

# ENTREGA 3

## Integrantes:

- Rosales Montero Angel
- Orellana Rodriguez, Jose Miguel
- Huaman Huaman, Israel Angel

## 1. INTRODUCCIÓN

En esta etapa final del proyecto, hemos transformado nuestra aplicación de una herramienta de gestión básica a un Sistema de Información Robusto. Si bien en la entrega anterior nos enfocamos en la integridad de datos (Constraints), en esta fase el objetivo fue migrar la lógica de negocio al servidor de base de datos.

Implementamos objetos avanzados de base de datos (Vistas, Procedimientos Almacenados y Triggers) para garantizar que las reglas críticas del negocio se ejecuten de manera centralizada, segura y eficiente, reduciendo la carga de procesamiento en la aplicación cliente (Python).

## 2. OBJETIVOS

- Abstracción de Datos: Simplificar consultas complejas mediante el uso de Vistas (Views).
- Automatización y Rendimiento: Ejecutar procesos masivos en el servidor mediante Procedimientos Almacenados (Stored Procedures).
- Seguridad y Auditoría: Implementar Triggers (Disparadores) para monitorear y registrar cambios críticos (eliminación de personal) de manera automática.
- Integración Avanzada: Actualizar la aplicación Python para invocar estos objetos de base de datos en lugar de enviar sentencias SQL planas.

## 3. INGENIERÍA DE BASE DE DATOS (SQL)

A continuación, se documentan los tres pilares de inteligencia implementados en Oracle 11g/21c XE.

### 3.1. Vistas (Views) - Reportes Inteligentes

Para evitar repetir la lógica de los JOIN y cálculos matemáticos en cada consulta de la aplicación, encapsulamos esta lógica en una vista.

Código SQL Implementado: Se creó la vista V\_REPORTE\_PLANILLA que une empleados con departamentos y calcula automáticamente el salario anual proyectado.

```
SQL> CREATE OR REPLACE VIEW V_REPORTE_PLANILLA AS
2  SELECT
3      e.CODIGO,
4      e.NOMBRE,
5      e.PUESTO,
6      d.DNOMBRE AS DEPARTAMENTO,
7      e.SALARIO,
8      (e.SALARIO * 12) AS SALARIO_ANUAL
9  FROM EMPLEADOS e
10 JOIN DEPARTAMENTOS d ON e.COD_DEPT = d.COD_DEPT;

Vista creada.
```

### 3.2. Procedimientos Almacenados - Lógica de Negocio

Las actualizaciones masivas (como aumentos de sueldo) son riesgosas y lentas si se hacen registro por registro desde Python. Delegamos esta tarea al motor de base de datos para asegurar atomicidad y velocidad.

Código SQL Implementado: El procedimiento SP\_AUMENTAR\_POR\_PUESTO recibe el cargo y el porcentaje de aumento, y actualiza a todos los empleados correspondientes en una sola transacción.

```
SQL> CREATE OR REPLACE PROCEDURE SP_AUMENTAR_POR_PUESTO (
2      p_puesto IN VARCHAR2,
3      p_porcentaje IN NUMBER
4  ) IS
5  BEGIN
6      UPDATE EMPLEADOS
7      SET SALARIO = SALARIO + (SALARIO * p_porcentaje / 100)
8      WHERE PUESTO = p_puesto;
9      COMMIT;
10 END;
11 /

Procedimiento creado.
```

### 3.3. Triggers - Sistema de Auditoría (Seguridad)

Se requería un mecanismo infalible para rastrear la pérdida de información. Implementamos un sistema de auditoría "silenciosa" que guarda copia de cualquier empleado eliminado, independientemente de si el borrado se hizo desde la App o directamente en la base de datos.

Código SQL Implementado:

Tabla de respaldo AUDITORIA\_DESPIDOS.

Trigger TRG\_AUDITAR Borrado que se activa AFTER DELETE.

```

SQL> CREATE OR REPLACE TRIGGER TRG_AUDITAR Borrado
2  AFTER DELETE ON EMPLEADOS
3  FOR EACH ROW
4  BEGIN
5      INSERT INTO AUDITORIA_DESPIDOS
6          (NOMBRE_EMPLEADO, PUESTO_ANTIGUO, FECHA_DESPIDO, USUARIO_QUE_BORRO)
7          VALUES
8          (:OLD.NOMBRE, :OLD.PUESTO, SYSDATE, USER);
9  END;
10 /

Disparador creado.

