Gestión Integral de Ventas y Empleados (GIVE)

Autores:

- Amaya Brenda
- Cortes Camila
- Zucaria Nahir
- Lelli Florencia

Descripción

Este proyecto consiste en el desarrollo de una aplicación en Python que gestiona las ventas, los empleados, el inventario y los datos de clientes de una empresa, con funcionalidades de evaluación de satisfacción del cliente. La base de datos MySQL almacena los datos y permite realizar consultas sobre las distintas áreas de gestión. Este proyecto se compone de las siguientes fases:

1. Planeación: Definición de requerimientos y diseño del sistema.
2. Desarrollo: Creación de la estructura de base de datos en MySQL y programación del sistema en Python.
3. Pruebas: Evaluación de funcionalidad y verificación de consistencia de datos.
4. Presentación y Evaluación: Presentación de resultados y evaluación de efectividad del sistema.

Justificación

Este proyecto es significativo para optimizar la gestión de ventas y empleados en pequeñas y medianas empresas. Al automatizar estas áreas, el sistema contribuye a mejorar la precisión de la información, facilita el monitoreo de desempeño y satisface las necesidades de evaluación de satisfacción del cliente. La solución es especialmente valiosa en contextos donde la eficiencia administrativa y la satisfacción del cliente tienen un impacto directo en la competitividad y el éxito del negocio.

Objetivos

- Objetivo General: Implementar un sistema integrado que optimice la gestión de ventas, empleados, inventario y satisfacción del cliente en una base de datos relacional.
- Objetivos Específicos:
    1. Diseñar y estructurar una base de datos en MySQL que almacene información de ventas, empleados, inventario y clientes.

2. Crear una aplicación en Python que facilite la interacción con los datos de manera intuitiva.
3. Implementar funcionalidades de evaluación de satisfacción del cliente y generación de reportes de desempeño.

Metodología

La ejecución del proyecto se organiza en las siguientes actividades:

- Planeación: Diseño de la estructura de la base de datos y planificación de la arquitectura del programa.
- Desarrollo del Sistema:
  - Programación de funciones de gestión de ventas, empleados y clientes.
  - Creación de interfaces de usuario con menú para la gestión de datos.
- Evaluación: Pruebas de cada módulo para asegurar el cumplimiento de los requerimientos.
- Recursos:
  - Equipo Humano: Programadores, diseñador de bases de datos.
  - Equipo Técnico: Computadoras, software de desarrollo (VS Code, MySQL Workbench).
  - Recursos Económicos: Sin costos adicionales, dado el uso de software libre.

Cronograma

Etapa del Proyecto	Actividad	Fecha Inicio	Fecha Fin
Planeación	Definición de requerimientos y diseño del sistema	03/09/2024	03/09/2024
Evidencia 1	Creación de la estructura de clases	04/09/2024	07/09/2024
	Diseño y creación de la BD	07/09/2024	08/09/2024

Evidencia 2	Desarrollo del CRUD de usuarios	04/10/2024	05/10/2024
	Almacenamiento de datos de archivos no binarios	05/10/2024	05/10/2024
	Pruebas de funcionalidad	06/10/2024	06/10/2024
Evidencia 3	Integración funciones avanzadas	21/10/2024	23/10/2024
	Implementación de gestión de registros pluviales	22/10/2024	23/10/2024
	Creación de gráficos	23/10/2024	23/10/2024
Presentación final	Ordenamiento	28/10/2024	31/10/2024
	Documentación	29/10/2024	01/11/2024

Presentación del Proyecto

Capturas de Pantalla del Script en Python

➔ Fragmento del menú principal

```python
def menu_principal():
    while True:
        print(Fore.CYAN + "\n--- Menú Principal ---")
        print(Fore.YELLOW + "1. Gestión de usuarios.")
        print(Fore.YELLOW + "2. Ingresar al sistema.")
        print(Fore.YELLOW + "3. Mostrar accesos.")
        print(Fore.YELLOW + "4. Salir.")

        opcion = input(Fore.GREEN + "Ingrese su opción: ")

        if opcion == "1":
            menu_gestion_usuarios()
        elif opcion == "2":
            ejecutar.ingresar_usuario()
        elif opcion == "3":
            ejecutar.mostrar_accesos()
        elif opcion == "4":
            print(Fore.RED + "Saliendo de la aplicación...")
            sys.exit()
        else:
            print(Fore.RED + "Opción incorrecta. Ingrese otra.")
```

