Profesores:

- Prof. Lisandro Lanfranco (Programación)
- Prof. Martín Gerlero (Bases de Datos)
- Prof. Pablo Figueroa (Ética y Deontología Profesional)

Alumnos:

- Nicolás Nahuel Barrionuevo, GitHub: @NicolasBa27.
- Diego Ares, GitHub: @diegote7.
- Fernando Gimenez Coria, GitHub: @FernandoGC.
- Raul Jara, GitHub: @r-j28.

Fecha de entrega: 10/06/2024

Objetivo

Desarrollar una pequeña aplicación que demuestre la integración de conceptos de programación y bases de datos, y que aborde aspectos éticos y profesionales según la normativa vigente en la provincia de Córdoba.

Parte 1 - Integradora: Definición de la Temática

<u>Nombre del proyecto:</u> Control de mantenimiento de Flota

Un emprendimiento rural necesita llevar el mantenimiento preventivo y correctivo de su flota de maquinaria rodante, Tractores, Cosechadoras y camiones, tanto aquellas que estén trabajando en el propio campo, como aquellas con las que presta servicio a otros campos colegas (cosecha, siembra, aplicación de agroquímicos, etc).

Para el funcionamiento del sistema se propone llevar el conteo de horas de trabajo de cada máquina y en base a su propio manual de taller, disparar alertas para realizar los mantenimientos periódicos. Adema llevar el estado de un stock de repuestos para realizar esto mantenimientos, manejando cantidades mínimas con alertas semanales para reposición).

El sistema llevara el registro de las horas trabajadas por cada máquina al final de cada jornada y también del operario a cargo ese día, con lo que se podrá generar un historial maquina/operario dando lugar a algunas evaluaciones estadísticas como frecuencia de roturas o desgastes prematuros.

Como operario puede definirse a aquel que conduce las maquinas, aquel que realiza los mantenimientos que a su vez puede pertenecer al staff o ser un operario externo.

Mediante el programa se deberá poder dar de alta y baja cada máquina, asignando cada mantenimiento preventivo indicado por su manual a la cantidad de horas determinada, indicando a su vez el repuesto a cambiar.

Para el caso de un mantenimiento o reparación que no esté discriminada en los mantenimientos preventivos, deberá existir una forma de registrar el evento, con

detalles tales como rotura, repuestos, causa, operario a cargo, medidas preventivas, etc.

Parte 2 - Programación: Desarrollo de aplicación por consola en Python

A continuación se detalla una explicación del código fuente

Descripción general

Este programa es una aplicación de gestión para una base de datos MySQL, específicamente diseñada para gestionar una flota y sus actividades relacionadas. La aplicación incluye varias funcionalidades como el manejo de maquinarias, actividades, operarios, repuestos, consumibles y proveedores. Está estructurada en tres archivos principales:

1. `main.py`: Contiene el punto de entrada del programa.
2. `database.py`: Maneja la conexión y las operaciones con la base de datos MySQL.
3. `menu.py`: Gestiona la interfaz de usuario en la consola y las interacciones con el usuario.

main.py

El archivo main.py sirve como punto de entrada del programa. Aquí se realiza lo siguiente:

- Se imprime una cabecera con el nombre del programa.
- Se crea una instancia de la clase Database para conectar a la base de datos.
- Si la conexión es exitosa, se crea una instancia de la clase Menu y se ejecuta el menú.
- Si la conexión falla, se imprime un mensaje de error.

```
from menu import Menu
from database import Database

def main():
    print("##################* SILVER AGRO
*##################\n\n#################* FLOTA CONTROL
*################\n\n")
    db = Database('localhost', 'root', 'root', 'mydb')

    if db.midb and db.cursor:
        menu = Menu(db)
        menu.ejecutar()
        db.cerrar_conexion()
    else:
        print("No se pudo establecer la conexión a la base de datos")

if __name__ == "__main__":
    main()
```

database.py

El archivo database.py contiene la clase Database, que maneja la conexión y las operaciones con una base de datos MySQL.