```

## 4. DESARROLLO DE LA APLICACIÓN FINAL (PYTHON)

La aplicación entrega\_final.py fue reescrita para dejar de comportarse como un simple editor de tablas y convertirse en una interfaz de control gerencial.

### 4.1. Innovaciones en el Código

- Uso de callproc: Para invocar procedimientos almacenados de forma segura.
- Lectura de Vistas: Tratamiento de la vista como si fuera una tabla física para reportes rápidos.
- Verificación de Auditoría: Módulo para visualizar la efectividad del Trigger.

### 4.2. Fragmento del Código Fuente (Nuevas Funciones)

```

def ver_reporte_vista():
1  def ver_reporte_vista():
2
3      print("\n--- REPORTE GERENCIAL (VISTA) ---")
4      conn = conectar()
5      cursor = conn.cursor()
6      cursor.execute("SELECT * FROM V_REPORTES_PLANILLA ORDER BY SALARIO DESC")
7
8  def ejecutar_aumento(puesto, porcentaje):
9      conn = conectar()
10     cursor = conn.cursor()
11     cursor.callproc('SP_AUMENTAR_POR_PUESTO', [puesto, porcentaje])
12     print("[ÉXITO] Oracle procesó el aumento masivo.")

```

## 5. CONCLUSIONES Y RESULTADOS

Al finalizar este proyecto, el grupo ha logrado simular un entorno empresarial real con las siguientes conclusiones clave:

- La implementación del Trigger demostró que es posible proteger la información crítica (auditoría) sin depender de que el programador de la aplicación recuerde escribir el código de respaldo. La base de datos se protege a sí misma.

- El uso de Procedimientos Almacenados redujo el tráfico de red. En lugar de enviar múltiples sentencias UPDATE desde Python, enviamos una sola instrucción y Oracle procesa todo internamente.
- Gracias a la Vista, si la fórmula del salario anual cambia en el futuro, solo modificamos la vista en Oracle y no necesitamos tocar ni una línea de código en Python.

## Capturas de pantalla:

```
import oracledb

DSN = "localhost:1521/xe"
USER = "angel"
PASSWORD = "princhipesa"

def conectar():
    return oracledb.connect(user=USER, password=PASSWORD, dsn=DSN)

def ver_reporte_vista():
    print("\n--- REPORTE GERENCIAL (Usando VISTA de Oracle) ---")
    conn = conectar()
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT * FROM V_REPORTE_PLANILLA ORDER BY SALARIO DESC")

        print(f"{'NOMBRE':<12} {'PUESTO':<15} {'DEPARTAMENTO':<15} {'SUELDO':<10} {'ANUAL (x12)'}")
        print("-" * 75)
        for fila in cursor:
            print(f"{fila[1]:<12} {fila[2]:<15} {fila[3]:<15} {fila[4]:<10} {fila[5]}")
            print("-" * 75)
    except oracledb.Error as e:
        print(f"[ERROR] {e}")
    finally:
        cursor.close(); conn.close()

def ejecutar_aumento(puesto, porcentaje):
    print(f"\n--- EJECUTANDO PROCEDURE: Aumento del {porcentaje}% a {puesto} ---")
    conn = conectar()
    cursor = conn.cursor()
    try:
        cursor.callproc('SP_AUMENTAR_POR_PUESTO', [puesto, porcentaje])
        print(" [ÉXITO] Oracle procesó el aumento correctamente.")
    except oracledb.Error as e:
        print(f" [ERROR] Falló el procedimiento: {e}")
    finally:
        cursor.close(); conn.close()
```

```

def crear_empleado_prueba():
    conn = conectar()
    cursor = conn.cursor()
    try:
        print("Creando empleado de prueba 'TRAMPA'")
        sql = "INSERT INTO EMPLEADOS (CODIGO, NOMBRE, PUESTO, FECHA_ING, SALARIO, COD_DEPT) VALUES (9999, 'TRAMPA', 'TEMPORAL', SYSDATE, 1000, 20)"
        cursor.execute(sql)
        conn.commit()
        print("Empleado creado.")
    except oracledb.Error as e:
        print(f"El empleado ya existía o hubo error: {e}")
    finally:
        cursor.close(); conn.close()

def eliminar_empleado_prueba():
    conn = conectar()
    cursor = conn.cursor()
    try:
        print("\nEliminando empleado 'TRAMPA'...")
        cursor.execute("DELETE FROM EMPLEADOS WHERE CODIGO = 9999")
        conn.commit()
        print("Empleado eliminado. (El Trigger debió guardar esto en auditoría)")
    except oracledb.Error as e:
        print(f"[ERROR] {e}")
    finally:
        cursor.close(); conn.close()

def ver_auditoria():
    conn = conectar()
    cursor = conn.cursor()
    try:
        print("\n--- TABLA SECRETA DE AUDITORÍA (Resultados del Trigger) ---")
        cursor.execute("SELECT NOMBRE_EMPLEADO, PUESTO_ANTIGUO, FECHA_DESPIDO, USUARIO_QUE_BORRO FROM AUDITORIA_DESPIDOS")
        print(f"{'EMPLEADO BORRADO':<20} {'PUESTO':<15} {'FECHA':<15} {'CULPABLE'}")
        print("-" * 70)
        for fila in cursor:
            print(f"{fila[0]:<20} {fila[1]:<15} {str(fila[2]):<10}:<15} {fila[3]}")
    except oracledb.Error as e:
        print(f"[ERROR] {e}")
    finally:
        cursor.close(); conn.close()

