“Universidad Autónoma de Nuevo León”

Facultad de Ingeniería Mecánica y Eléctrica

Administración de base de datos

Manual técnico

EQUIPO 4	
Matricula	**Nombre**
1858984	Jenifer Alejandra Rodríguez Méndez
1815466	Alondra Monsserrat Ramírez Proa
1808135	Cynthia Guadalupe Vega Villalba
1543003	Selene Aydee Abrego Bonilla
1802224	Maricarmen Regalado Espinoza
1733222	Fernando Uriel Zapata Rivas
1747264	Roberto Alan Rodríguez Monroy
1461955	José Alberto Vázquez Martínez
1578267	Juan Carlos Romero Ortiz
1811331	Braulio Rafael Torrez Ruiz
1626560	Diego Alan Sánchez Partida
1603039	William Gilberto Vargas Delgado

Salón: 4206 Grupo: 002

Docente: Dra. Norma Edith Marín Martínez

Día: Martes Hora: N1-N3

Semestre Agosto-Diciembre 2021

Contenido

Índice de imágenes

Manual técnico

Propósito

El propósito del proyecto es la creación de una aplicación para ordenar en agrupaciones las materias que llevará un alumno en un cierto semestre, carrera y turno en la Facultad de Ingeniería Mecánica y Eléctrica.

Alcance

El uso de la aplicación está dado solo para un usuario el cual es el coordinador relacionado al área que proporciona las materias a los alumnos.

Aunque la aplicación es un sencillo programa para gestión de una base de datos, da oportunidad a crecer de acuerdo con las necesidades del usuario también debido al lenguaje en el que está construido que es fácil de aprender.

Programas utilizados

El lenguaje en el que fue desarrollada la aplicación es Python 3.9, como editor se utilizó SublimeText pero eso es opcional para el equipo de trabajo. Además, se utilizó el editor de diseño de prototipos FIGMA con la ayuda de Proxlight Designer para desarrollar los diseños del sistema.

Construcción del proyecto

Interfaz inicio de sesión:

Ilustración 1. Pantalla de inicio de sesión.

La parte de la interfaz de inicio de sesión está constituida por la función ingreso() la cual nos permite comprobar el usuario y contraseña con los establecidos dentro de la aplicación comparándolos.

```
def ingreso(self):
    NombreUsuarioAdmin = "admin"
    PasswordAdmin = "ROOT1747264"
    self.db = Conexion()
    try:
        self.db.conectar()
    except:
        print("Error al conectar")

    self.recogerinformacion=self.db.cursor.execute(f"SELECT * FROM Usuarios").fetchall()
    count1=0
    for row in self.recogerinformacion:
        count1=1
    if count1 == 0:
        pass
    else:
        if self.nombreIn.get() != "" and self.passwordIn.get() != "":
            for i in self.recogerinformacion:
                valor=i
                usuario=valor[1]
                contraseña=valor[2]
                if usuario == self.nombreIn.get() and contraseña == self.passwordIn.get():
                    contador_de_salida=1
                    break
                elif NombreUsuarioAdmin == self.nombreIn.get() and PasswordAdmin == self.passwordIn.get():
                    contador_de_salida=1
                    break
                else:
                    contador_de_salida=0
            if contador_de_salida==1:
                messagebox.showinfo("Bienvenida", f"Bienvenido {self.nombreIn.get()}")
                self.window.destroy()
                #window= Tk()
                window = ThemedTk(theme="adapta")
                entrar_menu=Menu_organizacion(window,"Menú",1350,670)
                window.mainloop()
            else:
                messagebox.showinfo("Error", "Usuario o contraseña incorrectos.")
        else:
            messagebox.showinfo("Error", f"Se deben llenar todos los campos")
```

Ilustración 2. Código de función: ingreso()

Interfaz Visualización:

Ilustración 3. Pantalla interfaz de Visualización de datos.

La interfaz visualización como su propio nombre indica es la visualización de los datos.