➔ Fragmento menú gestión de usuarios

```python
def menu_gestion_usuarios():
    while True:
        print(Fore.CYAN + "\n--- Menú de Gestión de Usuarios ---")
        print(Fore.YELLOW + "1. Agregar un nuevo usuario.")
        print(Fore.YELLOW + "2. Modificar un usuario.")
        print(Fore.YELLOW + "3. Eliminar un usuario.")
        print(Fore.YELLOW + "4. Buscar un usuario.")
        print(Fore.YELLOW + "5. Mostrar todos los usuarios.")
        print(Fore.YELLOW + "6. Ordenar usuarios por burbuja y guardar.")
        print(Fore.YELLOW + "7. Volver al menú principal.")

        opcion = input(Fore.GREEN + "Ingrese su opción: ")

        if opcion == "1":
            ejecutar.agregar_usuario()
        elif opcion == "2":
            ejecutar.modificar_usuario()
        elif opcion == "3":
            ejecutar.eliminar_usuario()
        elif opcion == "4":
            ejecutar.buscar_usuario()
        elif opcion == "5":
            ejecutar.mostrar_usuarios()
        elif opcion == "6":
            ejecutar.ordenar_usuarios_burbuja()
        elif opcion == "7":
            break
        else:
            print(Fore.RED + "Opción incorrecta. Ingrese otra.")
```

➔ Función ordenación por burbuja

```python
def ordenar_usuarios_burbuja(self):
    usuarios_list = list(self.usuarios.values())  # Convertimos el diccionario de usuarios a una lista de objetos Usuario
    n = len(usuarios_list)
    for i in range(n):
        for j in range(0, n - i - 1):
            # Comparamos los usernames de los usuarios para ordenarlos
            if usuarios_list[j].username > usuarios_list[j + 1].username:
                usuarios_list[j], usuarios_list[j + 1] = usuarios_list[j + 1], usuarios_list[j]

    # Volver a convertir la lista ordenada en un diccionario
    self.usuarios = {user.username: user for user in usuarios_list}
    self.guardar_usuarios()  # Guardar los usuarios ordenados en el archivo binario
    self.usuarios_ordenados = True  # Actualizamos el estado a ordenado
    print("Usuarios ordenados por burbuja y guardados en usuarios.ispc.")
```

➔ Función búsqueda de usuarios (binaria y secuencial)

```python
class Usuarios:  #para crud de los usuarios
    def buscar_usuario(self):
        username = input("Ingrese el username del usuario a buscar: ")
        if self.usuarios_ordenados:
            # Implementación de búsqueda binaria
            print("Búsqueda realizada por técnica de búsqueda binaria.")
            resultado = self.busqueda_binaria(username)
        else:
            # Implementación de búsqueda secuencial
            print("Búsqueda realizada por técnica de búsqueda secuencial.")
            resultado = self.busqueda_secuencial(username)

        if resultado:
            print(f"ID: {resultado.id}, Username: {resultado.username}, Email: {resultado.email}")
        else:
            print("Usuario no encontrado.")

    def busqueda_secuencial(self, username):
        for user in self.usuarios.values():
            if user.username == username:
                return user
        return None

    def busqueda_binaria(self, username):
        usuarios_list = sorted(self.usuarios.values(), key=lambda user: user.username)
        low, high = 0, len(usuarios_list) - 1

        while low <= high:
            mid = (low + high) // 2
            if usuarios_list[mid].username == username:
                return usuarios_list[mid]
            elif usuarios_list[mid].username < username:
                low = mid + 1
            else:
                high = mid - 1

        return None
```

➔ Función menu de la base de datos

```python
def menu_consultas(conn):
    while True:
        print(Fore.CYAN + "\n--- Menú Base de datos: CRUD y consultas ---")
        print(Fore.YELLOW + "1. CRUD de empleados.")
        print(Fore.YELLOW + "2. Lista completa de productos.")
        print(Fore.YELLOW + "3. Clientes y sus direcciones.")
        print(Fore.YELLOW + "4. Empleados activos y sus ventas.")
        print(Fore.YELLOW + "5. Resumen de ventas por área.")
        print(Fore.YELLOW + "6. Satisfacción del cliente por empleado.")
        print(Fore.YELLOW + "7. Ventas totales por empleado.")
        print(Fore.YELLOW + "8. Productos con el stock")
        print(Fore.YELLOW + "9. Volver.")

        opcion = input(Fore.GREEN + "Seleccione una opción: ")

        if opcion == "1":
            crud_empleados(conn)
        elif opcion == "2":
            consultas.consulta_dos(conn)
        elif opcion == "3":
            consultas.consulta_tres(conn)
        elif opcion == "4":
            consultas.consulta_cuatro(conn)
        elif opcion == "5":
            consultas.consulta_cinco(conn)
        elif opcion == "6":
            consultas.consulta_seis(conn)
        elif opcion == "7":
            consultas.consulta_siete(conn)
        elif opcion == "8":
            consultas.consulta_ocho(conn)
        elif opcion == "9":
            break
        else:
            print(Fore.RED + "Opción no válida. Intente de nuevo.")
```

➔ Registros Pluviales

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Configurar la ruta de archivos #
path = os.path.dirname(os.path.abspath("__file__")) + "/"

# Crear directorio para los datos analizados si no existe #
datos_dir = path + "datosAnalizados/"
if not os.path.exists(datos_dir):
    os.makedirs(datos_dir)

