ENTREGA 2

Integrantes:

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

1. INTRODUCCIÓN

En esta segunda etapa del proyecto, evolucionamos de una estructura de base de datos estática a un sistema dinámico e interactivo. El objetivo principal es implementar la lógica de negocio necesaria para la gestión de empleados mediante una aplicación externa desarrollada en Python.

Asimismo, se refuerza la integridad y el rendimiento de la base de datos Oracle 11g implementando restricciones avanzadas (`CONSTRAINTS`) y optimización de búsquedas mediante índices (`INDEX`), cumpliendo con los estándares de administración de bases de datos relacionales.

2. OBJETIVOS

- Asegurar la integridad de los datos mediante restricciones `CHECK` y `UNIQUE`.
- Optimizar el rendimiento de las consultas mediante la creación de Índices.
- Generar reportes gerenciales utilizando consultas avanzadas (`JOINS`, subconsultas y funciones de agregación).
- Desarrollar una aplicación en Python que realice operaciones CRUD (Crear, Leer, Actualizar, Eliminar) conectada a Oracle.
- Implementar el manejo de transacciones (`COMMIT` y `ROLLBACK`) para garantizar la consistencia de los datos.

3. DESARROLLO DE BASE DE DATOS (SQL)

A continuación, se detallan las modificaciones y consultas ejecutadas directamente en el motor de base de datos.

3.1. Implementación de Restricciones (Constraints)

Para garantizar que los datos ingresados sean lógicos y consistentes, se aplicaron las siguientes reglas de negocio a la tabla `EMPLEADOS`:

1. Validación de Salario (CHECK): Se impide el ingreso de salarios negativos o iguales a cero.
2. Unicidad de Nombres (UNIQUE): Se evita la duplicidad de empleados con el mismo nombre exacto.

```
ALTER TABLE EMPLEADOS
ADD CONSTRAINT chk_salario_positivo CHECK (SALARIO > 0);


ALTER TABLE EMPLEADOS
ADD CONSTRAINT unq_nombre_emp UNIQUE (NOMBRE);
```

3.2. Optimización (Índices)

Dado que las búsquedas por el cargo o puesto del empleado serán frecuentes en la aplicación, se creó un índice para acelerar estas consultas.

Código SQL Ejecutado:

```
CREATE INDEX idx_puesto_emp ON EMPLEADOS(PUESTO);
```

3.3. Consultas Avanzadas y Reportes

Se diseñaron consultas complejas para extraer información valiosa para la toma de decisiones.

A. Reporte de Gasto Salarial por Departamento (JOIN + Agregación) Esta consulta une las tablas `EMPLEADOS` y `DEPARTAMENTOS` para calcular cuántos empleados hay en cada área y cuánto suman sus sueldos.

SQL

```
SELECT d.DNOMBRE AS DEPARTAMENTO,
    COUNT(e.CODIGO) as CANTIDAD_PERSONAL,
    SUM(e.SALARIO) as GASTO_TOTAL_PLANILLA
FROM EMPLEADOS e
JOIN DEPARTAMENTOS d ON e.COD_DEPT = d.COD_DEPT
GROUP BY d.DNOMBRE;
```

B. Empleados con Salario Superior al Promedio (Subconsulta) Se utiliza una subconsulta para filtrar a los empleados que ganan más que la media de la empresa.

SQL

SELECT NOMBRE, PUESTO, SALARIO

FROM EMPLEADOS

WHERE SALARIO > (SELECT AVG(SALARIO) FROM EMPLEADOS);

4. DESARROLLO DE LA APLICACIÓN (PYTHON + ORACLEDB)

Se desarrolló un script en Python capaz de interactuar con la base de datos. El sistema implementa el manejo de errores (`try-except`) y control de transacciones para asegurar que los cambios se guarden permanentemente (`commit`) o se deshagan en caso de error (`rollback`).