Funciones:

- Mostrar materias de la agrupación seleccionada (def click_treeview)
- Mostrar agrupaciones seleccionadas en la ListBox (def selección_del_combobox)
- Mostrar todas las agrupaciones (def mostrarTodo)
- Ir al CRUD (def ir_agregar)

Función click_treeview

Esta función nos permite mostrar las materias que se encuentran en la agrupación seleccionada dentro de la tabla “Grupos organizados”. Toma la variable “values” para registrar lo seleccionado dentro de la tabla, values[0] corresponde al ID del registro, a continuación recoge los datos de las Agrupaciones y compara esta ID con aquellos registros que contenga el mismo ID de registro de agrupación. Para finalizar incorpora los datos de dichos registros en la tabla “Materias del grupo”.

```
def click_treeview(self,e):
    seleccion = self.tabla_grupos.focus()
    values = self.tabla_grupos.item(seleccion,"values")

    self.tabla.delete(*self.tabla.get_children())

    self.recogerinformacion=self.db.cursor.execute(f"SELECT * FROM Grupos_ordenados").fetchall()
    if len(self.recogerinformacion) != 0:
        for i in self.recogerinformacion:
            if values[0] == i[1]:
                value1=i[1]
                value2=i[2]
                value3=i[3]
                value4=i[4]
                value5=i[5]
                value6=i[6]
                value7=i[7]
                value8=i[8]
                value10=i[10]
                value11=i[11]
                self.tabla.insert(parent="",index="end", text="", values=(value1,value2,value3,value4,value5,value6,value7,value8,value10,value11))
```

Ilustración 4. Código de la función: click_treeview

Función selección_del_combobox

Esta función es utilizada para filtrar las agrupaciones por carrera, semestre y turno. Los combobox están enlazados con condicionales para cualquier situación que se dé y se requiera encontrar la información seleccionada en ellos. En cada caso se reinicia la tabla con los nuevos valores comparados. Los posibles valores son prácticamente si se ha seleccionado o no el combobox.

```
def seleccion_del_combobox(self, event):
    self.recogerinformacion_grupos=self.db.cursor.execute(f"SELECT * FROM Agrupacion").fetchall()
    if len(self.recogerinformacion_grupos) != 0:
        self.tabla_grupos.delete(*self.tabla_grupos.get_children())
        for i in self.recogerinformacion_grupos:
            value0=i[0];value2=i[2];value3=i[3];value5=i[5]

            # si estan seleccionados los 3 combobox
            if value2 == self.comb_carreras.get() and value5 == self.comb_semestres.get() and value3 == self.comb_turnos.get():
                self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))

            # si estan seleccionados solo los 2 ultimos
            elif self.comb_carreras.get() == "" and value5 == self.comb_semestres.get() and value3 == self.comb_turnos.get():
                self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))

             # si estan seleccionados solo los 2 primeros
            elif self.comb_carreras.get() == value2 and self.comb_semestres.get() == value5 and self.comb_turnos.get() == "":
                self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))

            # si estan seleccionadoel primero
            elif self.comb_carreras.get() == value2 and self.comb_semestres.get() == "" and self.comb_turnos.get() == "":
                self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))

            # si estan seleccionados solo el ultimo
            elif self.comb_carreras.get() == "" and self.comb_semestres.get() == "" and value3 == self.comb_turnos.get():
                self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))

            # si estan seleccionado el segundo
            elif self.comb_carreras.get() == "" and self.comb_semestres.get() == value5 and self.comb_turnos.get() == "":
                self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))
```

Ilustración 5. Código de la función seleccion_del_combobox

Función mostrarTodo

Esta función sencillamente muestra todas las agrupaciones en existencia tomándolas de la base de datos y reinicia los combobox.

```
#_____________________________Seleccion de los Combobox_____________________#
def mostrarTodo(self):
    self.recogerinformacion_grupos=self.db.cursor.execute(f"SELECT * FROM Agrupacion").fetchall()
    if len(self.recogerinformacion_grupos) != 0:
        self.tabla_grupos.delete(*self.tabla_grupos.get_children())
        for i in self.recogerinformacion_grupos:
            value0=i[0]
            value2=i[2]
            value3=i[3]
            value5=i[5]
            self.tabla_grupos.insert(parent="",index="end", text="", values=(value0,value2,value3,value5))

    self.comb_carreras.current(0)
    self.comb_semestres.current(0)
    self.comb_turnos.current(0)
```

Ilustración 6. Código de la función mostrarTodo.

Función ir_agregar

Esta función cierra la ventana actual y llama a la función para ir a la ventana CRUD.

```
def ir_agregar(self):
    self.window.destroy()
    window=ThemedTk(theme="adapta")
    llamada = Agregar(window, "Agregar registro", 1350, 670)
    window.mainloop()
```

Ilustración 7. Código de función ir_agregar.

