



UNIVERSIDAD NACIONAL DE TRUJILLO

Facultad de Ingeniería

Escuela Profesional de Ingeniería Mecatrónica

DESARROLLO DE UNA APLICACIÓN PARA LA GESTIÓN DE HISTORIALES MÉDICOS DE LOS PACIENTES DE CENTROS MÉDICOS

Trabajo de Investigación formativa

Programación I

AUTOR (es):

**Sánchez Rojas, Jhonatan Artemio
Valdiviezo Jiménez, Víctor Javier
Vigo Villar, Cristhian Aaron**

DOCENTE :

Asto Rodriguez, Emerson Máximo

CICLO :

III

**Trujillo, Perú
2022**

Resumen

El presente informe consiste en la elaboración de un proyecto de programación que busca la implementación de historiales clínicos digitales en aras de la agilización en el servicio médico, mediante la creación de un programa de escritorio, el cuál contribuya al almacenamiento y consultoría de historiales clínicos, donde el medico tendría acceso rápido a la ‘hoja de vida’, referida a asuntos netamente clínicos, del paciente, como atenciones, diagnósticos, prescripciones, tratamientos, sus datos, entre otros. El desarrollo del software requerirá del uso de herramientas de programación como Python, visual studio code, biblioteca gráfica Tkinter y el sistema de gestión para una base de datos SQLite. La motivación que sigue el proyecto es la busca de la mejora progresiva en la atención médico-paciente, evitando así, el desgaste de los procesos internos administrativos en los centros médicos como el uso de hojas de papel y archivos físicos.

***Palabras Claves:* Python, Tkinter, SQLite, historial médico**

Abstract

This report consists of the elaboration of a programming project that seeks the implementation of digital medical records in order to streamline the medical service, through the creation of a desktop program, which contributes to the storage and consulting of clinical records, where the doctor would have quick access to the 'resume', referred to purely clinical matters, of the patient, such as care, diagnoses, prescriptions, treatments, their data, among others. The development of the software will require the use of programming tools such as Python, visual studio code, Tkinter graphic library and the management system for a SQLite database. The motivation followed by the project is the search for progressive improvement in medical-patient care, thus avoiding the wear and tear of internal administrative processes in medical centers such as the use of sheets of paper and physical files.

Keywords: Python, Tkinter, SQLite, medical historial

Tabla de Contenidos

Resumen	2
Abstract	3
Capítulo 1 Introducción	5
1.1. Realidad Problemática	5
1.2. Formulación del Problema	10
1.3. Objetivos	10
1.4. Justificación	10
Capítulo 2 Marco Teórico	11
Capítulo 3 Materiales y Métodos	14
3.1. Descripción general de los procedimientos	14
3.2. Desarrollo de los procedimientos	15
3.2.1. Implementación de la Base de Datos:	15
3.2.2. Elaboración del código ejecutor del programa y de la interfaz principal:	19
3.2.3. Realización de la conexión del programa con la base de datos:	19
3.2.4. Elaboración de los iconos gráficos de la interfaz y su funcionalidad:	29
Capítulo 4 Resultados	35
4.1. Implementación de la Base de Datos	35
4.2. Elaboración del código ejecutor del programa y de la interfaz principal	36
4.3. Realización de la conexión del programa con la base de datos	37
4.4. Elaboración de los iconos gráficos de la interfaz y su funcionalidad	37
Capítulo 5 Conclusiones y Recomendaciones	39
5.1. Conclusiones	39
5.2. Recomendaciones	40
Referencias Bibliográficas	41
ANEXOS	45

Capítulo 1

Introducción

Aun estando en un siglo donde la tecnología es lo que se prioriza, siguen existiendo centros médicos que tramitan las historias clínicas manualmente, en otras palabras, las tienen almacenadas de manera física. Y en la mayoría de circunstancias, este hecho provoca un gran retraso en el tiempo de servicio que se le da a los pacientes que llegan para ser atendidos en dicho centro. El hecho de hacer la búsqueda de una historia clínica en un almacén y luego llevarlo hasta el consultorio médico, aumenta notoriamente el tiempo de atención al paciente. Este proyecto busca dar una solución al diseñar un sistema de control para historiales médicos, el cual permita una gestión más organizada y automatizada de las historias clínicas en un centro de atención médica, para perfeccionar el servicio de atención a sus pacientes.

1.1. Realidad Problemática

- **Trabajos Previos:**

Sanunga y Pérez (2019) realizaron un proyecto con el objetivo de implementar un sistema que permitiera el registro y la gestión del historial del paciente, la entrada del tratamiento y las citas de los pacientes para acudir a la clínica. Se implementó como herramienta de apoyo al Centro Odontológico Grupo Dental. Los creadores del proyecto concluyeron que la implementación de un sistema de gestión de registros médicos permitió una mejor gestión y organización de los registros médicos de los pacientes y fue respaldado por el Centro Dental de Dental Group.



Por otro lado, dirigiéndonos al sector público, tenemos como antecedentes del trabajo, a la institución EsSalud, la cual, puso en marcha en el año 2019, un nuevo Sistema de Gestión de Servicios de Salud que se denominó EsSI (Servicio de Salud Inteligente).

En el cual comprende la digitalización de los historiales médicos mediante una interfaz virtual, en la cual los médicos podrían acceder a la toda la información médica del paciente: consultas, exámenes médicos, recetas y entre otros datos. Con la finalidad de reducir el tiempo y mejorar la atención en centros de salud, beneficiando a más de 11 millones usuarios del seguro.

Y, además, significo un gran acontecimiento en el sistema de Salud del Perú, siendo un primer paso, a gran escala, en la era de la digitalización integral de los procesos y servicios de Salud a nivel nacional; y sirviendo como base para el desarrollo de futuros proyectos en busca de la masificación de los historiales médicos digitales, como el presente proyecto. Y como un proyecto de desarrollo sostenible amigable con el medio ambiente, se sabe que, con la plataforma digital, la institución economizará cerca de cuatro millones de soles al no tener que invertir en papel. (EsSalud, 2019)



- **Datos Estadísticos:**

Perú ocupa el puesto 71 en el nivel de digitalización de servicios en el Índice de Desarrollo de Gobierno Electrónico de las Naciones Unidas (ONU) 2020, lo que lo convierte en uno de los últimos en América Latina en su conjunto. En cuanto a la adopción de la historia clínica electrónica, según la Organización Panamericana de la Salud (OPS), solo el 52,6% de los países que la conforman cuentan con un sistema nacional de historia clínica electrónica, y solo el 26,3% cuenta con leyes que avalan su uso.

Según datos del Ministerio de Salud (Minsa) al 2021, el porcentaje de establecimientos de atención primaria con historia clínica electrónica en Lima no llega al 40%, e incluso al 10% en zonas como Cajamarca. Lo ideal es cambiar esta situación con la digitalización y modernización del sector salud, integrando en un solo lugar la información generada por los profesionales de la salud, los pacientes y los sistemas de información.

Además, Según la Ley N°30024 y las normas de la Organización Mundial de la Salud, la historia clínica digital debe tener al menos con estas cuatro características: ser única, es decir, cada peruano debe tener un historial clínico único; estar integrada, lo que significa que toda prestación o servicio de salud que se le brinde al paciente debe figurar en su historia clínica; debe ser acumulativa, todo evento pasado o futuro debe de quedar archivado en su historia; y debe ser portable, pudiendo acceder a la historia clínica desde cualquier parte del territorio nacional.

Y de las iniciativas o aplicativos desarrollados en el Perú, tanto en el sector público como privado, solo pocos cuentan con algunas de estas características, y hasta el año 2019 solo EsSalud cumplía en su totalidad con estos parámetros. (EsSalud, 2019) Según Medigest Consultants, la digitalización de las historias clínicas traerá importantes beneficios a la gestión de las instalaciones médicas, entre ellos, la tasa de retorno de la inversión en espacio físico al evitar grandes cantidades de registros físicos y reducir el tiempo dedicado a buscar registros médicos y posibles errores de archivo.

De lo expuesto en párrafos anteriores es evidente que el proyecto planteado puede contribuir a la solución del problema, el implementar una aplicación de escritorio que permita almacenar el historial médico del paciente, puede ayudar a tener un mejor seguimiento del estado del paciente, además, permite economizar en cuanto a gastos de oficina.

En el marco donde se rige una historia clínica, se halla la Norma Técnica de Salud (NTS 022, 2006), la cual brinda los métodos tradicionales y convencionales de archivamiento de una historia clínica, incluyendo los formatos que ésta debe contener, tales como: las fichas familiares, formatos de emergencia, formatos de consulta externa, etc.

El 22 de mayo del 2013, el Congreso aprobó la (Ley N°30024, 2014), la cual instituye el Registro Nacional de Historias Clínicas Electrónicas (RENHICE), cuyo fin es recolectar datos de toda persona que haya sido atendida en cualquier centro de salud del Perú (público o privado). Además, el usuario o su representante legal tienen permitido, además de los médicos, contar con acceso a los datos correspondientes a su respectiva historia clínica electrónica. (Ministerio, 2014)



En octubre del año 2015, se realizó la Jornada Internacional de Integración de Sistemas e Historia Clínica Electrónica, la cual tuvo como objetivo capacitar a todo el personal médico sobre las nuevas tecnologías de información (TI), las cuales se encuentran íntimamente relacionadas con sector Salud, debido a que estos serán las bases del desarrollo de la HCE y posteriormente para su implementación.

Durante el mes de mayo del 2014 “Lolimsa”, una empresa que desarrolla softwares para el sector salud, expresó que, hasta ese momento en el Perú, más del 89% de los datos de los pacientes no se encontraban en historias clínicas virtuales, y un pequeño porcentaje usaba algunos medios electrónicos y los demás centros médicos seguían empleando el archivado manual mediante el uso del papel. (Comercio, 2016)

En el mes de diciembre, el 17 del 2015 fue promulgado por el gobierno peruano el Decreto Supremo N° 039-2015-SA con respecto el reglamento de la (Ley N°30024, 2014), Ley que instituye el Registro Nacional de Historias Clínicas Electrónicas. Mediante este decreto, las clínicas y hospitales deben seguir labores para adecuarse a la nueva realidad.

EsSalud últimamente hace uso de historias clínicas electrónicas, alineadas a los objetivos de la Política Nacional de Gobierno Electrónico 2013-2017. Este sistema permite que los procesos en la atención sean más rápidos y eficaces hacia los usuarios. Osmeli Navarro, Gerente de Procesos Asistenciales en IBTgroup, Lima Perú (sociedades de operadoras de salud), define a este sistema como una herramienta que agiliza y mejora el trabajo. (Yrinna Benites, 2016)

1.2. Formulación del Problema

¿Cómo desarrollar una aplicación para la gestión de historiales médicos que mejore la calidad de servicio de un centro médico?

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar una aplicación para la gestión de historiales médicos de los pacientes para mejorar la calidad de servicio en los centros médicos.

1.3.2 Objetivos Específicos

1. Implementar la base de datos.
2. Elaborar el código ejecutor del programa y de la interfaz principal.
3. Realizar la conexión del programa con la base de datos.
4. Elaborar los iconos gráficos de la interfaz y su funcionalidad.