4.1. Código Fuente

El siguiente código muestra la implementación completa de las funciones CRUD:

Python:

```
import oracledb

oracledb.init_oracle_client(lib_dir=r"C:\oracle")

DSN = "localhost:1522/xe"

USER = "angel"

PASSWORD = "princhipesa"


def conectar():

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


def crear_empleado(codigo, nombre, puesto, salario, dept):

    conn = conectar()

    cursor = conn.cursor()
```

```
    try:
        sql = """
            INSERT INTO EMPLEADOS (CODIGO, NOMBRE, PUESTO, FECHA_ING,
SALARIO, COD_DEPT)
            VALUES (:1, :2, :3, SYSDATE, :4, :5)
        """
        cursor.execute(sql, [codigo, nombre, puesto, salario, dept])
        conn.commit()
        print(f"[ÉXITO] Empleado '{nombre}' insertado correctamente.")
    except oracledb.Error as e:
        conn.rollback()
        print(f"[ERROR] No se pudo insertar: {e}")
    finally:
        cursor.close()
        conn.close()


def leer_empleados():
    conn = conectar()
    cursor = conn.cursor()
    try:
        sql = """
            SELECT e.CODIGO, e.NOMBRE, e.PUESTO, e.SALARIO, d.DNOMBRE
            FROM EMPLEADOS e
            JOIN DEPARTAMENTOS d ON e.COD_DEPT = d.COD_DEPT
            ORDER BY e.CODIGO
        """
        cursor.execute(sql)
```

```
        print("\n" + "="*70)
        print(f"{'COD':<5} {'NOMBRE':<12} {'PUESTO':<15} {'SALARIO':<10} {'DEPARTAMENTO'}")
        print("-" * 70)
        for fila in cursor:
            print(f"{fila[0]:<5} {fila[1]:<12} {fila[2]:<15} {fila[3]:<10} {fila[4]}")
        print("="*70 + "\n")
    except oracledb.Error as e:
        print(f"[ERROR] Consulta fallida: {e}")
    finally:
        cursor.close()
        conn.close()

def actualizar_salario(codigo, nuevo_salario):
    conn = conectar()
    cursor = conn.cursor()
    try:
        sql = "UPDATE EMPLEADOS SET SALARIO = :1 WHERE CODIGO = :2"
        cursor.execute(sql, [nuevo_salario, codigo])
        conn.commit() # Confirmar transacción
        print(f"[ÉXITO] Salario del empleado {codigo} actualizado a {nuevo_salario}.")
    except oracledb.Error as e:
        conn.rollback()
        print(f"[ERROR] Actualización fallida: {e}")
    finally:
        cursor.close()
        conn.close()
```

```
def eliminar_empleado(codigo):
    conn = conectar()
    cursor = conn.cursor()
    try:
        sql = "DELETE FROM EMPLEADOS WHERE CODIGO = :1"
        cursor.execute(sql, [codigo])
        conn.commit() # Confirmar transacción
        print(f"[ÉXITO] Empleado {codigo} eliminado del sistema.")
    except oracledb.Error as e:
        conn.rollback()
        print(f"[ERROR] Eliminación fallida: {e}")
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    print("--- INICIANDO DEMOSTRACIÓN DEL SISTEMA CRUD ---")

    print("1. Creando registro de prueba...")
    crear_empleado(9999, 'TEST_USER', 'DESARROLLADOR', 2500, 20)

    print("2. Listando empleados...")
    leer_empleados()

    print("3. Actualizando salario...")
```

```
actualizar_salario(9999, 4500)


input("Presiona ENTER para eliminar el registro y finalizar...")

eliminar_empleado(9999)


print("FIN DE LA DEMOSTRACIÓN")
```

5. EVIDENCIA DE EJECUCIÓN

En esta entrega se logró integrar exitosamente el lenguaje de programación Python con la base de datos Oracle 11g. Se destaca la importancia del uso de transacciones (`commit`) para asegurar que los datos persistan correctamente en la base de datos. Asimismo, las mejoras a nivel de SQL (Restricciones e Índices) garantizan una base de datos más segura y eficiente para el futuro crecimiento del sistema.

Link de GitHub:

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