Procedimiento

Ejemplo 1

```
DELIMITER //
CREATE PROCEDURE registrar_auditoria_libro(
  IN p_id_libro INT,
  IN p_accion VARCHAR(100)
)
BEGIN
  INSERT INTO auditoria_libros (id_libro, accion, usuario)
  VALUES (p_id_libro, p_accion, CURRENT_USER());
END //
DELIMITER ;
```

Ejmeplo 2

```
DELIMITER //
CREATE PROCEDURE actualizar_stock_libro(
  IN p_id_libro INT,
  IN p_nuevo_stock INT
)
BEGIN
  UPDATE libros
  SET stock = p_nuevo_stock
  WHERE id_libro = p_id_libro;
  CALL registrar_auditoria_libro(p_id_libro, CONCAT('Stock
actualizado a ', p_nuevo_stock));
END //
DELIMITER ;
```

Funcion

Ejemplo 1

```
DELIMITER //
CREATE FUNCTION total_stock_libros()
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE total INT;
  SELECT SUM(stock) INTO total FROM libros;
  RETURN total;
END //
DELIMITER ;
```

Ejemeplo 2

```
DELIMITER //
CREATE FUNCTION obtener_autor(p_id_libro INT)
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
  DECLARE nombre_autor VARCHAR(100);
  SELECT autor INTO nombre_autor FROM libros WHERE id_libro
= p_id_libro;
  RETURN nombre_autor;
END //
DELIMITER ;
```

Triggers

Ejemplo 1

```
DELIMITER //
CREATE TRIGGER trigger_insert_libro
AFTER INSERT ON libros
FOR EACH ROW
BEGIN
  INSERT INTO auditoria_libros (id_libro, accion, usuario)
  VALUES (NEW.id_libro, 'Libro insertado', CURRENT_USER());
END //
DELIMITER ;
```

Ejemeplo 2

```
DELIMITER //
CREATE TRIGGER trigger_update_stock
AFTER UPDATE ON libros
FOR EACH ROW
BEGIN
  IF OLD.stock != NEW.stock THEN
    INSERT INTO auditoria_libros (id_libro, accion, usuario)
    VALUES (NEW.id_libro, CONCAT('Stock cambiado de ',
OLD.stock, ' a ', NEW.stock), CURRENT_USER());
  END IF;
END //
DELIMITER ;
```

Ejemplo 1

```
CREATE TABLE productos (
```

```
 id_producto NUMBER PRIMARY KEY,
 nombre VARCHAR2(100),
 precio NUMBER(10, 2),
 stock NUMBER
);
```

Ejemplo 2

```
CREATE TABLE pedidos (
 id_pedido NUMBER PRIMARY KEY,
 id_producto NUMBER,
 cantidad NUMBER,
 fecha_pedido DATE DEFAULT SYSDATE
);
```

ORACLE

Tablas

Procedimiento

Ejemplo 1

```
CREATE OR REPLACE PROCEDURE agregar_producto(
 p_id_producto IN NUMBER,
 p_nombre IN VARCHAR2,
 p_precio IN NUMBER,
 p_stock IN NUMBER
)
IS
BEGIN
```

```
  INSERT INTO productos (id_producto, nombre, precio, stock)
  VALUES (p_id_producto, p_nombre, p_precio, p_stock);

  COMMIT;
END;
```

Ejemeplo 2

```
CREATE OR REPLACE PROCEDURE registrar_pedido(
  p_id_pedido IN NUMBER,
  p_id_producto IN NUMBER,
  p_cantidad IN NUMBER
)
IS
BEGIN
  INSERT INTO pedidos (id_pedido, id_producto, cantidad)
  VALUES (p_id_pedido, p_id_producto, p_cantidad);
  UPDATE productos
  SET stock = stock - p_cantidad
  WHERE id_producto = p_id_producto;
  COMMIT;
END;
```