1.4 Justificación

- **Social:** Mejorar la calidad de servicio y atención a los pacientes.
- **Económica:** Disminuirá la inversión en los materiales de oficina, en este caso, el uso del papel bond.
- **Tecnología:** Implementar este tipo de sistema, habré las a futuro de mejorar las interfaces tecnológicas.

Capítulo 2

Marco Teórico

Python:

Python es un lenguaje de programación, el cual tiene mucha similitud con el lenguaje humano. Por otro lado, se define como un lenguaje de código abierto y lo mejor de todo, “gratis”, lo cual permite desarrollar software sin limitaciones.

El usar Python está muy extendido en dos áreas que son las más desarrolladas actualmente: el análisis de datos y el big data. La simplicidad de este lenguaje y su gran número de bibliotecas para procesar datos, la hacen ideal a la hora de hacer un análisis y gestión de una gran cantidad de datos.

Tkinter:

Es una librería que viene preinstalada en Python, la cual podemos usar en cualquier momento, utilizando el comando “import tkinter”. Además, es un conjunto de herramientas GUI de Tcl/Tk (Tcl: Tool Command Language), que proporciona una gran variedad de usos, en las cuales está el desarrollo de aplicaciones de escritorio, aplicaciones web, también se desempeña en la administración, entre otros. Tkinter no es solo la única librería para python especializada en la creación de interfaces gráficas, entre las más utilizadas están wxPython, PyQt y PyGtk.



SQLite:

SQLite es un software gratuito, el cual permite almacenar información en diferentes dispositivos, tales como una laptop, PC, Tablet o celular. Dicho software es compatible con múltiples plataformas, lo cual hace que la integración sea perfecta. SQLite posee su propio lenguaje de programación, además de contar con muchos componentes, bibliotecas y controladores que nos permiten interactuar con diferentes lenguajes de programación y plataformas.

Visual Studio Code:

Visual Studio Code es un editor de código de fuente propiedad de Microsoft. Es un software libre y además es compatible con muchos sistemas operativos, tales como: Windows, Linux y macOS. VS Code posee una gran cantidad de extensiones gratuitas, las cuales facilitan escribir y ejecutar códigos en cualquier lenguaje de programación.



Historia clínica:

El historial médico se puede obtener desde una variedad de perspectivas, desde una perspectiva gramatical, una perspectiva legal, un concepto médico o una perspectiva forense, que es un documento médico requerido por ley para registrar toda la información: Relaciones cuidador-paciente, todas las acciones y actividades de higiene médica realizadas sobre ellos, y todos los datos relacionados con su salud elaborados para facilitar su cuidado.

El objetivo principal de la historia clínica es recopilar datos sobre la salud del paciente y promover la atención médica. La razón por la que los médicos comienzan a obtener un historial médico y lo motivan a continuar durante mucho tiempo es la necesidad de servicios médicos por parte del paciente.

Usando las herramientas anteriores y el conocimiento del historial médico, se desarrollará una aplicación de escritorio que pueda recopilar datos médicos del paciente durante una visita al centro de salud. El historial médico del paciente es muy importante porque le da al paciente un antecedente.

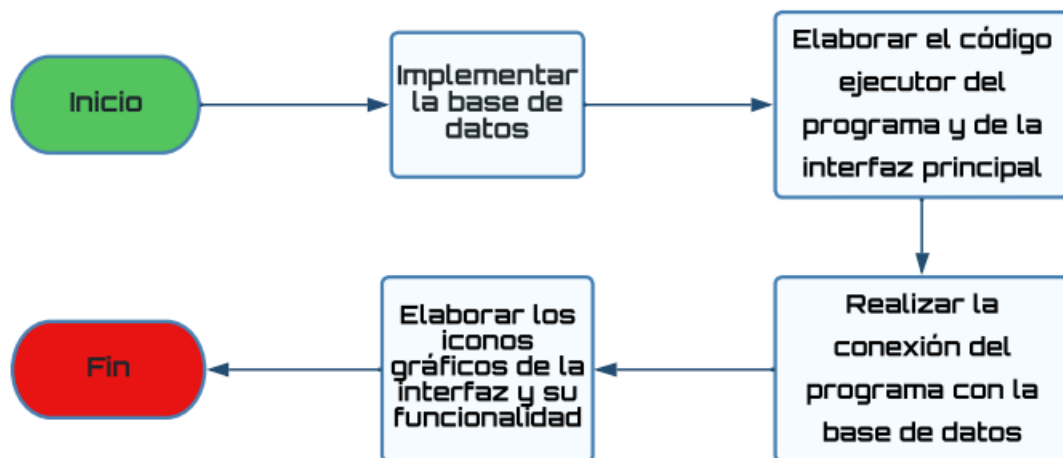
Capítulo 3

Materiales y Métodos

Para el desarrollo de este programa de escritorio, se hará uso del lenguaje de programación Python con la librería Tkinter, cuyo caso de estudio son los historiales médicos.

3.1. Descripción general de los procedimientos

- Descripción general de los objetivos:

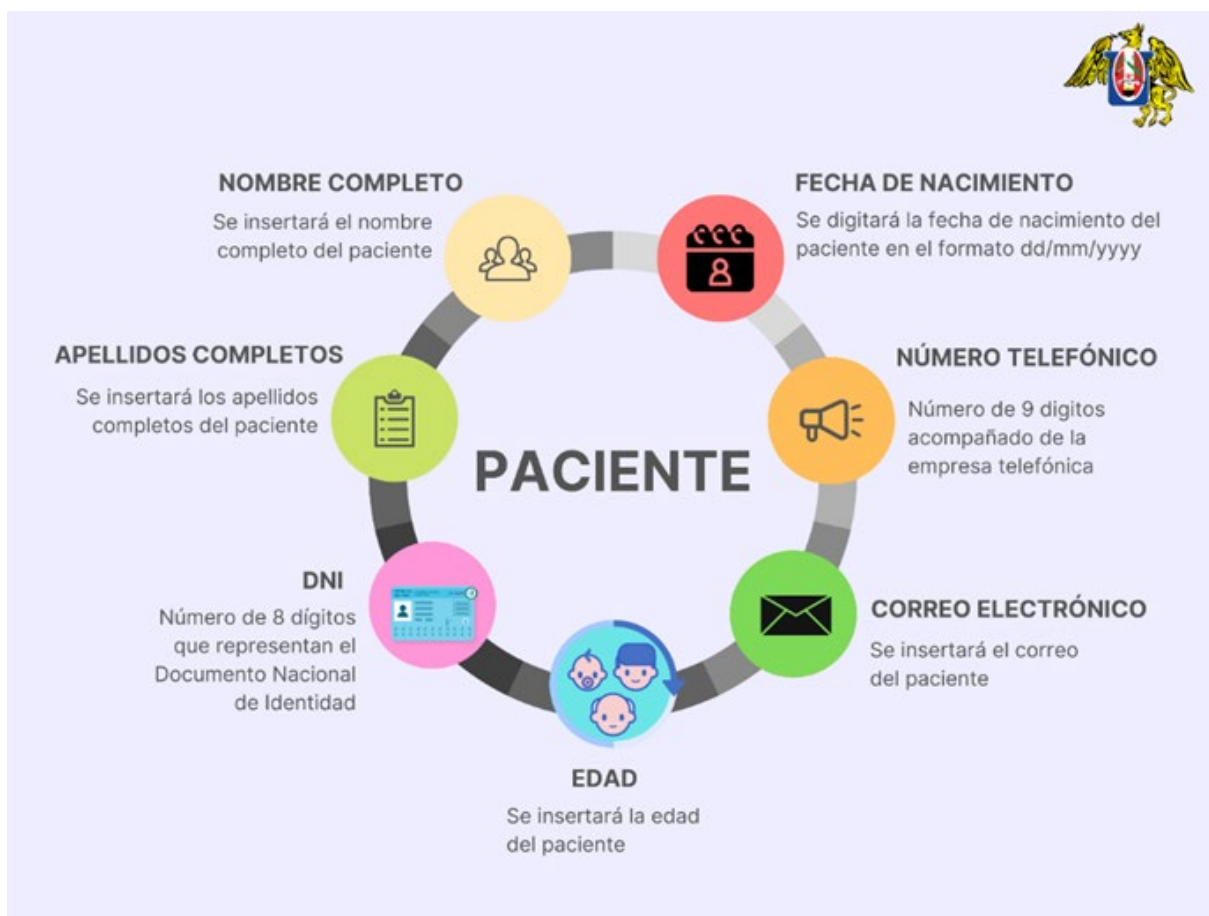


3.2. Desarrollo de los procedimientos

3.2.1. Implementación de la Base de Datos:

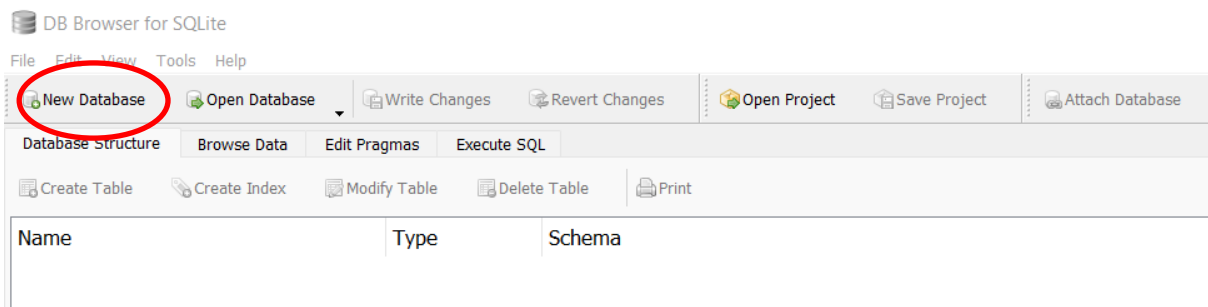
Para la implementación de la base de datos, se usó el software DB Browser for SQLite, en dicho software creamos las tablas que posteriormente se importaran al código general para hacerles la debida conexión con este mismo.

- Iniciamos creando la tabla que contendrá los datos del paciente, los datos que se tomaran del paciente están expresados en la siguiente imagen.

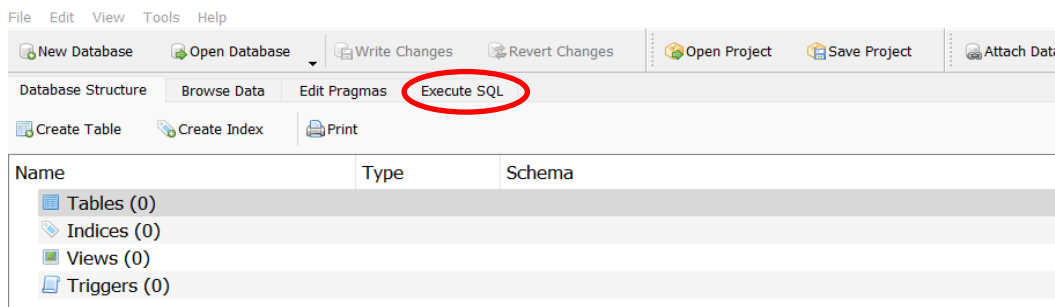




- El software DB Browser posee su propio lenguaje de programación, a continuación, se explicará paso a paso:
1. Abrimos la aplicación y damos click en donde dice: New Database.