Principales métodos de la clase Database:

- `__init__(self, host, user, password, database)`: Inicializa la conexión a la base de datos.
- `ejecutar_consulta(self, consulta, valores=None)`: Ejecuta una consulta SQL (INSERT, UPDATE, DELETE).
- `obtener_resultados(self, consulta, valores=None)`: Ejecuta una consulta SQL (SELECT) y retorna los resultados.
- `cerrar_conexion(self)`: Cierra la conexión a la base de datos y el cursor.

```
import mysql.connector

class Database:
    def __init__(self, host, user, password, database):
        try:
            self.midb = mysql.connector.connect(
                host=host,
                user=user,
                password=password,
                database=database
            )
            self.cursor = self.midb.cursor()
            print("Conexión establecida exitosamente")
        except:
            print("Error: ", mysql.connector.Error)
            self.midb = None
            self.cursor = None
```

```
    def obtener_resultados(self, consulta, valores=None):
        self.cursor.execute(consulta, valores)
        return self.cursor.fetchall()

    def ejecutar_consulta(self, consulta, valores=None):
        self.cursor.execute(consulta, valores)
        self.midb.commit()

    def cerrar_conexion(self):
        self.cursor.close()
        self.midb.close()
```

menu.py

La clase Menu es responsable de la interfaz de usuario en la consola y maneja la lógica para las opciones del menú principal y los submenús.

```
from database import Database

class Menu:
    def __init__(self, db):
        self.db = db
```

- **Constructor (`__init__`)**
  - Inicializa la instancia de la clase con una referencia a la base de datos.

Método ejecutar

Este método es el bucle principal que mantiene el programa en ejecución hasta que el usuario decide salir.

```
def ejecutar(self):
    self.alerta_mantenimiento()
    while True:
        self.mostrar_menu()
        opcion = input("Seleccione una opción: ")

        if opcion == '1':
            self.mostrar_altas()
        elif opcion == '2':
            self.mostrar_bajas()
        elif opcion == '3':
            self.programar_actividad()
        elif opcion == '4':
            self.historial_maquina()
        elif opcion == '5':
            self.historial_operario()
        elif opcion == '6':
            self.informe_almacen()
        elif opcion == '7':
            self.carga_horas_diarias()
        elif opcion == '8':
            self.ver_estadisticas_uso()
        elif opcion == '9':
            self.salir()
```

```
            break
        else:
            print("Opción no válida. Por favor, seleccione nuevamente.")
```

- **Método alerta_mantenimiento**
    - Este método (asumiendo su implementación en otra parte del código) probablemente notifica sobre el mantenimiento pendiente o necesario.
- **Bucle del Menú Principal**
    - Muestra el menú principal.
    - Lee la entrada del usuario.
    - Dirige la ejecución según la opción seleccionada, llamando a métodos correspondientes a cada funcionalidad.

Método mostrar_menu

Muestra las opciones disponibles en el menú principal.

```
def mostrar_menu(self):
    print("\nMenú de opciones:")
    print("1. Ingresar Submenu Altas")
    print("2. Ingresar Submenu Bajas")
    print("3. Programar Actividad")
    print("4. Historial Maquina")
    print("5. Historial Operario")
    print("6. Informe Almacen")
    print("7. Carga Diaria Horas")
    print("8. Estadisticas Generales")
    print("9. Salir")
```