Funcion

Ejemplo 1

```
CREATE OR REPLACE FUNCTION calcular_total_pedido(
  p_id_producto IN NUMBER,
  p_cantidad IN NUMBER
```

```
) RETURN NUMBER
IS
  v_precio NUMBER;
BEGIN
  SELECT precio INTO v_precio FROM productos WHERE
id_producto = p_id_producto;
  RETURN v_precio * p_cantidad;
END;
```

Ejemplo 2

```
CREATE OR REPLACE FUNCTION obtener_stock(
  p_id_producto IN NUMBER
) RETURN NUMBER
IS
  v_stock NUMBER;
BEGIN
  SELECT stock INTO v_stock FROM productos WHERE
id_producto = p_id_producto;
  RETURN v_stock;
END;
```

Triggers

Ejemplo 1

```
CREATE OR REPLACE TRIGGER verificar_stock_pedido
BEFORE INSERT ON pedidos
FOR EACH ROW
DECLARE
```

```
 v_stock NUMBER;
BEGIN
 SELECT stock INTO v_stock FROM productos WHERE
id_producto = :NEW.id_producto;
 IF :NEW.cantidad > v_stock THEN
  RAISE_APPLICATION_ERROR(-20001, 'Stock insuficiente para
realizar el pedido.');
 END IF;
END;
```

Ejemplo 2

```
CREATE OR REPLACE TRIGGER devolver_stock_al_eliminar
AFTER DELETE ON pedidos
FOR EACH ROW
BEGIN
 UPDATE productos
 SET stock = stock + :OLD.cantidad
 WHERE id_producto = :OLD.id_producto;
END;
```

Ejemplo 1

```
CREATE TABLE cursos (
 id_curso SERIAL PRIMARY KEY,
 nombre VARCHAR(100),
```

```
 duracion_horas INT
);
```

Ejemplo 2

```
CREATE TABLE estudiantes (
 id_estudiante SERIAL PRIMARY KEY,
 nombre VARCHAR(100),
 email VARCHAR(100)
);
CREATE TABLE inscripciones (
 id_inscripcion SERIAL PRIMARY KEY,
 id_estudiante INT REFERENCES
estudiantes(id_estudiante),
 id_curso INT REFERENCES cursos(id_curso),
 fecha_inscripcion DATE DEFAULT CURRENT_DATE
);
```

POSTGRE

Tablas

INDICE

Procedimiento

Ejemplo 1

```
CREATE OR REPLACE PROCEDURE inscribir_estudiante(
 p_id_estudiante INT,
 p_id_curso INT
)
LANGUAGE plpgsql
AS $$
```

```
BEGIN
  INSERT INTO inscripciones (id_estudiante, id_curso)
  VALUES (p_id_estudiante, p_id_curso);
END;
$$;
```

Ejemplo 2

```
CREATE OR REPLACE PROCEDURE actualizar_duracion_curso(
  p_id_curso INT,
  p_nueva_duracion INT
)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE cursos
  SET duracion_horas = p_nueva_duracion
  WHERE id_curso = p_id_curso;
END;
$$;
```

INDICE

Funcion

Ejemplo 1

```
CREATE OR REPLACE FUNCTION contar_inscritos(p_id_curso INT)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
```

```
 total INT;
BEGIN
 SELECT COUNT(*) INTO total
 FROM inscripciones
 WHERE id_curso = p_id_curso;
 RETURN total;
END;
$$;
```

Ejemplo 2

```
CREATE OR REPLACE FUNCTION esta_inscrito(p_id_estudiante INT,
p_id_curso INT)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
DECLARE
 existe BOOLEAN;
BEGIN
 SELECT EXISTS (
  SELECT 1
  FROM inscripciones
  WHERE id_estudiante = p_id_estudiante AND id_curso = p_id_curso
 ) INTO existe;
 RETURN existe;
END;
$$;
```