2. Automáticamente se crea la sección de tablas, pero están vacías, por lo tanto, le damos click en “Execute SQL” para ingresar los códigos correspondientes y poder crear las tablas.



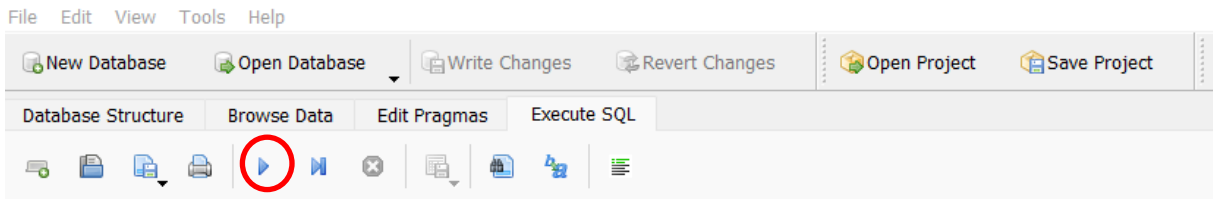
3. Se digitó el siguiente código para crear la tabla que contenga los datos del paciente:

```
SQL 1 x
1 CREATE TABLE DatosPaciente(
2   idPersona INTEGER PRIMARY KEY AUTOINCREMENT,
3   NombreCompleto VARCHAR(20),
4   ApellidosCompleto VARCHAR(20),
5   DNI INTEGER UNIQUE,
6   FechaNacimiento VARCHAR(20),
7   Edad INTEGER,
8   NumeroTelefonico INTEGER,
9   CorreoElectronico VARCHAR(20),
0   activo INTEGER
1 );
```

Donde:

- CREATE TABLE: Este comando nos permite crear la tabla.
- INTEGER: Define un dato como entero.
- INTEGER UNIQUE: Similar al comando INTEGER, solo que al tener el UNIQUE, garantiza que no se duplique en las columnas o secciones establecidas.
- VARCHAR: Define un dato como cadena o texto.
- PRIMARY KEY: Define un dato como único y además de tomar valores que no sean nulos, para poder identificar de forma exclusiva cada fila de las tablas.
- AUTOINCREMENT: Este comando es para generar un número único cuando se inserta algún nuevo registro en la tabla.

4. Posteriormente al haber colocado el código, compilaremos el código para que se ejecute y cree la tabla si todo está correcto:



5. De manera análoga se repitió los pasos anteriores para crear la tabla de Historia Médica:

```
CREATE TABLE HistoriaMedica (  
    idHistorial INTEGER PRIMARY KEY AUTOINCREMENT,  
    idPersona INTEGER,  
    FechaHistoria VARCHAR(16),  
    MotivoDeLaVisita VARCHAR(20),  
    Operacion VARCHAR(20),  
    Tratamiento VARCHAR(40),  
    DetalleAdicional VARCHAR(30),  
    FOREIGN KEY (idPersona) REFERENCES DatosPaciente(idPersona)  
);
```

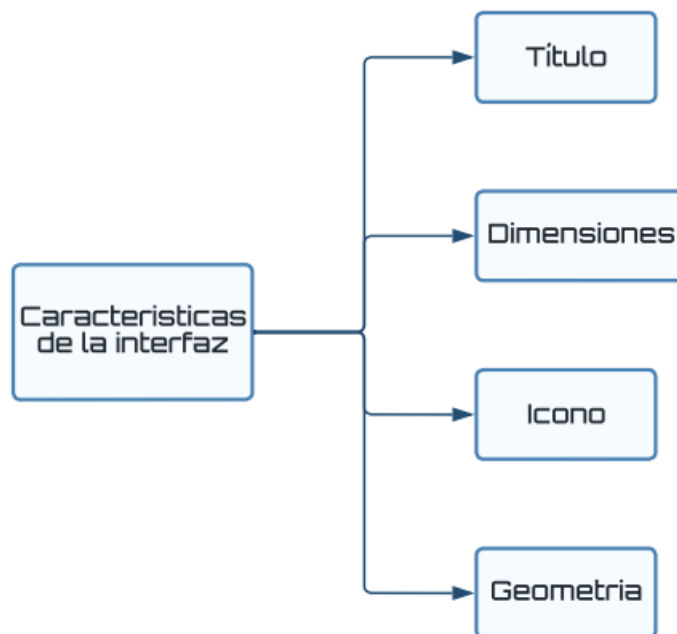
Dos nuevos comandos aparecen en este caso:

- FOREIGN KEY: Sirve para señalar la clave primaria (PRIMARY KEY) de otra tabla.
- REFERENCES: Esto acompaña a FOREIGN KEY para señalar a que tabla está haciendo referencia.

3.2.2. Elaboración del código ejecutor del programa y de la interfaz principal:

Creamos la interfaz principal en el archivo “PROGRAMA.py” importando el módulo tkinter y definimos la función principal “main()” que será la ejecutora del programa general. Dentro de la función especificamos los valores para la interfaz como el título, tamaño, el color, entre otros. (Código en imagen: [ANEXO 1](#))

DIAGRAMA:

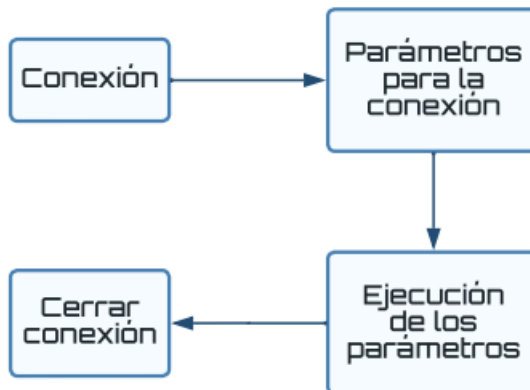


3.2.3. Realización de la conexión del programa con la base de datos:

Comenzamos estableciendo la conexión con la base de datos que anteriormente se implementó, para lo cual se creó un archivo Python con el nombre de conexion.py. Importamos el módulo sqlite3, el cual nos permitirá establecer la conexión con la base de datos anteriormente implementada. Después colocamos la clase ConexionDB, en la cual se definirán, primero mediante el constructor de objetos `__init__`, sus atributos utilizando self. En el renglón 5 se establece la ruta en donde se encuentra el archivo .db (base de datos), en el renglón 6 se usó una función del módulo sqlite3 para hacer referencia al renglón 5 y así poder conectarnos a la base de datos. En el renglón 7 se utilizó el método cursor el cual nos permite modificar los datos.

Luego se definirá el método `cerrarConexion(self)` en el renglón 9, en el renglón 10 hacemos uso del método `commit` que nos permitirá subir los datos insertados en las entradas, además asegura que los cambios realizados sean coherentes en la base de datos. Por último, en el renglón 11 usamos el método `close` para cerrar la conexión luego de insertar los datos. (Código en imagen: [ANEXO 2](#))

DIAGRAMA:



Archivo “pacienteDao.py”

Luego de haber creado el archivo `conexión.py`, se creará otro archivo Python con el nombre `PacienteDao.py`, en el cual se establecerán funciones que permitan vincular a la base de datos, a la GUI y la clase `DatosPaciente`.

Una vez creado el archivo, se importará la clase `ConexionDB` del archivo `conexion.py` y el módulo `messagebox` de la librería `tkinter`, el cual servirá para mostrar ventanas emergentes. (Código en imagen: [ANEXO 3](#))

Ahora, se colocará las funciones que vinculen a la base de datos creada en DB Browser for SQLite:

1. Función para guardar los datos del paciente:

Definimos la función `guardarDatoPaciente(persona)`, la cual guardará los datos en la clase `DatosPaciente`, en el renglón 10 establecemos la conexión con la base de datos (esto se colocará en cada función que vincule a `sqlite`). En el renglón 11 establecemos la variable `sql` que permite almacenar texto, en este caso los datos del paciente (nombre completo, apellidos, etc).

En el renglón 14 usamos el método `execute` para ejecutar el `sql` (los datos que se agreguen) y en el renglón 15 se cierra la conexión. Por último, en el renglón 18 se aplica el módulo

`messagebox` con el método `showinfo` para mostrar el título y mensaje en la interfaz a la hora de guardar los datos del paciente. (Código en imagen: [ANEXO 4](#)).

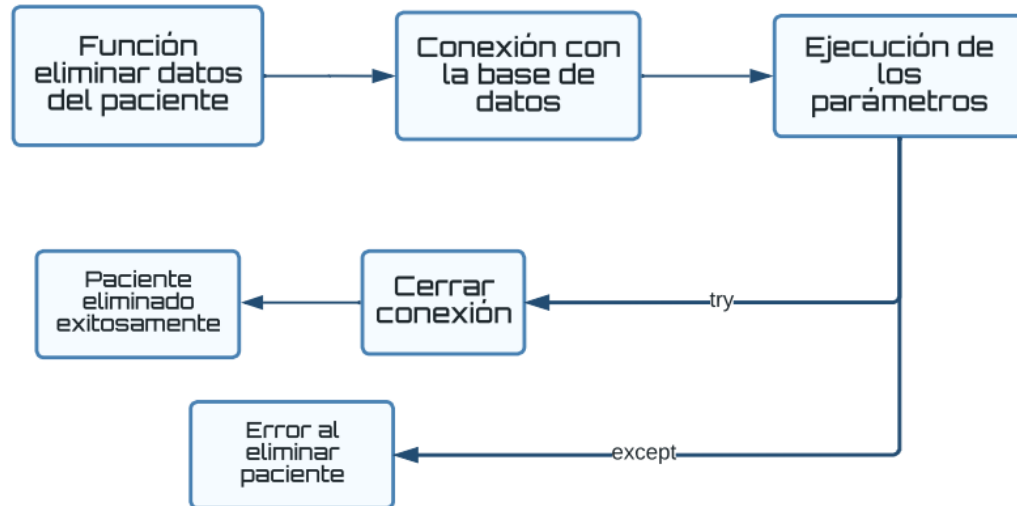
DIAGRAMA:



2. Función para eliminar los datos del paciente:

Definimos la función `eliminarPaciente(idPersona)` (`idPersona` será el parámetro), el cual tiene la finalidad de volver inactivo a un paciente y consecuentemente no mostrarlo en la tabla generada con `Treewiew` en la interfaz. En el renglón 23 establecemos la conexión con la base de datos y el renglón 24 establecemos la variable `sql` la cual tiene almacenada el `idPersona` (este número se genera automáticamente al registrar un paciente y es único). Se hizo uso de `try` y `except`, en caso que el `try` presente errores, se ejecutará el `except`. Del renglón 26 al 30 es similar a lo hecho en del renglón 14 al 18. En el `except` se muestra la función `showwarning`, la cual es muy similar al `showinfo`, solo que muestra más alerta en el mensaje. (Código en imagen: [ANEXO 5](#)).