4. Métodos de Submenús

mostrar_altas

Muestra y gestiona el submenú de altas, permitiendo al usuario elegir qué entidad desea agregar.

```
def mostrar_altas(self):
    print("\nSubMenú de Altas:")
    print("1. Alta Maquina")
    print("2. Alta Actividad")
    print("3. Alta Operario")
    print("4. Alta Repuesto")
    print("5. Alta Consumible")
    print("6. Alta Proveedor")
    print("9. Salir")

    opcion = input("Seleccione una opción: ")
    if opcion == '1':
        self.alta_maquina()
    elif opcion == '2':
        self.alta_actividad()
    elif opcion == '3':
        self.alta_operario()
```

```
    elif opcion == '4':
        self.alta_repuesto()
    elif opcion == '5':
        self.alta_consumible()
    elif opcion == '6':
        self.alta_proveedor()
    elif opcion == '9':
        self.ejecutar()
    else:
        print("Opción no válida. Por favor, seleccione nuevamente.")
```

- **Métodos de Alta**
    - alta_maquina
    - alta_actividad
    - alta_operario
    - alta_repuesto
    - alta_consumible
    - alta_proveedor

mostrar_bajas

Muestra y gestiona el submenú de bajas, permitiendo al usuario elegir qué entidad desea eliminar.

```
def mostrar_bajas(self):
    print("\nSubMenú de Bajas:")
    print("1. Baja Maquina")
    print("2. Baja Actividad")
    print("3. Baja Operario")
    print("4. Baja Repuesto")
    print("5. Baja Consumible")
    print("6. Baja Proveedor")
    print("9. Salir")

    opcion = input("Seleccione una opción: ")
    if opcion == '1':
        self.baja_maquina()
    elif opcion == '2':
        self.baja_actividad()
    elif opcion == '3':
        self.baja_operario()
    elif opcion == '4':
        self.baja_repuesto()
    elif opcion == '5':
        self.baja_consumible()
    elif opcion == '6':
        self.baja_proveedor()
    elif opcion == '9':
        self.ejecutar()
    else:
        print("Opción no válida. Por favor, seleccione nuevamente.")
```

- **Métodos de Baja**
    - baja_maquina
    - baja_actividad
    - baja_operario
    - baja_repuesto
    - baja_consumible
    - baja_proveedor

Otras Funcionalidades del Menú Principal

- **Programar Actividad**
  - Método: programar_actividad
  - Funcionalidad: Permite programar una actividad para una máquina u operario.
- **Historial Maquina**
  - Método: historial_maquina
  - Funcionalidad: Muestra el historial de actividades de una máquina específica.
- **Historial Operario**
  - Método: historial_operario
  - Funcionalidad: Muestra el historial de actividades de un operario específico.
- **Informe Almacen**
  - Método: informe_almacen
  - Funcionalidad: Genera y muestra un informe del inventario de almacén, incluyendo repuestos y consumibles.
- **Carga Diaria Horas**
  - Método: carga_horas_diarias
  - Funcionalidad: Permite registrar las horas diarias trabajadas por los operarios.
- **Estadísticas Generales**
  - Método: ver_estadisticas_uso
  - Funcionalidad: Muestra estadísticas generales de uso de máquinas y operarios, proporcionando una visión general de la eficiencia y el rendimiento.
- **Salir**
  - Método: salir
  - Funcionalidad: Termina la ejecución del programa de manera segura.

Ejemplo de Métodos de Alta y Baja

Cada método de alta y baja sigue una estructura similar, pidiendo información específica al usuario e interactuando con la base de datos a través de la instancia de Database. Aquí hay ejemplos genéricos:

```
def alta_maquina(self):
    nombre = input("Ingrese el nombre de la máquina: ")
    tipo = input("Ingrese el tipo de máquina: ")
    # Más entradas según sea necesario
    consulta = "INSERT INTO maquinas (nombre, tipo) VALUES (%s, %s)"
    self.db.ejecutar_consulta(consulta, (nombre, tipo))
    print("Máquina agregada exitosamente.")

def baja_maquina(self):
    id_maquina = input("Ingrese el ID de la máquina a eliminar: ")
    consulta = "DELETE FROM maquinas WHERE id = %s"
    self.db.ejecutar_consulta(consulta, (id_maquina,))
    print("Máquina eliminada exitosamente.")
```