INDICE

Triggers

Ejemplo

```
CREATE OR REPLACE FUNCTION evitar_inscripcion_duplicada()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
 IF EXISTS (
  SELECT 1 FROM inscripciones
  WHERE id_estudiante = NEW.id_estudiante AND id_curso = NEW.id_curso
 ) THEN
  RAISE EXCEPTION 'El estudiante ya está inscrito en este curso.';
 END IF;
 RETURN NEW;
END;
$$;
CREATE TRIGGER trigger_evitar_inscripcion_duplicada
BEFORE INSERT ON inscripciones
FOR EACH ROW
EXECUTE FUNCTION evitar_inscripcion_duplicada();
```

Ejemplo 2

```
CREATE OR REPLACE FUNCTION log_curso_nuevo()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
```

```
 RAISE NOTICE 'Curso agregado: %', NEW.nombre;
 RETURN NEW;
END;
$$;
CREATE TRIGGER trigger_log_curso
AFTER INSERT ON cursos
FOR EACH ROW
EXECUTE FUNCTION log_curso_nuevo();
```

EJERCICIOS EN CLASE

Oracle

```
CREATE OR REPLACE FUNCTION dv (codigo VARCHAR)
RETURN PLS_INTEGER IS
i PLS_INTEGER DEFAULT 1;
suma PLS_INTEGER DEFAULT O;
digito PLS_INTEGER DEFAULT 0;
```

```
BEGIN
WHILE (i<12) LOOP
suma := suma + TO_NUMBER (SUBSTR(codigo.i,l))
3 * (TO_NUMBER (SUBSTR(codigo,i+1.1)));
i : i+2;
END LOOP;
digito := MOD((TRUNC(suma/10)+1)*10 - suma, 10)
RETURN digito;
END;
SELECT dv('123456789012') FROM dual; //--- 8
SELECT dv('750151110231') FROM dual;//---- 1
SELECT dv('750142870359') FROM dual; //----5
SELECT dv('750223725095') FROM dual; //--- 9
SELECT dv('750158000079') FROM dual; //--- 2
SELECT dv('978020137962') FROM dual; //-----4
SELECT dv('077211183150') FROM dual; //---4
SELECT DBMS_RANDOM.string('U',4) FROM DUAL;
SELECT DBMS_RANDOM, string('L',4) FROM DUAL;
SELECT TRUNC(DBMS_RANDOM.value(0,9)) FROM DUAL;
```

EJERCICIOS EN CLASE

Oracle

INDICE

```
SELECT DBMS_RANDOM.string('U',4) FROM DUAL;
SELECT DBMS_RANDOM.string('L',5) FROM DUAL;
SELECT TRUNC(DBMS_RANDOM.value(0,9)) FROM DUAL;
SELECT DBMS_RANDOM.string('U',1) FROM DUAL;
```

```
SELECT SUBSTR('(+$#()-!()([])',DBMS_RANDOM.value(1,14),1) FROM
DUAL;
SELECT DBMS_RANDOM.string('A',100) FROM DUAL;
CREATE OR REPLACE FUNCTION generatePassword RETURN VARCHAR AS
password VARCHAR(20);
first_letter VARCHAR(1);
second_digit VARCHAR(1);
third_special_symbol VARCHAR(1);
alphanumeric VARCHAR(4);
others VARCHAR(12);
BEGIN
SELECT BMS_RANDOM.string('U',1) INTO first_letter FROM DUAL;
SELECT TRUNC(DBMS_RANDOM.value(0,9)) INTO second_digit FROM DUAL;
SELECT SUBSTR('(+S#()-I(>([])',TRUNC(DBMS_RANDOM.value(1,14)),1) INTO
third_special_symbol FROM DUAL;
SELECT DBMS_RANDOM.string('A',4) INTO alphanumeric FROM DUAL;
SELECT DBMS_RANDOM.string('A',TRUNC(DBMS_RANDOM.value(1,12))) INTO
others FROM DUAL;
password
:= first_letter || second_digit |l third_special_symbol ||
alphanumeric ||
others;
RETURN password;
END;
/
INDICE
```

```
CREATE TABLE empleados
nombre VARCHAR2 (50),
id_empleado NUMBER PRIMARY KEY,
salario NUMBER
);
CREATE OR REPLACE PROCEDURE agregar_empleado(
p_id IN NUMBER,
p_nombre IN VARCHAR2
p_salario IN NUMBER
) IS
BEGIN
INSERT INTO empleados (id_empleado, nombre
salario)
VALUES (p_id, p_nombre, p_salario);
COMMIT;
END agregar_empleado;
CALL agregar_empleado(2, 'Maria Martinez', 50000);
SELECT * FROM empleados;
CREATE OR REPLACE PROCEDURE listar_empleados IS
CURSOR _empleados IS
SELECT id_empleado, nombre FROM empleados;
v_id empleados.id_empleado%TYPE;
v_nombre empleados.nombre%TYPE;
BEGIN
OPEN c_empleados;
LOOP
```

```
FETCH _empleados INTO v_id, v_nombre;
EXIT WHEN c_empleados%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('ID: ' || v_id || ' Nombre:' || v_nombre);
END LOOP;
CLOSE c_empleados;
END listar_empleados;
```