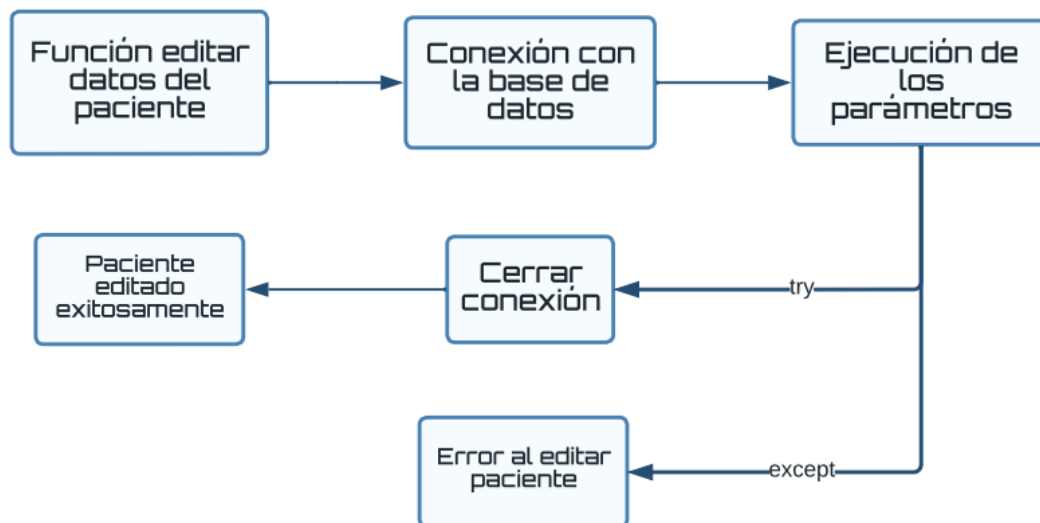
DIAGRAMA:



3. Función para editar los datos del paciente:

Definimos la función `editarDatoPaciente(persona, idPersona)` del objeto `persona` con el parámetro `idPersona`, esta función permitirá sobrescribir los datos en cada objeto `persona` de la clase `DatosPaciente`, en el renglón 39 establecemos la conexión como anteriormente se ha hecho, en el renglón 40 se establece la variable `sql`, la cual contendrá los nuevos datos para luego sobrescribirlos. Por último, del renglón 44 al 53 se hizo lo mismo que en la función anterior, solamente se cambió el título y mensaje que se mostrará. (Código en imagen: [ANEXO 6](#))

DIAGRAMA:



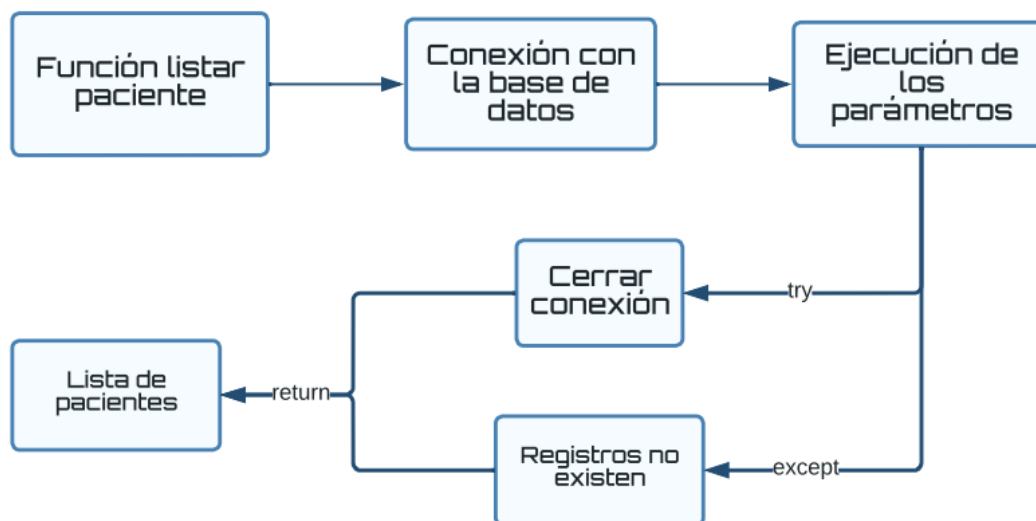
4. Funciones para mandar los datos insertados a la tabla en la GUI:

Antes de crear las funciones, debemos crear la clase DatosPaciente con el constructor class. El cual contendrá como atributos los datos especificados en la base de datos. Por último, con el método `__str__(self)` mostraremos los objetos. (Código en imagen: [ANEXO 7](#))

- **Función Listar:**

Definimos la función `listar()`, la cual nos va a permitir mandar los datos a una lista vacía llamada “ListaDatosPaciente”, en el renglón 58 establecemos la conexión. En el renglón 61 se establece la variable `sql` para seleccionar los pacientes que se encuentren activos (`activo = 1`). Del renglón 63 al 71 se repite casi mismo que en las anteriores funciones, salvo en el renglón 65 en donde se añade el método `fetchall`, el permitirá añadir los datos a la lista, además también se cambió el título y mensaje mostrado en el `except`, al final todo retorna a la `ListaDatosPaciente`. (Código en imagen: [ANEXO 8](#))

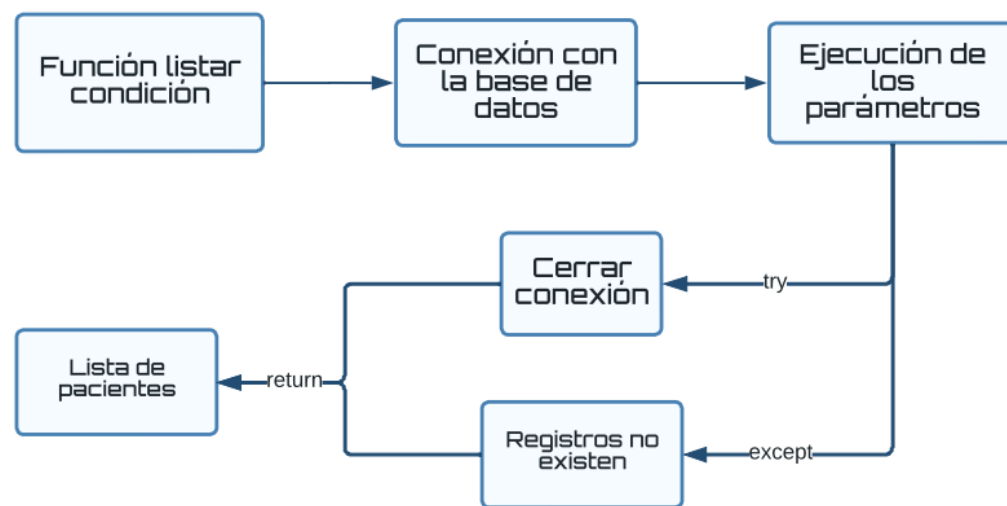
DIAGRAMA:



- **Función listarCondicion:**

Definimos la función listarCondicion(where), esta función mandará los datos de la lista a la tabla que aparece en la GUI, en el renglón 74 establecemos la conexión, en el renglón 75 colocamos la variable listaDatosPaciente, la cual representa a la lista, después en el renglón 76 establecemos la variable sql, la cual seleccionará los datos de paciente activos. Por último, del renglón 78 al 86, es similar a la función anterior. (Código en imagen: [ANEXO 9](#))

DIAGRAMA:



Archivo “HistorialDao.py”

Una vez hecho lo anterior, se creará otro archivo Python con el nombre HistorialDao.py, en el cual se establecerán funciones que permitan vincular a la base de datos, a la GUI y la clase HistoriaMedica.

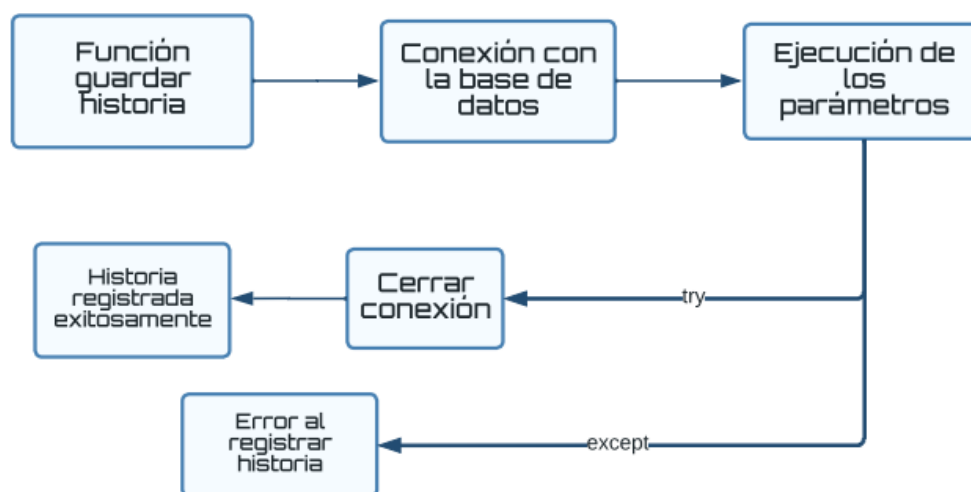
Una vez creado el archivo, se importará la clase ConexionDB del archivo conexion.py y el módulo messagebox de la librería tkinter, el cual servirá para mostrar ventanas emergentes. (Código en imagen: [ANEXO 10](#))

Ahora, se colocará las funciones que vinculen a la base de datos creada en DB Browser for SQLite:

5. Función para guardar la historia médica del paciente:

Definimos la función guardarHistoria(), la cual guardará los datos en la clase HistoriaMedica, en el renglón 30 establecemos la conexión con la base de datos (esto se colocará en cada función que vincule a sqlite). En el renglón 31 establecemos la variable sql que permite almacenar texto, en este caso la historia médica (tratamiento, operación, etc). En el renglón 34 usamos el método execute para ejecutar el sql (los datos que se agreguen) y en el renglón 35 se cierra la conexión. Por último, en el renglón 38 se aplica el módulo messagebox con el método showinfo para mostrar el título y mensaje en la interfaz a la hora de guardar los datos del paciente; en caso que se presente un error se ejecutará el except del renglón 39. (Código en imagen: [ANEXO 11](#)).

DIAGRAMA:

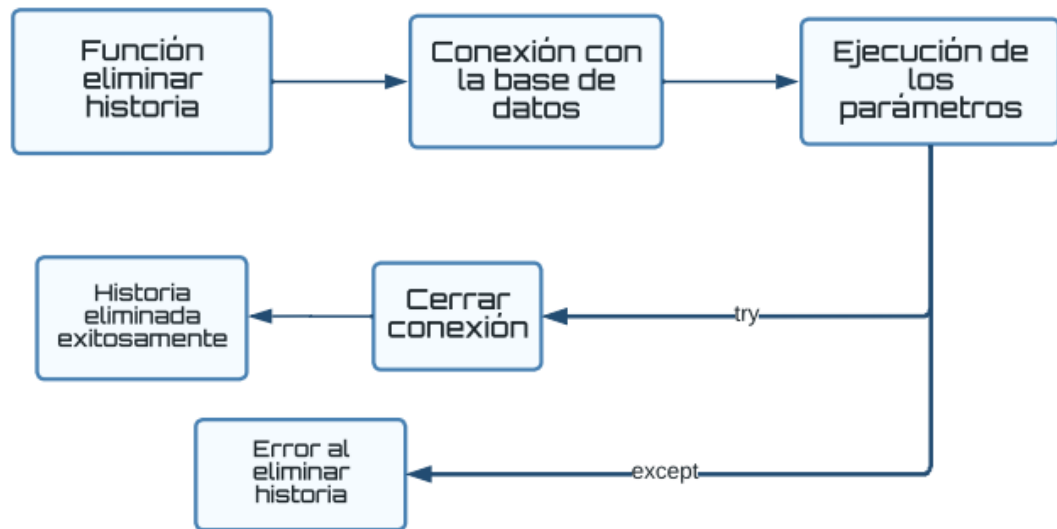


6. Función para eliminar la historia médica del paciente:

Definimos la función eliminarHistoria(idHistorial) (idHistorial será el parámetro), el cual tiene la finalidad de volver inactiva la historia médica de un paciente y consecuentemente no mostrarlo en la tabla generada con Treeview en la interfaz. En el renglón 47 establecemos la conexión con la base de datos y el renglón 48 establecemos la variable sql la cual tiene almacenada el idHistoria (este número se genera automáticamente al registrar la historia y es único).