Interfaz CRUD

Ilustración 8. CRUD modo Create.

La interfaz CRUD es la relacionada con la base de datos, tanto para crear, leer, actualizar y eliminar registros.

Funciones:

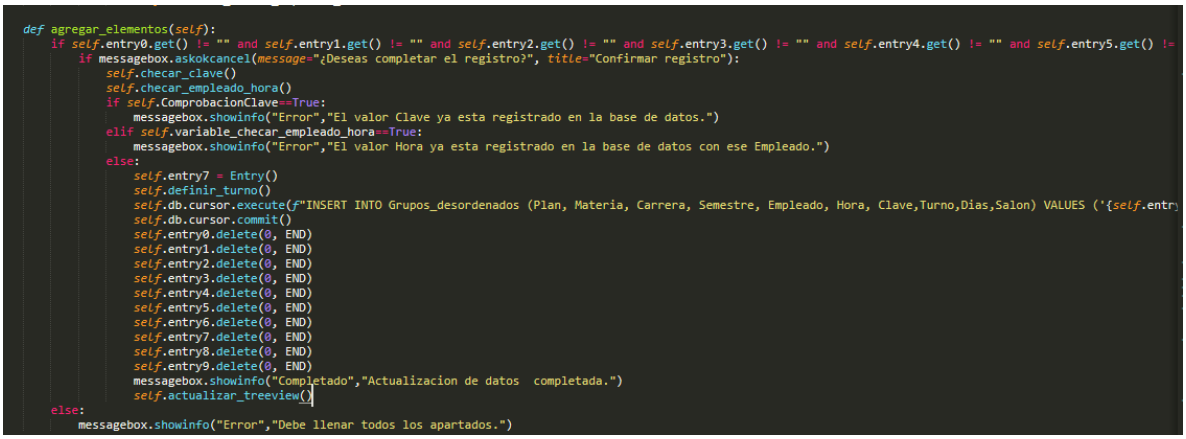
- Agregar registro (agregar_elementos)
- Eliminar registro (eliminar_registro)
- Actualizar registro (actualizar)
- Crear los grupos (crear_grupos)

Función agregar_elementos

Esta función es la utilizada para agregar los elementos del formulario de la interfaz a la base de datos.

Primero se realiza la comprobación de que todos los apartados del formulario han sido llenados. A continuación, se hace la confirmación del registro con el usuario, después se llama a una función auxiliar que verifica si la clave proporcionada ya está registrada en la base de datos y otra que verifica que el empleado (maestro) ya tiene una clase a una hora asignada.

Si se cumplen las condiciones adecuadas se usa otra función auxiliar para asignar el turno de la materia y a continuación se agregan los datos a la tabla en la base de datos. Para terminar, se manda llamar la función auxiliar para actualizar la tabla de la interfaz.



```
def agregar_elementos(self):
    if self.entry0.get() != "" and self.entry1.get() != "" and self.entry2.get() != "" and self.entry3.get() != "" and self.entry4.get() != "" and self.entry5.get() !=
        if messagebox.askokcancel(message="¿Deseas completar el registro?", title="Confirmar registro"):
            self.checar_clave()
            self.checar_empleado_hora()
            if self.ComprobacionClave==True:
                messagebox.showinfo("Error","El valor Clave ya esta registrado en la base de datos.")
            elif self.variable_checar_empleado_hora==True:
                messagebox.showinfo("Error","El valor Hora ya esta registrado en la base de datos con ese Empleado.")
            else:
                self.entry7 = Entry()
                self.definir_turno()
                self.db.cursor.execute(f"INSERT INTO Grupos_desordenados (Plan, Materia, Carrera, Semestre, Empleado, Hora, Clave,Turno,Dias,Salon) VALUES ('{self.entry
                self.db.cursor.commit()
                self.entry0.delete(0, END)
                self.entry1.delete(0, END)
                self.entry2.delete(0, END)
                self.entry3.delete(0, END)
                self.entry4.delete(0, END)
                self.entry5.delete(0, END)
                self.entry6.delete(0, END)
                self.entry7.delete(0, END)
                self.entry8.delete(0, END)
                self.entry9.delete(0, END)
                messagebox.showinfo("Completado","Actualizacion de datos  completada.")
                self.actualizar_treeview()
    else:
        messagebox.showinfo("Error","Debe llenar todos los apartados.")
```

Ilustración 9. Código de función agregar_elementos.