INDICE

```
CREATE OR REPLACE PROCEDURE procesar_pago(
p_id_empleado IN NUMBER,
p_monto_pago IN NUMBER
) IS
BEGIN
SAVEPOINT inicio_proceso;
UPDATE empleados
SET salario = salario + p_monto_pago
WHERE id_empleado = p_id_empleado;
IF SQL%ROWCOUNT = 0 THEN
ROLLBACK TO inicio_proceso;
DBMS_OUTPUT.PUT_LINE('Error: Empleado no encontrado.');
ELSE
COMMIT;
DBMS_OUTPUT.PUT_LINE('Pago procesado correctamente.');
END IF;
EXCEPTION
WHEN OTHERS THEN
ROLLBACK TO inicio_proceso;
```

```
DBMS_OUTPUT.PUT_LINE('Error durante el proceso: ' || SQLERRM);
END procesar_pago;
/
INDICE
CREATE TABLE auditoria_empleados(
id_auditoria NUMBER GENERATED BY DEFAULT AS IDENTITY,
id_empleado NUMBER(20),
accion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
usuario VARCHAR(30)
);
CREATE OR REPLACE PROCEDURE auditoria_empleado(
p_id_empleado IN NUMBER,
p_accion IN VARCHAR2
) IS
BEGIN
INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
VALUES (p_id_empleado, p_accion, USER);
COMMIT;
END auditoria_empleado ;
/
GRANT CREATE TRIGGER TO CO;
-------------------------TRIGGERS--------------------------CREATE
OR
REPLACE
TRIGGER
validar_salario
```

```
BEFORE
INSERT
ON
empleados
FOR
EACH
ROW
BEGIN
IF :NEW,salario < 0 THEN
RAISE_APPLICATION_ERROR(-20001,
'El salario no puede ser
negativo.');
END IF;
END validar_salario;
INSERT INTO empleados (ID_EMPLEADO, nombre, SALARIO)
VALUES (1, 'Ambrosio Cardoso', -10000);
INDICE
CREATE OR REPLACE TRIGGER auditoria_insercion_empleado
AFTER INSERT ON empleados
FOR EACH ROW
BEGIN
INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
VALUES (:NEW.id_Empleado, 'INSERCION', USER);
END auditoria_insercion_empleado;
INSERT INTO empleados (ID_EMPLEADO, nombre, SALARIO)
VALUES (1, 'Ambrosio Cardoso', 10000);
```

```
CREATE OR REPLACE TRIGGER auditoria_cambio_empleado
BEFORE UPDATE OF salario ON empleados
FOR EACH ROW
BEGIN
INSERT INTO auditoria_empleados (id_empleado, accion, usuario)
VALUES (:OLD.id_Empleado, 'CAMBIO SALARIO', USER);
END auditoria_cambio_empleado;
UPDATE EMPLEADOS e SET salario=salario * 1.10
WHERE id_empleado = 1;
SELECT * FROM EMPLEADOS e;
SELECT * FROM auditoria_empleados;
CREATE OR REPLACE TRIGGER prevenir_borrado_jefe
BEFORE DELETE ON empleados
FOR EACH ROW
BEGIN
IF :OLD.nombre = 'Director General' THEN
RAISE_APPLICATION_ERROR(-20002, 'No se puede borrar al Director
General');
END IF
END prevenir_borrado_jefe;
CREATE OR REPLACE TRIGGER trigger_bloqueo_tabla
```