Se hizo uso de try y except, en caso que el try presente errores, se ejecutará el except. Del renglón 51 al 59 es similar a lo hecho en del renglón 34 al 42. En el except se muestra la función showwarning, la cual es muy similar al showinfo, solo que muestra más alerta en el mensaje. (Código en imagen: [ANEXO 12](#)).

DIAGRAMA:



7. Función para editar historia médica del paciente:

Definimos la función `editarHistoria()`, esta función permitirá sobrescribir los datos en cada objeto persona de la clase `HistoriaMedica`, en el renglón 64 establecemos la conexión como anteriormente se ha hecho, en el renglón 65 se establece la variable `sql`, la cual contendrá los nuevos datos para luego sobrescribirlos. Por último, del renglón 66 al 75 se hizo lo mismo que en la función anterior, solamente se cambió el título y mensaje que se mostrará. (Código en imagen: [ANEXO 13](#))

DIAGRAMA:

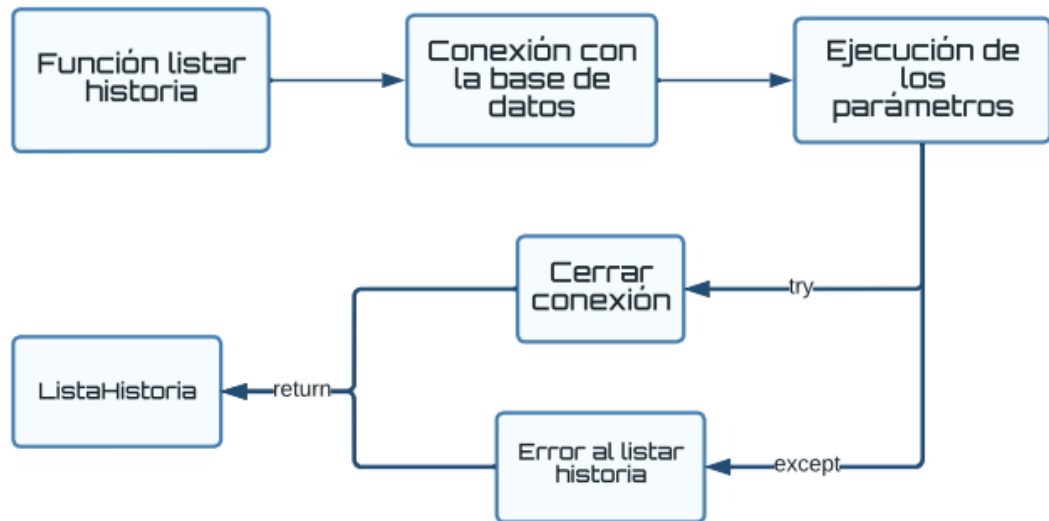


8. Funciones para mandar los datos insertados a la tabla en la GUI:

- **Función listarHistoria:**

Definimos la función listarHistoria(idPersona), la cual nos va a permitir mandar los datos a una lista vacía llamada “listaHistoria”, en el renglón 80 establecemos la conexión. En el renglón 83 se establece la variable sql para seleccionar a los que se encuentren activos (activo = 1). Del renglón 86 al 93 se repite casi lo mismo que en las anteriores funciones, salvo en el renglón 88 en donde se añade el método fetchall, el permitirá añadir los datos a la lista, además también se cambió el título y mensaje mostrado en el except, al final todo retorna a la listaHistoria. (Código en imagen: [ANEXO 14](#))

DIAGRAMA:



3.2.4. Elaboración de los iconos gráficos de la interfaz y su funcionalidad:

- Creamos la GUI, dentro del archivo “INTERFAZ” e importamos los módulos tkinter, tkcalendar, datetime, PIL, fpdf y las conexiones creadas en PacienteDao e HistorialDao. (Código en imagen: [ANEXO 15](#))
- Creamos la clase “Frame(tk.Frame)”, dentro de la cual definiremos el constructor “__init__(self, aplicación)” y super para llevar a cabo herencias múltiples hacia la superclase aplicación. (Código en imagen: [ANEXO 16](#))
- Definimos el método “campoPaciente(self)”, donde irán los ítems de la interfaz principal para ingresar los datos del paciente, llamados “labels”. A los cuales les colocaremos el texto del respectivo ítem con “text=” y lo personalizaremos en el apartado “.config”: su tipo de letra y tamaño con “Font=”, su color con “background=” y su ubicación dentro del widgets con “anchor=”. Además de la ubicación del widget en general, en el apartado “.grid” con los comandos “column=” para especificar columna y “row=” para especificar fila. (Código en imagen: [ANEXO 17](#))
- Creamos los espacios de inserción de los datos pedidos por los labels, los cuales son llamados entrys. Los definimos mediante la función StringVar() y les agregamos tipo de letra, tamaño y ubicación de la misma forma que los labels. Además, de algunas opciones de edición como el comando “columnspan=” que define las columnas que ocupara el entry, en este caso, 2. (Código en imagen: [ANEXO 18](#))
- Creamos los botones, necesarios para el campo paciente como el “NUEVO”, “GUARDAR” y “CANCELAR”. Y también les agregamos los valores de tamaño, tipo de letra, color y ubicación como con los anteriores elementos. (Código en imagen: [ANEXO 19](#))
- Creamos la función de buscador. Empezando por la creación de su respectivo label, entry y button. Y también les agregamos los valores de tamaño, tipo de letra, color y ubicación como con los anteriores elementos. (Código en imagen: [ANEXO 20](#))



- Luego, definimos el método “buscarCondicion(self)” donde le asignaremos el funcionamiento de búsqueda al apartado compuesto por el “lblBuscarDni”, “entryBuscarDni” y “btnBuscarCondicion”, la cual trabajara con un condicional que restringe la búsqueda a los pacientes mediante su DNI. (Código en imagen: [ANEXO 21](#))
- Definimos el método “deshabilitar()”, la cual mantendrá los entrys bloqueados y en blanco con la finalidad de que no se pueda escribir nada hasta que se seleccione el botón nuevo. Al cual, más adelante, le agregaremos una función que contrarreste este bloqueo. (Código en imagen: [ANEXO 22](#))
- Definimos el método “habilitar()”, la cual habilita en blanco los entrys para que se puedan colocar los datos. Esta función es inversa a “deshabilitar()” y se le agregara al botón NUEVO, para habilitar el ingreso de los datos del paciente con este botón, mediante el comando “command=self.habilitar”. (Código en imagen: [ANEXO 23](#))
- Definimos el método “tablaPaciente(self, where=”n”):” para insertar la tabla de datos de los pacientes desde el Sqlite a la interfaz principal. Acá, hacemos el llamado a la función “listar()” y “listarCondicion()” desde la Dao, para establecer el condicional que permita el listado de datos de los pacientes activos, caso contrario se muestren los pacientes con la condición requerida.

Creamos la tabla con el comando “ttk.Treeview”, definiendo las columnas que tendrá, su ubicación, su tipo de letra y su color de filas, de manera similar a como se configuraron estos parámetros para elementos anteriores. Como comando adicional usamos el “ttk.Scrollbar” el cual crea el parámetro para desplazarse (subir y bajar) en la interfaz de la tabla. (Código en imagen: [ANEXO 24](#))
- Luego, definimos los títulos de cada columna mediante el método “.heading()” y algunas configuraciones de detalle de estas mismas como el anchor y el width con el método “.column”. Insertamos los datos desde el Sqlite hacia la tabla, asignándole a cada valor una columna en especial e iterando en filas mediante el “for”. (Código en imagen: [ANEXO 25](#))

- Creamos los botones que trabajaran con la tabla de datos de los pacientes: “Editar Paciente” “Eliminar Paciente” y “Historial Paciente”. Los cuales fueron configurados de forma similar a los botones antes vistos. (Código en imagen: [ANEXO 26](#))
- Creamos la función “MostrarCalendario(self)”, la cual ejecutara el calendario como una ventana secundaria. Es así, que usamos el método “TopLevel()” para esta ventana y la configuramos de acuerdo a como se hizo con la interfaz principal, aplicando métodos como él “.title”, “.geometry”, “.resizable”, entre otros. Además, se hace uso del módulo importado al inicio, para hacer las correctas configuraciones del calendario, sea fecha de inicio, color, tipo de letra, etc. (Código en imagen: [ANEXO 27](#))
- Definimos la función “EnviarFecha(self)”, que se encargara de enviar la fecha seleccionada en el calendario hacia el apartado “Fecha de nacimiento”. (Código en imagen: [ANEXO 28](#))
- Definimos la función “CalcularEdad(self)”, la cual sera responsable de hallar la edad del paciente, haciendo uso de la fecha insertada del calendario. Además, colocara automáticamente esta edad en el apartado “Edad”. (Código en imagen: [ANEXO 29](#))
- Definimos el método “guardarPaciente(self):” que trabajara con el botón “GUARDAR” y conectara los datos ingresados en los entrys del campo paciente, con la tabla en Sqlite, agregando cada paciente a la tabla. Todo esto, con la ayuda de la función “guardarDatoPaciente(persona)” llamada desde el PacienteDao. Además, crea una funcionalidad para el caso de la edición de los datos de un paciente, mediante el condicional, donde si el idPersona es nuevo, se guarda los datos como nuevo, en caso contrario se editarán los datos del idPersona puesto, evaluándolo con la función “editarDatoPaciente()” también llamada desde el PacienteDao. (Código en imagen: [ANEXO 30](#))

- Definimos el método “editarPaciente(self):” la cual lleva los datos de la tabla hacia los entrys del campo paciente. Esto consiste solo en traslado de datos, una vez los datos se ubiquen en los entrys, se ejecutará el método “guardarPaciente()” en la cual también se definió la función editar según sea el caso. Luego, el método se agregará al botón “Editar Paciente” para su posterior ejecución mediante este botón. (Código en imagen: [ANEXO 31](#))
- Definimos el método “eliminarDatoPaciente(self):” la cual llamara a la función “eliminarPaciente(idPersona)” desde la Dao, para usarlo directamente como cambio de activo en la función. Luego, la función se agregará al botón “Eliminar Paciente” para su posterior ejecución mediante este botón. (Código en imagen: [ANEXO 32](#))