Función eliminar_registro

Esta función elimina el registro seleccionado en la tabla de la interfaz. Primero se pide la confirmación al usuario, a continuación, se registra el valor del registro seleccionado de la tabla en “values”. Se compara values[0] con la base de datos y cuando lo encuentre será eliminado el registro y se actualizará la tabla de la interfaz.

```
def eliminar_registro(self):
    decision2=messagebox.askquestion("Confirmar","¿Seguro que quieres eliminar el registro?")
    if decision2 == "yes":
        seleccion = self.tabla.focus()
        values = self.tabla.item(seleccion,"values")

        self.infoC=self.db.cursor.execute(f"SELECT * FROM Grupos_desordenados").fetchall()
        if len(self.infoC) != 0:
            if values =="":
                messagebox.showinfo("Error","No ha seleccionado un registro")
            else:
                valor_eliminar = values[0]
                self.db.cursor.execute(f"DELETE FROM Grupos_desordenados WHERE Id = ?", valor_eliminar)
                self.db.cursor.commit()
                messagebox.showinfo("Completado","Registro eliminado.")
                self.actualizar_treeview()
```

Ilustración 10. Código de función eliminar_registro

Función actualizar

Esta función actualiza el registro seleccionado por la tabla de la interfaz. Es simple y recoge los valores guardados en la ventana extra para editar un registro. Y como todas las demás funciones, actualiza la tabla de la interfaz.

```
#_____FUNCION PARA ACTUALIZAR VALORES EDITADOS_____#
def actualizar(self):
    decision2=messagebox.askquestion("Confirmar","¿Seguro que quieres actualizar el registro?")
    if decision2 == "yes":
        self.extra_combobox_turno = Entry()
        self.definir_turno2()

        self.db.cursor.execute("UPDATE Grupos_desordenados SET Plan = ? WHERE Id = ?",self.extra_plan.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Materia = ? WHERE Id = ?",self.extra_materia.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Carrera = ? WHERE Id = ?",self.extra_combobox_carrera.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Semestre = ? WHERE Id = ?",self.extra_combobox_semestre.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Empleado = ? WHERE Id = ?",self.extra_empleado.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Hora = ? WHERE Id = ?",self.extra_combobox_horas.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Clave = ? WHERE Id = ?",self.extra_clave.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Turno = ? WHERE Id = ?",self.extra_combobox_turno.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Dias = ? WHERE Id = ?",self.extra_combobox_dias.get(),self.valor_extra0)
        self.db.cursor.execute("UPDATE Grupos_desordenados SET Salon = ? WHERE Id = ?",self.extra_salon.get(),self.valor_extra0)

        self.db.cursor.commit()
        self.top.destroy()
        messagebox.showinfo("Completado","Actualizacion de datos  completada.")
        self.actualizar_treeview()
```

Ilustración 11. Código función actualizar.

Función crear_grupos

Esta función es la encargada de crear las agrupaciones. No es una función tan compleja, pero requiere mucha atención al usar los datos debido a las condiciones a las que se tienen que someter para crear una Agrupación.

Primeramente, se comprueba que haya al menos algunos valores en la base de datos, a continuación, recorre la base de datos y comprueba si el registro ya se encuentra o no en una agrupación.

```
def crear_grupos(self):
    self.nueva_tabla_access()
    self.info_crear_grupos=self.db.cursor.execute(f"SELECT * FROM Tabla_temporal").fetchall()

    if len(self.info_crear_grupos) != 0:
        for i in self.info_crear_grupos:
            comprobar_grupos_ordenados=self.db.cursor.execute(f"SELECT * FROM Grupos_ordenados").fetchall()
            comprobation=False
            for g6 in comprobar_grupos_ordenados:
                v1=g6[0]
                v2=i[0]
                if v2 == v1 or i==None:
                    comprobation=True
```

Ilustración 12. Código de función crear_grupos (parte1).