Conclusión

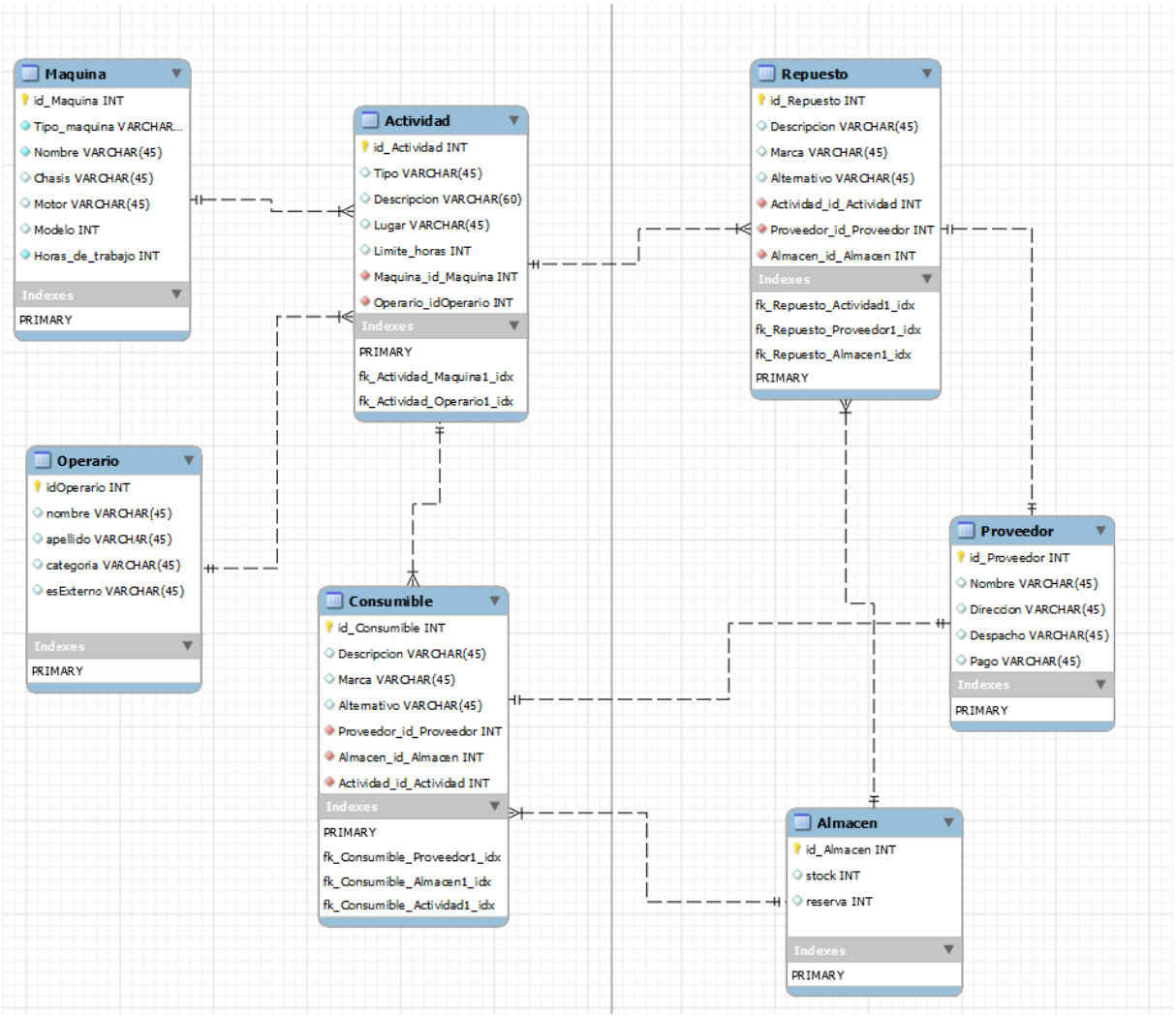
El archivo menu.py proporciona una interfaz de usuario robusta y extensible para interactuar con la base de datos, gestionando diferentes aspectos de una flota. La estructura modular del menú y submenús facilita la navegación y operación, permitiendo una gestión eficiente de las maquinarias, actividades, operarios, y más.