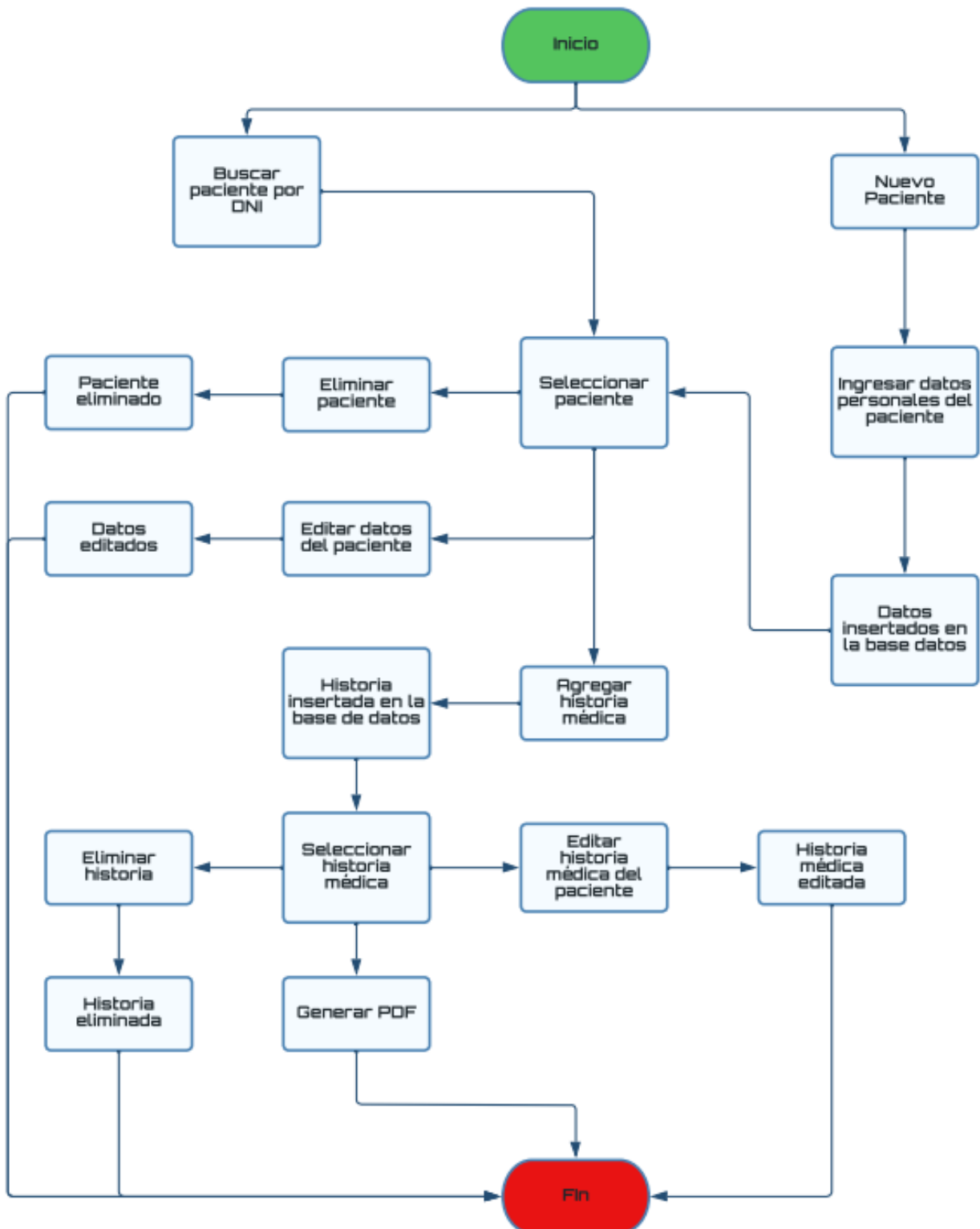
TopLevel de la historia medica

- Definimos la función “tablaHistoria(self)”, la cual será la responsable de mostrar los historiales de los pacientes, en caso se seleccione el botón al cual se le asignará esta función. Dentro de la forma try except definimos las cabezas de columnas de la tabla, sus ubicaciones y sus tamaños. (Código en imagen: [ANEXO 33](#))
- Definimos la función” historiaMedica(self)”, la cual generará la venta emergente donde se colocará la tabla de la historia con la función antes mencionada. Esta ventana es creada mediante el código “TopLevel()” y se configura sus detalles como título, tamaño y color de la misma que se hizo con la interfaz principal. Dentro de esta función definimos los botones que trabajaran con las historias médicas, como son “Agregar historia”, “Editar historia”, “Eliminar historia” y “Generar PDF”, a los cuales le asignaremos sus respectivas funciones que definiremos más adelante. (Código en imagen: [ANEXO 34](#))
- Creamos la función “crearPDF(self)”, la cual será la responsable de crear el historial de cada paciente en formato PDF. Esta función se le asigna al botón “Generar PDF”, y consiste en llevar y ordenar los datos de una de las historias seleccionadas en la tabla, en una hoja para su fácil entendimiento y uso.

Dentro del código creamos el PDF en su totalidad, asignando el texto respectivo, las imágenes y las firmas de los involucrados de la forma que se aprecia en el código. (Código en imagen: [ANEXO 35](#))

- Codificamos la función “topAgregarHistoria(self)”, que será la encargada de abrir una nueva ventana donde se podrán agregar los datos de una nueva historia. Las configuraciones de esta nueva ventana se realizan de la misma forma como ventanas anteriores. Dentro de esta ventana definimos dos secciones(frame’s) donde se ubicarán los respectivos labels, entrys y buttons. Siendo la primera parte donde estarán ubicados los widgets referentes a los campos de datos y la segunda sección los widgets de la fecha y hora; y los botones “Agregar” y “Salir”. Estos se configuran de la misma forma como ya se vio en anteriores secciones. (Código en imagen: [ANEXO 36](#))
- Definimos la función “agregaHistorialMedico(self)”, la cual agregara el historial médico nuevo a la tabla de los historiales del paciente. Esta función trabajara con el botón “Agregar”. (Código en imagen: [ANEXO 37](#))
- Definimos la función “eliminarHistorialMedico(self)”, que se encargara de eliminar el historial seleccionado. Luego esta función es agregada al botón “Eliminar Historia”. (Código en imagen: [ANEXO 38](#))
- Creamos una nueva ventana mediante la función “topEditarHistorialMedico(self)”. La cual será parecida a la ventana de Agregar historia, tanto en labels, entrys y buttons; con la diferencia de los botones “Agregar” y “Editar historia”; y de que esta ventana se abrirá con los datos insertados de un historial existente para su posterior edición. (Código en imagen: [ANEXO 39](#))
- Creamos la función “historiaMedicaEditar(self)” para guardar la edición de un historial. Esta función trabajará con el botón “Editar Historia” de la ventana antes vista. (Código en imagen: [ANEXO 40](#))
- Definimos la función “error()” para diferentes casos donde posiblemente pueda ocurrir errores que convengan en la paralización del programa y la ejecutamos en el código principal mediante un “try except”. (Código en imagen: [ANEXO 41](#))

DIAGRAMA GENERAL:



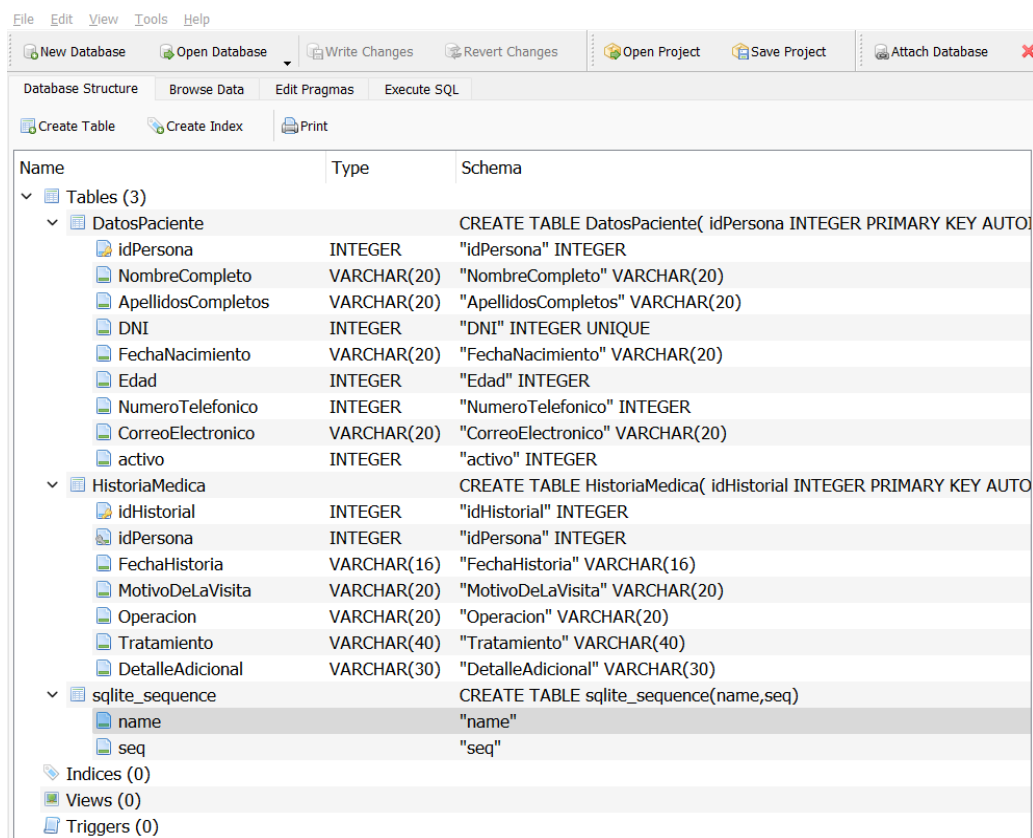
Capítulo 4

Resultados

En este capítulo, se discutirán los resultados obtenidos del proyecto de investigación.