La siguiente sección separa cada dato del registro en variables y añade el registro en una tabla donde se encuentran las materias que han sido añadidas a una agrupación, además de añadir esa primera materia como referencia para la agrupación.

```
                comprobation=True
        if comprobation==False:
            valorA0= i[0]; valorA1= i[1]; valorA2= i[2]; valorA3= i[3]; valorA4= i[4]; valorA5= i[5]; valorA6= i[6]; valorA7= i[7]; valorA8= i[8];valorA9= i[9];v

            self.db.cursor.execute(f"INSERT INTO Agrupacion (Plan, Carrera, Turno,id_primer_materia,semestre) VALUES ('{valorA1}','{valorA4}','{valorA8}','{valor
            self.db.cursor.commit()

            self.info_agrupaciones=self.db.cursor.execute(f"SELECT * FROM Agrupacion").fetchall()
            for a in self.info_agrupaciones: #Conseguir la info de agrupaciones
                valorlidadndo = a[0]   #ID DEL GRUPO
                VALORC4 = a[4]   # ID DE LA MATERIA CON LA QUE SE DIO DE ALTA LA AGRUPACION

                if int(valorA0) == int(VALORC4):
                    VALORC0 = valorlidadndo
                    self.db.cursor.execute(f"INSERT INTO Grupos_ordenados (Id,Grupo_asignado,Plan, Clave,Materia, Carrera, Semestre, Empleado, Hora,Turno,Dias,Sa
                    self.db.cursor.execute(f"DELETE FROM Tabla_temporal WHERE Id = ?", valorA0)
                    self.db.cursor.commit()
```

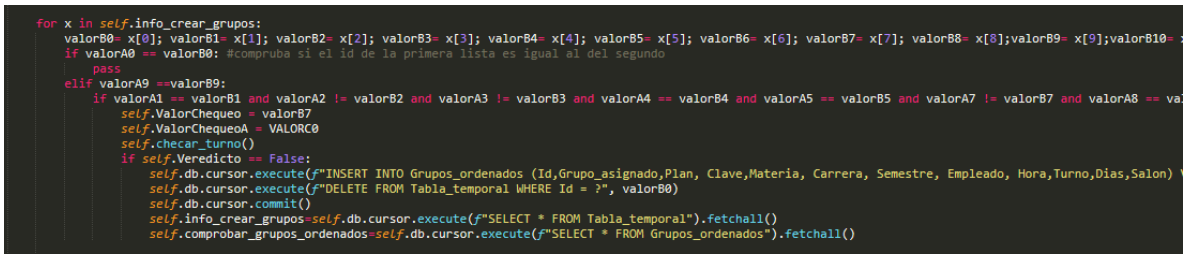
Ilustración 13. Código de función crear_grupos (parte 2).

La siguiente sección recorre los demás registros y los asigna a variables para ser comparados al primer registro asignado. Como se observa se van comparando las variables de acuerdo con lo requerido en los grupos.

Las condiciones son:

- Las materias **deben** tener asignado el mismo plan de estudio, carrera, semestre y turno para pertenecer al mismo grupo.

- Las materias **no deben** tener asignado el mismo nombre de materia y hora clase (excluyendo si son en diferentes días).



```
for x in self.info_crear_grupos:
    valorB0= x[0]; valorB1= x[1]; valorB2= x[2]; valorB3= x[3]; valorB4= x[4]; valorB5= x[5]; valorB6= x[6]; valorB7= x[7]; valorB8= x[8];valorB9= x[9];valorB10= 
    if valorA0 == valorB0: #compruba si el id de la primera lista es igual al del segundo
        pass
    elif valorA9 ==valorB9:
        if valorA1 == valorB1 and valorA2 != valorB2 and valorA3 != valorB3 and valorA4 == valorB4 and valorA5 == valorB5 and valorA7 != valorB7 and valorA8 == va
            self.ValorChequeo = valorB7
            self.ValorChequeoA = VALORC0
            self.checar_turno()
            if self.Veredicto == False:
                self.db.cursor.execute(f"INSERT INTO Grupos_ordenados (Id,Grupo_asignado,Plan, Clave,Materia, Carrera, Semestre, Empleado, Hora,Turno,Dias,Salon) 
                self.db.cursor.execute(f"DELETE FROM Tabla_temporal WHERE Id = ?", valorB0)
                self.db.cursor.commit()
                self.info_crear_grupos=self.db.cursor.execute(f"SELECT * FROM Tabla_temporal").fetchall()
                self.comprobar_grupos_ordenados=self.db.cursor.execute(f"SELECT * FROM Grupos_ordenados").fetchall()
```

Ilustración 14. Código de función crear grupos (parte3).