# Solicitar el año y validacion de rango de años #
try:
    año = int(input("Ingresar el año a analizar: "))
    if año < 1990 or año > 2025:
        raise ValueError("El año debe estar entre 1990 y 2025")
    archivo = f"{datos_dir}registroPluvial{año}.csv"
except ValueError as e:
    print("Por favor ingrese un año válido")
    exit()

# Verificar si existe el archivo #
if os.path.exists(f"registroPluvial{año}.csv"):
    df_precipitaciones = pd.read_csv(f"registroPluvial{año}.csv")
else:
    # Generar datos aleatorios #
    # Considerando años no bisiestos #
    np.random.seed(42)
    precipitaciones = np.round(np.random.rand(372) * 100, 2)
    precipitaciones = precipitaciones.reshape(31, 12)

    # Crear DataFrame #
    columnas = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio',
                'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre']
    df_precipitaciones = pd.DataFrame(precipitaciones, columns=columnas)

    # Guardar archivo CSV #
    df_precipitaciones.to_csv(f"registroPluvial{año}.csv", index=False)

# 1. Estadisticas anuales #
print(f"\nEstadísticas del año {año}:")
print(f"Precipitación máxima: {df_precipitaciones.values.max():.2f} mm")
print(f"Precipitación mínima: {df_precipitaciones.values.min():.2f} mm")
print(f"Precipitación promedio: {df_precipitaciones.values.mean():.2f} mm")

# Graficos anuales solo los guarda#
#  Grafico de barras anual #
plt.figure(figsize=(12, 6))
df_precipitaciones.mean().plot(kind='bar')
plt.title(f'Precipitación promedio mensual - {año}')
plt.xlabel('Mes')
plt.ylabel('Precipitación (mm)')
plt.tight_layout()
plt.savefig(f"{datos_dir}barras_anual_{año}.png")
plt.close()
```

```python
#  Grafico de dispersion #
plt.figure(figsize=(12, 6))
for mes in range(12):
    plt.scatter([mes+1]*31, range(1,32), c=df_precipitaciones.iloc[:,mes], cmap='Blues')
plt.colorbar(label='Precipitación (mm)')
plt.title(f'Distribución de lluvias - {año}')
plt.xlabel('Mes')
plt.ylabel('Día')
plt.tight_layout()
plt.savefig(f"{datos_dir}dispersion_{año}.png")
plt.close()

# Grafico circular anual #
plt.figure(figsize=(10, 10))
plt.pie(df_precipitaciones.mean(), labels=df_precipitaciones.columns, autopct='%1.1f%%')
plt.title(f'Distribución porcentual de lluvias por mes - {año}')
plt.savefig(f"{datos_dir}circular_anual_{año}.png")
plt.close()

# 2. Analisis mensual #
try:
    # Solicitar y corroborar el mes #
    mes = int(input("\nIngrese el mes para analizar (1-12): "))
    if mes < 1 or mes > 12:
        raise ValueError("El mes debe estar entre 1 y 12")

    mes_idx = mes - 1
    mes_nombre = df_precipitaciones.columns[mes_idx]

    #  Mostrar registros del mes #
    print(f"\nRegistros de {mes_nombre}:")
    print("Día    Precipitación (mm)")
    print("-" * 25)
    datos_mes = df_precipitaciones[mes_nombre].copy()  # Crear una copia de los datos del mes #
    for dia, valor in enumerate(datos_mes, 1):
        print(f"{dia:2d}     {valor:6.2f}")

    # Estadisticas mensuales #
    print(f"\nEstadisticas de {mes_nombre}:")
    print(f"Precipitación máxima: {datos_mes.max():.2f} mm")
    print(f"Precipitación mínima: {datos_mes.min():.2f} mm")
    print(f"Precipitación promedio: {datos_mes.mean():.2f} mm")

    # Grafico circular mensual #
    # Muestra el grafico en pantalla y lo guarda #
    plt.figure(figsize=(10, 10))

    # Filtrar valores mayores que 0 para el grafico #
    datos_grafico = datos_mes[datos_mes > 0]
    if len(datos_grafico) > 0:
        plt.pie(datos_grafico,
                labels=[f'Día {i+1}' for i in datos_grafico.index],
                autopct='%1.1f%%')
        plt.title(f'Distribución porcentual de lluvias - {mes_nombre} {año}')
        plt.savefig(f"{datos_dir}circular_mensual_{mes_nombre}_{año}.png")
        plt.show()
        plt.close()
    else:
        print(f"No hay datos de precipitación mayores que 0 para {mes_nombre}")

except ValueError as e:
    print(f"Error: {e}")
    print("Por favor, ingrese un número válido entre 1 y 12")
except Exception as e:
    print(f"Error inesperado: {e}")
```

Conclusiones

La implementación del sistema de gestión de ventas y empleados ha permitido optimizar el acceso a la información, la organización de datos y la eficiencia en la evaluación del desempeño de los empleados. Este proyecto demuestra la utilidad de integrar herramientas de programación y bases de datos para resolver necesidades administrativas, mostrando que la digitalización es una solución accesible y efectiva para empresas de diferentes escalas.