4.1. Implementación de la Base de Datos

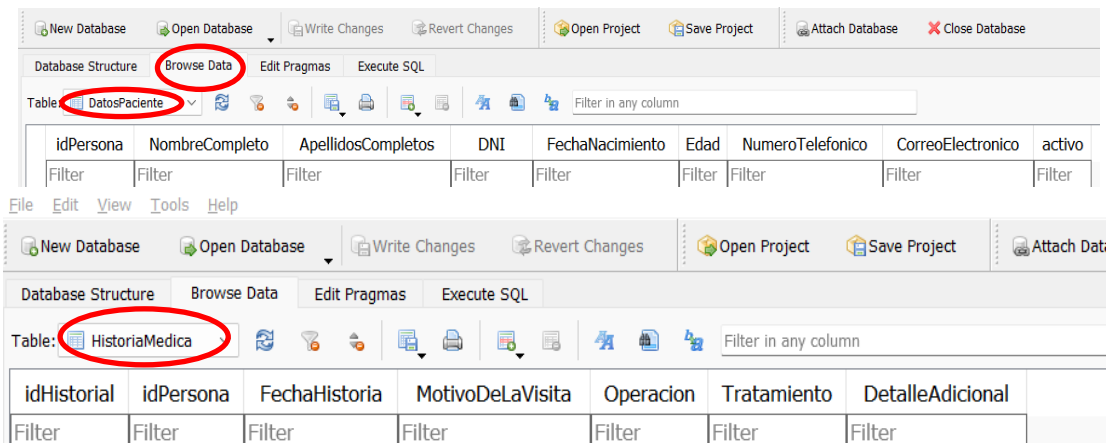
Una vez ejecutado los códigos para la creación de tablas, nos queda lo siguiente:



Name	Type	Schema
Tables (3)		
DatosPaciente		CREATE TABLE DatosPaciente(idPersona INTEGER PRIMARY KEY AUTO
idPersona	INTEGER	"idPersona" INTEGER
NombreCompleto	VARCHAR(20)	"NombreCompleto" VARCHAR(20)
ApellidosCompleto	VARCHAR(20)	"ApellidosCompleto" VARCHAR(20)
DNI	INTEGER	"DNI" INTEGER UNIQUE
FechaNacimiento	VARCHAR(20)	"FechaNacimiento" VARCHAR(20)
Edad	INTEGER	"Edad" INTEGER
NumeroTelefonico	INTEGER	"NumeroTelefonico" INTEGER
CorreoElectronico	VARCHAR(20)	"CorreoElectronico" VARCHAR(20)
activo	INTEGER	"activo" INTEGER
HistoriaMedica		CREATE TABLE HistoriaMedica(idHistorial INTEGER PRIMARY KEY AUTO
idHistorial	INTEGER	"idHistorial" INTEGER
idPersona	INTEGER	"idPersona" INTEGER
FechaHistoria	VARCHAR(16)	"FechaHistoria" VARCHAR(16)
MotivoDeLaVisita	VARCHAR(20)	"MotivoDeLaVisita" VARCHAR(20)
Operacion	VARCHAR(20)	"Operacion" VARCHAR(20)
Tratamiento	VARCHAR(40)	"Tratamiento" VARCHAR(40)
DetalleAdicional	VARCHAR(30)	"DetalleAdicional" VARCHAR(30)
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
name		"name"
seq		"seq"
Indices (0)		
Views (0)		
Triggers (0)		

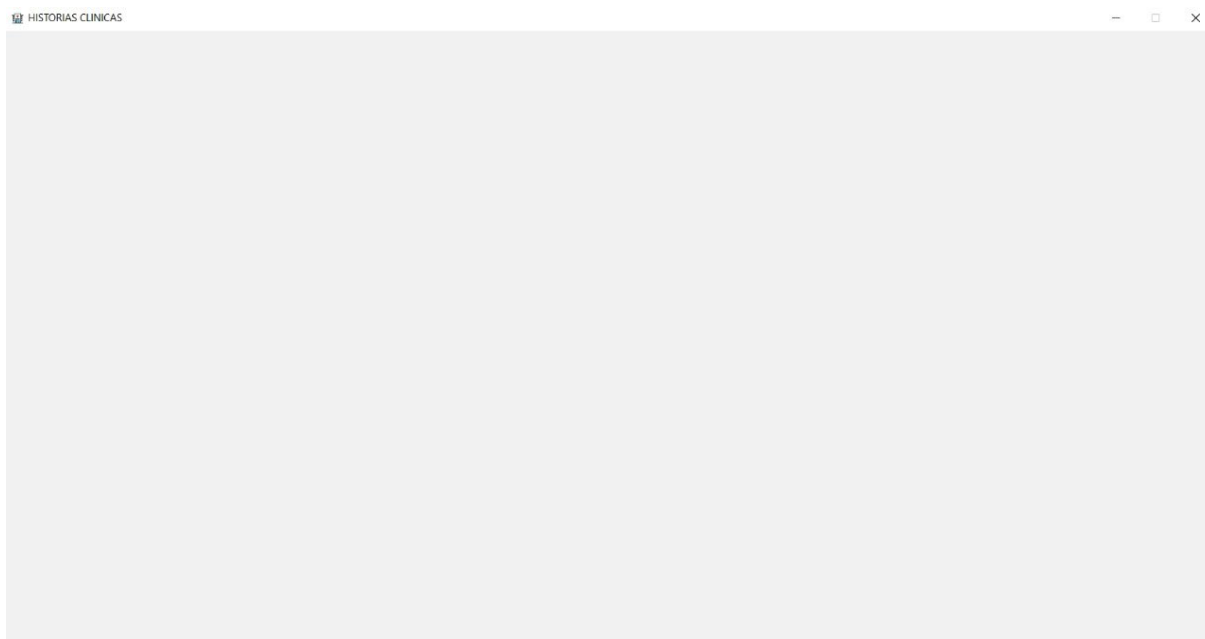
La tabla “sqlite_sequence” se genera de manera automática al ejecutar los códigos de creación de tablas.

Dando click en “Browse Data” se visualiza las columnas de las tablas:



4.2. Elaboración del código ejecutor del programa y de la interfaz principal

Como resultado del objetivo 2 obtenemos la ejecución de la interfaz del programa. Esta se ira llenando a medida que se avance en los objetivos, de la misma forma que el código ejecutor del programa, al cual mientras se avanzaba se le fue agregando las funciones creadas en los archivos de conexión, Dao y GUI.



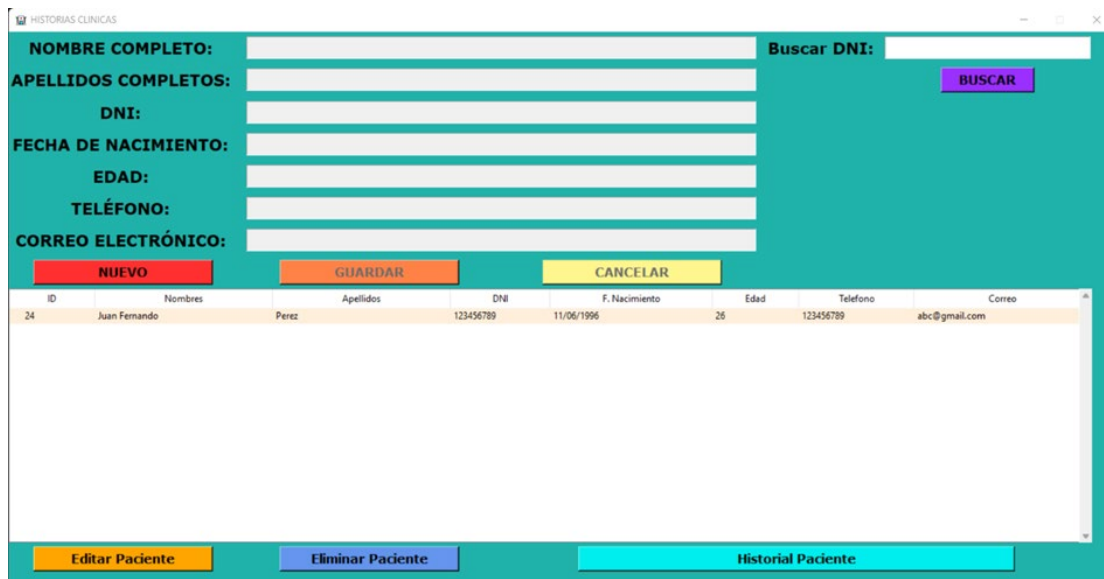
4.3. Realización de la conexión del programa con la base de datos

Esta conexión es fundamental para trabajar con Python y SQLite, ya que establece una relación entre nuestro código y la base de datos, permitiendo el flujo de información entre ambos, haciendo el llamado de esta misma para algunas funciones de los botones que se verán más adelante, como la de guardar paciente.

```
CONEXION > conexion.py > ...
1  import sqlite3
2
3  class ConexionDB:
4      def __init__(self):
5          self.baseDatos = "BaseDeDatos/DbHistorial.db"
6          self.conexion = sqlite3.connect(self.baseDatos)
7          self.cursor = self.conexion.cursor()
8
9      def cerrarConexion(self):
10         self.conexion.commit()
11         self.conexion.close()
```

4.4. Elaboración de los iconos gráficos de la interfaz y su funcionalidad

Se le agrego los widgets a la interfaz principal como los “labels” “entrys” y “buttons”. Además, también se le agrego la tabla de pacientes.



ID	Nombres	Apellidos	DNI	F. Nacimiento	Edad	Telefono	Correo
24	Juan Fernando	Perez	123456789	11/06/1996	26	123456789	abc@gmail.com

Se le agrego la interfaz de la ventana secundaria correspondiente a los historiales de los pacientes.

HISTORIAL MEDICO						
ID	Paciente	Fecha y Hora	Motivo de la visita	Operacion	Tratamiento	Detalle adicional
15	Juan Perez Garcia	22/08/2022, 09:22 PM	Molestia en zona superior derecha di	Resonancia magnetica	Medicacion periodica con paracetamol	
9	Juan Perez Garcia	21/08/2022, 10:34 AM	Consulta	Transplante de corazón		Descanso por 1 año y no hacer actividades que requ
2	Juan Perez Garcia	22/08/2022, 09:24 PM	Consulta	Circunsición	Descanso por 3 meses	

Agregar Historia
Editar Historia
Eliminar Historia
Generar PDF

Así como también las ventanas terciarias productos de los siguientes botones:

- “Agregar Historia”

AGREGAR HISTORIA

Motivo de la visita

Operación

Tratamiento

Detalle adicional

Fecha y Hora

29/08/2022, 06:45 PM

Agregar

Salir

- “Editar Historia”

EDITAR HISTORIA MEDICA

Motivo de la visita

Malestar a la hora de evacuar las eses

Operación

Ninguna

Tratamiento

Dieta saludable y purgantes

Detalle adicional

control medico semanal por un mes

Fecha y Hora

29/08/2022, 06:47 PM

Editar Historia

Salir

En el [ANEXO 42](#) podrá encontrar los links tanto del repositorio como de la carpeta drive.

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

- El trabajar con un gestor de base de datos como el SQLite, nos facilita mucho la realización de este tipo de programas, evitándonos la generación de un código propio para la gestión de los datos.
- En la creación de programas es necesario que el código principal sea una síntesis de todos los componentes del código para hacer más sencilla su legibilidad. Es decir, que debe ser un ejecutor de funciones y clases llamadas de otros archivos en donde si están definidas detalladamente como por ejemplo en la Dao o en la GUI.
- La forma de conectar el SQLite con Python, para poder trabajar con flujo de información entre estos programas, es muy simplificada. Lo que hace más recomendable el uso conjunto de ambos.
- La creación de los widgets de la interfaz principal puede constituir lo más trabajoso del proyecto puesto que es la parte que más líneas de código genera; sin embargo, muchas de estas son repetidas y solo conllevan algunos cambios entre ellas. Lo que es todo lo contrario a la hora de definir las funcionalidades de estas mismas, donde el nivel de líneas de código va acorde con la dificultad que la cantidad de líneas representa.