El código fuente completo será cargado adjunto al presente informe, no obstante, en el repositorio de GitHub, puede accederse a todos los archivos del proyecto.

https://github.com/ISPC-TST-MOD-PROGR-2024/Proyecto-final-modulo-programador.git

Parte 3 - Base de Datos: Diseño y Normalización

Modelo relacional de la base de datos.



De acuerdo con el modelo relacional planteado se adjunta base de datos con datos de ejemplo cargados.

Listado de consultas SQL

1. Alta de Entidades

`alta_maquina`

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO maquinas (nombre, tipo) VALUES (%s, %s)
```

`alta_actividad`

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO actividades (descripcion, fecha) VALUES (%s, %s)
```

`alta_operario`

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO operarios (nombre, dni) VALUES (%s, %s)
```

`alta_repuesto`

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO repuestos (nombre, cantidad) VALUES (%s, %s)
```

`alta_consumible`

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO consumibles (nombre, cantidad) VALUES (%s, %s)
```

`alta_proveedor`

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO proveedores (nombre, contacto) VALUES (%s, %s)
```

2. Baja de Entidades

`baja_maquina`

- **Consulta SQL:**

```
sql
Copiar código
DELETE FROM maquinas WHERE id = %s
```

baja_actividad

- **Consulta SQL:**

```
sql
Copiar código
DELETE FROM actividades WHERE id = %s
```

baja_operario

- **Consulta SQL:**

```
sql
Copiar código
DELETE FROM operarios WHERE id = %s
```

baja_repuesto

- **Consulta SQL:**

```
sql
Copiar código
DELETE FROM repuestos WHERE id = %s
```

baja_consumible

- **Consulta SQL:**

```
sql
Copiar código
DELETE FROM consumibles WHERE id = %s
```

baja_proveedor

- **Consulta SQL:**

```
sql
Copiar código
DELETE FROM proveedores WHERE id = %s
```

3. Otras Funcionalidades

programar_actividad

- **Consulta SQL:**

```
sql
Copiar código
INSERT INTO programaciones (id_maquina, id_actividad, fecha) VALUES
(%s, %s, %s)
```

historial_maquina

- **Consulta SQL:**

```
sql
Copiar código
SELECT * FROM historial_maquinas WHERE id_maquina = %s
```

historial_operario

- **Consulta SQL:**

```
sql
Copiar código
SELECT * FROM historial_operarios WHERE id_operario = %s
```

informe_almacen

- **Consulta SQL:**

```
SELECT * FROM almacen
```

carga_horas_diarias

- **Consulta SQL:**

```
INSERT INTO horas_diarias (id_operario, horas, fecha) VALUES (%s, %s,
%s)
```

Parte 4 - Aspectos Éticos y Profesionales

Aspectos Éticos y Profesionales

La regulación del ejercicio profesional en informática es fundamental para salvaguardar los derechos y la seguridad de los usuarios. **La Ley 7.642 y el Estatuto del Consejo Profesional de Ciencias Informáticas de Córdoba** proporcionan un marco ético y legal que garantiza la responsabilidad y la confiabilidad del trabajo de los informáticos.

Este Consejo establece la normativa que regula el ejercicio de las profesiones relacionadas con las ciencias informáticas en la provincia. Su propósito principal es

regular la práctica profesional, establecer estándares éticos y fomentar el desarrollo y la excelencia en este campo.

