **INSTEAD OF** solo se pueden usar con vistas, y no se puede usar la cláusula WHEN.

**Reglas de oro al usar triggers:**

1. **Evitar el uso excesivo de triggers**, ya que pueden hacer que la base de datos sea más compleja de mantener.
2. **Evitar recursividad**: Ten cuidado con los triggers que pueden llamarse entre sí (por ejemplo, un trigger que modifique la tabla y dispare otro trigger).
3. **Evitar el uso innecesario**: No utilices triggers para realizar operaciones que ya estén cubiertas por las restricciones de la base de datos o lógica de negocio.
4. **Verificar el impacto en el rendimiento**: Los triggers pueden afectar el rendimiento de las transacciones, así que úsalos con cautela en sistemas de alto tráfico.

Siguiendo esta guía, puedes usar triggers de manera efectiva para automatizar operaciones en la base de datos y garantizar la integridad de los datos.

En PL/SQL, la diferencia entre FOR EACH ROW y FOR EACH STATEMENT en un *trigger* depende de cuándo deseas que se ejecute el código del *trigger*:

1. **FOR EACH ROW**: Este tipo de *trigger* se ejecuta una vez por cada fila afectada por la operación de modificación (INSERT, UPDATE o DELETE). Es útil cuando necesitas que se realicen acciones específicas para cada fila que se está insertando, actualizando o eliminando. Por ejemplo, si estás calculando un valor o validando datos en una fila particular, usarías FOR EACH ROW.

Ejemplo de trigger con FOR EACH ROW:

CREATE OR REPLACE TRIGGER after\_update\_emp

AFTER UPDATE ON employees

FOR EACH ROW

BEGIN

IF :NEW.salary > 10000 THEN

DBMS\_OUTPUT.PUT\_LINE('Salary updated: ' || :NEW.salary);

END IF;

END;

1. **FOR EACH STATEMENT**: Este tipo de *trigger* se ejecuta solo una vez, independientemente de cuántas filas estén afectadas por la operación. Es útil cuando no es necesario procesar fila por fila, sino cuando se necesita realizar una acción global sobre toda la operación (por ejemplo, actualizar estadísticas o realizar una acción solo una vez, aunque se afecten varias filas).

Ejemplo de trigger con FOR EACH STATEMENT:

CREATE OR REPLACE TRIGGER after\_delete\_emp

AFTER DELETE ON employees

FOR EACH STATEMENT

BEGIN

DBMS\_OUTPUT.PUT\_LINE('A delete operation was performed on the employees table.');

END;

**¿Cuándo usar cada uno?**

* **Usa FOR EACH ROW** cuando necesites ejecutar acciones para cada fila afectada por la operación.
* **Usa FOR EACH STATEMENT** cuando quieras que la acción del *trigger* se ejecute una sola vez, independientemente de cuántas filas estén involucradas en la operación.

Si no estás seguro, piensa en si el cambio o validación depende de cada fila individual o de la operación completa.