EJERCICIOS EN CLASE

MariaDB

```
DELIMITER //
CREATE FUNCTION generatePassword() RETURNS VARCHAR(20)
NOT DETERMINISTIC
BEGIN
DECLARE generated_password VARCHAR(20) DEFAULT '';
DECLARE first_letter CHAR(1) DEFAULT '';
DECLARE second_digit CHAR(1) DEFAULT '';
DECLARE third_special_symbol CHAR(1) DEFAULT '';
DECLARE alphanumeric VARCHAR(4) DEFAULT '';
DECLARE others VARCHAR(12) DEFAULT '';
DECLARE symbols VARCHAR(20) DEFAULT '(+$#()-!{}{[])';
DECLARE i INT DEFAULT 0;
DECLARE len INT DEFAULT 0;
-- Letra mayúscula (A-Z)
SET first_letter = CHAR(FLOOR(RAND() * 26) + 65);
-- Dígito (0-9)
SET second_digit = CHAR(FLOOR(RAND() * 10) + 48);
-- Símbolo especial
SET i = FLOOR(RAND() * CHAR_LENGTH(symbols)) + 1;
SET third_special_symbol = SUBSTRING(symbols, i, 1);
-- 4 letras minúsculas aleatorias
SET alphanumeric = '';
```

```
SET i = 0;
WHILE i < 4 DO
SET alphanumeric = CONCAT(alphanumeric, CHAR(FLOOR(RAND() * 26) +
97));
SET i = i + 1;
END WHILE;
-- De 1 a 12 letras minúsculas aleatorias
SET others = '';
SET len = FLOOR(RAND() * 12) + 1;
```

EJERCICIOS EN CLASE

MariaDB

INDICE

```
SET i = 0;
WHILE i < len DO
SET others = CONCAT(others, CHAR(FLOOR(RAND() * 26) + 97));
SET i = i + 1;
END WHILE;
-- Concatenar componentes paso a paso
SET generated_password = CONCAT(first_letter, second_digit);
SET
generated_password
=
CONCAT(generated_password,
third_special_symbol);
SET generated_password = CONCAT(generated_password, alphanumeric);
SET generated_password = CONCAT(generated_password, others);
```

```
RETURN generated_password;
END;
//
DELIMITER ;
Crear BD rh y tabla empleados
CREATE DATABASE
rh CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
USE rh;
CREATE TABLE empleados (
id_empleado INT PRIMARY KEY,
nombre VARCHAR(100),
salario DOUBLE
);
Paso 3: Crear un procedimiento para agregar empleados
DELIMITER //
CREATE PROCEDURE agregar_empleado(
IN p_id INT,
IN p_nombre VARCHAR(255),
IN p_salario DECIMAL(10,2)
)
BEGIN
INSERT INTO empleados (id_empleado, nombre, salario)
VALUES (p_id, p_nombre, p_salario);
COMMIT;
END;
```

```
//
DELIMITER ;
INDICE
-- Ejemplo de uso:
CALL agregar_empleado(1,'Ambrosio Cardoso Jimenez',10000);
Paso 4: Modificar el procedimiento agregando validación
DELIMITER //
CREATE OR REPLACE PROCEDURE agregar_empleado(
IN p_id INT,
IN p_nombre VARCHAR(255),
IN p_salario DECIMAL(10,2)
)
BEGIN
IF p_salario >= 5000 AND p_salario <= 30000 THEN
INSERT INTO empleados (id_empleado, nombre, salario)
VALUES (p_id, p_nombre, p_salario);
COMMIT;
ELSE
-- Como MariaDB no tiene DBMS_OUTPUT, usamos SIGNAL para lanzar un
mensaje de error
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'No se pudo agregar el empleado: salario fuera de
rango';
END IF;
END;
//
```

```
DELIMITER ;
-- Ejemplo de uso:
CALL agregar_empleado(2,'Juan Climaco Rubio Cosme',15000);
CALL agregar_empleado(3,'Rosa Lujan Martinez',12000);
CALL agregar_empleado(4,'Alienigena',-5000);
```