Estas normativas buscan garantizar la calidad y seguridad en el desarrollo y aplicación de tecnologías informáticas, protegiendo los derechos y la seguridad de los usuarios. El Código de Ética del Consejo Profesional de Ciencias Informáticas de Córdoba establece principios morales que deben guiar el ejercicio profesional, tales como responsabilidad, respeto, confidencialidad, trabajo en equipo, honestidad y justicia.

Para el desarrollo del proyecto de **Control de Mantenimiento de Flota**, es esencial contar con un sólido enfoque ético y profesional que asegure el funcionamiento eficiente y seguro de la flota de maquinaria rodante en el emprendimiento rural.

El sistema propuesto gestiona el mantenimiento preventivo y correctivo de la flota, beneficiándose de la ética profesional al asegurar la integridad y seguridad de las operaciones agrícolas. Basado en las horas de trabajo y los manuales de taller de cada máquina, garantiza la realización oportuna de los mantenimientos necesarios, evitando posibles fallas y maximizando la vida útil de los equipos.

Además, el registro de horas trabajadas por máquina y operario promueve la responsabilidad y la transparencia en la gestión de recursos humanos, permitiendo evaluaciones estadísticas que orienten decisiones estratégicas en capacitación y asignación de tareas de acuerdo a la capacidad máquina-usuario.

La implementación de alertas para el control de stock de repuestos también evidencia un compromiso con la eficiencia operativa y la disponibilidad de recursos, minimizando tiempos de inactividad por falta de repuestos.

La inclusión de un registro detallado de eventos de mantenimiento o reparación no contemplados en los mantenimientos preventivos demuestra una visión proactiva hacia la resolución de problemas y el aprendizaje constante, promoviendo una cultura de mejora continua en la gestión de la maquinaria.

El enfoque ético y profesional, respaldado por la legislación pertinente, proporciona un sólido fundamento para el desarrollo exitoso de este proyecto, asegurando la eficiencia, seguridad y sustentabilidad del emprendimiento rural.

Es fundamental considerar varios aspectos éticos y profesionales al trabajar en este proyecto, regidos por la Ley 7.642 y el Estatuto de Regulación del Consejo Profesional de Ciencias Informáticas de Córdoba:

• **Confidencialidad y Protección de Datos:** Manejar la información de los operarios, las máquinas y los mantenimientos de manera confidencial y respetando la privacidad de los datos personales de acuerdo con las disposiciones legales y éticas.

• **Transparencia y Responsabilidad:** Mantener registros precisos y transparentes de las operaciones del sistema, incluyendo el mantenimiento de la flota y el manejo del stock de repuestos, para garantizar la responsabilidad y la rendición de cuentas en todas las actividades relacionadas con el proyecto.

• **Calidad y Seguridad:** Cumplir con los estándares de calidad y seguridad establecidos por las normativas y prácticas éticas en el campo de las ciencias informáticas, asegurando que el sistema sea confiable, preciso y seguro en su funcionamiento, especialmente considerando su impacto en la operación de la maquinaria y la seguridad de los operarios.

• **Formación y Capacitación:** Fomentar la formación y capacitación continua del personal involucrado en el uso y mantenimiento del sistema, contribuyendo a un mejor desempeño en sus funciones y promoviendo una cultura de aprendizaje y mejora continua.

• **Equidad y No Discriminación:** Evitar cualquier forma de discriminación en el trato hacia los operarios, asegurando igualdad de oportunidades y trato justo a todos los involucrados en el proyecto, ya sean parte del staff interno o externo.

• **Cumplimiento Legal y Ético:** Familiarizarse y cumplir con todas las disposiciones legales y éticas relevantes relacionadas con el ejercicio profesional en el campo de las ciencias informáticas, respetando los derechos de todas las partes involucradas.

Adherirse a estos principios éticos y profesionales garantizará el cumplimiento de la ley y las regulaciones pertinentes, contribuyendo al éxito y la sostenibilidad a largo plazo del proyecto.