```

```

if __name__ == "__main__":
    while True:
        print("\n" + "="*40)
        print(" SISTEMA FINAL RRHH (ENTREGA 3) ")
        print("="*40)
        print("1. Ver Reporte (VISTA)")
        print("2. Aumentar Sueldos (PROCEDURE)")
        print("3. Probar Trigger (Crear, Borrar y Auditar)")
        print("4. Salir")

        opcion = input("\nElige una opción: ")

        if opcion == "1":
            ver_reporte_vista()
        elif opcion == "2":
            p = input("Ingrese Puesto (ej. GERENTE, ANALISTA): ").upper()
            pct = float(input("Porcentaje % (ej. 10): "))
            ejecutar_aumento(p, pct)
            print("(Tip: Vuelve a la opción 1 para ver los nuevos sueldos)")
        elif opcion == "3":
            crear_empleado_prueba()
            input("Presiona ENTER para borrarlo y activar el Trigger...")
            eliminar_empleado_prueba()
            ver_auditoria()
        elif opcion == "4":
            print("Cerrando sistema...")
            break
        else:
            print("Opción no válida")

```

## Ejecutando la funcionalidad del proyecto:

```
=====
SISTEMA FINAL RRHH (ENTREGA 3)
=====
1. Ver Reporte (VISTA)
2. Aumentar Sueldos (PROCEDURE)
3. Probar Trigger (Crear, Borrar y Auditar)
4. Salir

Elige una opción: 1

--- REPORTE GERENCIAL (Usando VISTA de Oracle) ---
NOMBRE      PUESTO      DEPARTAMENTO  SUELDO      ANUAL (x12)
-----
FARFAN      ANALISTA    INVESTIGACION 4500.0      54000
BENAVIDES   GERENTE     VENTAS         2850.0      34200
CASTRO      GERENTE     CONTABILIDAD   2450.0      29400
-----
```

```
=====
SISTEMA FINAL RRHH (ENTREGA 3)
=====
1. Ver Reporte (VISTA)
2. Aumentar Sueldos (PROCEDURE)
3. Probar Trigger (Crear, Borrar y Auditar)
4. Salir

Elige una opción: 2
Ingrese Puesto (ej. GERENTE, ANALISTA): GERENTE
Porcentaje % (ej. 10): 10

--- EJECUTANDO PROCEDURE: Aumento del 10.0% a GERENTE ---
[ÉXITO] Oracle procesó el aumento correctamente.
(Tip: Vuelve a la opción 1 para ver los nuevos sueldos)
```

```
Elige una opción: 3
Creando empleado de prueba 'TRAMPA'
Empleado creado.
Presiona ENTER para borrarlo y activar el Trigger...

Eliminando empleado 'TRAMPA'...
Empleado eliminado. (El Trigger debió guardar esto en auditoría)

--- TABLA SECRETA DE AUDITORÍA (Resultados del Trigger) ---
EMPLEADO BORRADO  PUESTO      FECHA      CULPABLE
-----
TRAMPA            TEMPORAL    2025-11-29  ANGEL
TRAMPA            TEMPORAL    2025-11-29  ANGEL
```

Link de GitHub:

<https://github.com/AngelRosales-78/Entrega-1>