Triggers MariaDB

Paso 5: Crear una tabla

```
CREATE TABLE auditoria_cambios (
id_auditoria INT AUTO_INCREMENT PRIMARY KEY,
tabla_afectada VARCHAR(50),
operacion VARCHAR(10),
usuario_bd VARCHAR(30),
fecha_operacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
id_registro VARCHAR(100),
valor_anterior TEXT,
valor_nuevo TEXT
);
```

INDICE

Paso 6: Crear trigger de INSERCIÓN (AFTER)

```
DELIMITER //
CREATE TRIGGER trg_auditoria_empleados_insert
AFTER INSERT ON empleados
FOR EACH ROW
BEGIN
INSERT INTO auditoria_cambios (
tabla_afectada,
```

```
operacion,
usuario_bd,
id_registro,
valor_anterior,
valor_nuevo
)
VALUES (
'EMPLEADOS',
'INSERT',
CURRENT_USER(),
NEW.id_empleado,
NULL,
CONCAT('Nombre: ', NEW.nombre, ', Salario: ', NEW.salario)
);
END;
//
```

Paso 7: Crear trigger de UPDATE (AFTER)

```
CREATE TRIGGER trg_auditoria_empleados_update
AFTER UPDATE ON empleados
FOR EACH ROW
BEGIN
INSERT INTO auditoria_cambios (
tabla_afectada,
operacion,
usuario_bd,
id_registro,
```

```
valor_anterior,
valor_nuevo
)
VALUES (
'EMPLEADOS',
'UPDATE',
CURRENT_USER(),
NEW.id_empleado,
CONCAT('Nombre: ', OLD.nombre, ', Salario: ', OLD.salario),
CONCAT('Nombre: ', NEW.nombre, ', Salario: ', NEW.salario)
);
END;
//
INDICE
Paso 8: Crear trigger de DELETE (AFTER)
CREATE TRIGGER trg_auditoria_empleados_delete
AFTER DELETE ON empleados
FOR EACH ROW
BEGIN
INSERT INTO auditoria_cambios (
tabla_afectada,
operacion,
usuario_bd,
id_registro,
valor_anterior,
valor_nuevo
```

```
)
VALUES (
'EMPLEADOS',
'DELETE',
CURRENT_USER(),
OLD.id_empleado,
CONCAT('Nombre: ', OLD.nombre, ', Salario: ', OLD.salario),
NULL
);
END;
//
DELIMITER ;
```

Paso 9: Ejecutar las siguientes operaciones

```
INSERT INTO empleados (id_empleado, nombre, salario)
VALUES (5, 'Laura Mendoza', 60000);
-- Actualizar salario
UPDATE empleados
SET salario = salario + 5000
WHERE id_empleado = 5;
-- Eliminar empleado
DELETE FROM empleados
WHERE id_empleado = 5;
```

Paso 10: Consulta la tabla auditoría:

```
SELECT * FROM auditoria_cambios ORDER BY fecha_operacion DESC;
```