5.2. Recomendaciones

- Al momento de utilizar datetime y querer la hora, junto con la fecha, usar strftime, pero no de forma directa: `datetime.date.today().strftime("%d/%m/%Y, %I:%M %p ")`
Sino separar la fecha con today y la hora con now, debido a que en nuestro proyecto generó un error inesperado. Luego solo agregarlos a variables y concatenarlas para poder mostrarlas.

```
xFecha=str(datetime.date.today().strftime("%d/%m/%Y, "))
xHora=str(datetime.datetime.now().strftime("%I:%M %p"))
self.svFechaHistoria.set(xFecha+xHora)
```

- Cuando se ingresa el DNI, si el número empieza en 0, se elimina y en la base de datos, tanto como en el treeview de la interfaz se ven 6 números, no 7, debido a el uso de `tk.StringVar` en casi todas las variables de los entrys. En este caso es recomendable usar `tk.IntVar` para números.
- El volver a cada momento los ID a None es muy importante ya que son las instancias que más se usan en todo el proyecto, tanto para `DatosPaciente` e `HistoriaMedica`, esto se debe a que toman muchos valores en cada función y para que Python no genere error es que se debe agregar este valor vacío al ID para que pueda tomar uno nuevo consecutivamente cada vez que se haga uso de una función al presionar los botones en la interfaz.
- El método `pack` si se sabe manejar correctamente, puede ser mejor opción para ubicar los widgets de Tkinter, pero por simplicidad se utilizó `grid`, lo cual redució notablemente las líneas de código. Con el método `pack` no se distorsiona tanto como con `grid` al maximizar la ventana y es por eso que utilizamos.

```
aplicacion.resizable(width=False, height=False)
```

- Al momento de concatenar en SQLite, no utilizar nuevamente comillas dobles, ya que Python lo leerá como 2 textos y lo que deseamos es que sea uno solo, es por eso que combinamos tanto comillas simples, como dobles.

```
sql = f"SELECT H.idHistorial, D.NombreCompleto || ' ' || D.ApellidosCompleto AS Paciente,
# Cuando se cumple el parámetro WHERE, se cumple el INNER JOIN
```


Referencias Bibliográficas

- EsSalud. (26 de Septiembre de 2019). *Implementa historia clínica digital para atención de asegurados*. Obtenido de EsSalud: <http://noticias.essalud.gob.pe/?publicacion=implementa-historia-clinica-digital-para-atencion-de-asegurados>
- Fernandez, R. (4 de Enero de 2021). *¿Qué es Tkinter y por qué utilizarlo?* Obtenido de Unipython: <https://unipython.com/tkinter-introduccion/>
- Flores, F. (13 de Mayo de 2022). *¿Qué es Visual Studio Code y qué ventajas ofrece?* Obtenido de OpenWebinars: <https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/>
- RPP Noticias. (2 de Febrero de 2022). *Historia clínica electrónica: Cinco ventajas de tener un sistema digital en todos los centros de salud*. Obtenido de RPP Noticias: <https://rpp.pe/campanas/valor-compartido/historia-clinica-electronica-cinco-ventajas-de-tener-un-sistema-digital-en-todos-los-centros-de-salud-el-pais-que-queremos-noticia-1383397?ref=rpp>
- Santander Universidades. (1 de Marzo de 2022). *¿Qué es Python?* Obtenido de Becas Santander: <https://www.becas-santander.com/es/blog/python-que-es.html>
- Sanunga Totoy, J. E., & Pérez Palma, K. N. (28 de Enero de 2019). *Repositorio Institucional de la Universidad Politécnica Salesiana: Implementación del sistema para el control de historia clínica de pacientes en centro odontológico Dental Group*. Obtenido de DSpace: <https://dspace.ups.edu.ec/handle/123456789/16767>
- Silvia Velito, A., & Tejada Soriano, S. R. (2010). *La historia clinica como instrumento de calidad*. Obtenido de [auditoriamedicahoy.com](http://www.auditoriamedicahoy.com/biblioteca/La%20historia%20cl%C3%ADnica%20como%20instrumento%20de%20calidad%20Tejada%20Velito.pdf): <http://www.auditoriamedicahoy.com/biblioteca/La%20historia%20cl%C3%ADnica%20como%20instrumento%20de%20calidad%20Tejada%20Velito.pdf>
- Yrinna Benites, K. A. (19 de Noviembre de 2016). ANÁLISIS Y DISEÑO DE PROTOTIPO DE SOFTWARE PARA LA AUTOMATIZACIÓN DE HISTORIAS CLÍNICAS DEL POLICLÍNICO UDEP. Piura, Perú.

REFERENCIAS LINKOGRÁFICAS

- ANACONDA. (2018). Understanding Conda and Pip. Retrieved from <https://www.anaconda.com/blog/understanding-conda-and-pip>
- Python. (2022). tkinter — Interface de Python para Tcl/Tk. Retrieved from <https://docs.python.org/es/3/library/tkinter.html>
- Estrada web group. (2022). ¿Qué es el tipo de dato varchar de SQL Server y cuando utilizarlo? Retrieved from <https://estradawebgroup.com/Post/-Que-es-el-tipo-de-dato-varchar-de-SQL-Server-y-cuando-utilizarlo-/20355>
- SALVADOR, U. D. EL. (2019). Introducción a SQLITE. Retrieved from https://eisi.fia.ues.edu.sv/materialpublico/pdm115/2019/labs/PDM115_guia_lab03_SQLite.pdf
- Runebook.dev. (n.d.). Tipos de datos en SQLITE. Retrieved from <https://runebook.dev/es/docs/sqlite/datatype3>
- TUTORIAL, P. (2022). Tkinter Object-Oriented Window. Retrieved from <https://www.pythontutorial.net/tkinter/tkinter-object-oriented-window/>
- Guia Tkinter. (2015). Interfaz gráfica con Tkinter. Retrieved from <https://guia-tkinter.readthedocs.io/es/develop/index.html>
- FJSevilla. (2020). Fuentes disponibles en Tkinter. Retrieved from <https://es.stackoverflow.com/questions/330481/fuentes-disponibles-en-tkinter>
- Python. (2022). Envoltorio de fuente Tkinter. Retrieved from <https://docs.python.org/es/3.9/library/tkinter.font.html>
- Colors-symbolic color names recognized by Tk. (2018). Retrieved from <https://www.tcl.tk/man/tcl8.5/TkCmd/colors.html>
- Patricia. (2019). Tabla de colores Tkinter. Retrieved from <http://patriciaemiguel.com/python/2019/08/01/python-tkinter-colores.html>
- SQLite. (2022). SQLite. Retrieved from <https://www.sqlite.org/index.html>
- Anaconda / packages / sqlite. (2022). Retrieved from <https://anaconda.org/anaconda/sqlite>
- Python 3 para impacientes. (2016). Retrieved from <https://python-para-impacientes.blogspot.com/2016/02/variables-de-control-en-tkinter.html>

- Mizipzor. (2009). Understanding Python super() with __init__() methods [duplicate]. Retrieved from <https://stackoverflow.com/questions/576169/understanding-python-super-with-init-methods>
- Educative.io. (n.d.). What is super() in Python? Retrieved from <https://www.educative.io/answers/what-is-super-in-python>
- Dev Prakash, S. (2021). How to Disable / Enable a Button in Tkinter? Retrieved from <https://www.tutorialspoint.com/how-to-disable-enable-a-button-in-tkinter#:~:text=Tkinter Button widgets can be,and disabling the button%2C respectively.>
- tutorialesprogramacion. (n.d.). Tipos de datos básicos para definir los campos de una tabla. Retrieved from <https://www.tutorialesprogramacionya.com/sqliteya/detalleconcepto.php?punto=4&codigo=4&inicio=0>
- JC Chouinard. (2020). if __name__ == '__main__': What does it mean (Python). Retrieved from <https://www.jcchouinard.com/python-if-name-equals-main/#:~:text=When you start working with,What does it mean%3F&text=What this function do is,the module from another one.%0A>
- Python. (2022). tkinter.messagebox. Retrieved from <https://docs.python.org/3/library/tkinter.messagebox.html>
- Python. (n.d.). tkinter.ttk — Tk widgets temáticos. Retrieved from <https://docs.python.org/es/3/library/tkinter.ttk.html%0A>
- Python. (n.d.). tkinter.ttk — Tk widgets temáticos. Retrieved from <https://stackoverflow.com/questions/19561727/what-is-the-difference-between-the-widgets-of-tkinter-and-tkinter-ttk-in-python>
- recursos python. (2017). Posicionar elementos en Tkinter. Retrieved from <https://recursospython.com/guias-y-manuales/posicionar-elementos-en-tkinter/>
- Manuel, G. (2022). Python-Tkinter. Retrieved from https://www.youtube.com/watch?v=Yvs7YFmKwuE&list=PLh7JzoyIyU4LKz9h3KC7VNrKPdkm0o8N4&index=20&ab_channel=ManuelGonzález%0A
- RIP Tutorial. (n.d.). Difference between Tk and Toplevel. Retrieved from <https://riptutorial.com/tkinter/example/22130/difference-between-tk-and-toplevel>
- Python. (2022). *Funciones generales relacionadas con el calendario*. 28 de Agosto. <https://docs.python.org/es/3/library/calendar.html>



pypi. (2019). *Tkcalendar 1.6.1*. 28 de Diciembre. <https://pypi.org/project/tkcalendar/>

J2LOGO. (n.d.). **args y **kwargs en Python. Una explicación y ejemplos de uso*.
<https://j2logo.com/args-y-kwargs-en-python/>

Python. (n.d.). *Tipos básicos de fecha y hora*.
<https://docs.python.org/es/3/library/datetime.html#module-datetime>

Programación-y-más. (2021). *¿Cómo funciona INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN?* <https://programacionymas.com/blog/como-funciona-inner-left-right-full-join>

Anaconda.org. (2021). *conda-forge / packages / fpdf*. Enero. <https://anaconda.org/conda-forge/fpdf>

ANEXOS

- ANEXO 1:

```
PROGRAMA.py > main
1  import tkinter as tk
2  from INTERFAZ.GUI import Frame
3  from INTERFAZ.GUI import error
4
5  def main():
6      aplicacion = tk.Tk()
7      aplicacion.title("HISTORIAS CLINICAS")
8      aplicacion.resizable(width=False, height=False)
9      aplicacion.geometry("1420x720+40+40")
10     aplicacion.minsize(width=1280, height=720)
11     aplicacion.iconbitmap("ICONOS/ICONO.ico")
12     fondo = Frame(aplicacion)
13     fondo.mainloop()
14
15     try:
16         if __name__ == "__main__":
17             main()
18     except:
19         error()
```

- ANEXO 2:

```
CONEXION > conexion.py > ...
1  import sqlite3
2
3  class ConexionDB:
4      def __init__(self):
5          self.baseDatos = "BaseDeDatos/DbHistorial.db"
6          self.conexion = sqlite3.connect(self.baseDatos)
7          self.cursor = self.conexion.cursor()
8
9      def cerrarConexion(self):
10         self.conexion.commit()
11         self.conexion.close()
```

- ANEXO 3:

```
1  from CONEXION.conexion import ConexionDB
2  from tkinter import messagebox
```

- ANEXO 4:

```
9  def guardarDatoPaciente(persona):
10     conexion = ConexionDB()
11     sql = f"""INSERT INTO DatosPaciente (NombreCompleto, ApellidosCompleto, DNI, FechaNacimiento
12         |      ({persona.NombreCompleto},"{persona.ApellidosCompleto}", "{persona.DNI}", "{perso
13
14     conexion.cursor.execute(sql)
15     conexion.cerrarConexion()
16     titulo = "Registrar Paciente"
17     mensaje = "Paciente Registrado Exitosamente"
18     messagebox.showinfo(titulo, mensaje)
```

- ANEXO 5:

```
22 def eliminarPaciente(idPersona):
23     conexion = ConexionDB()
24     sql = f"""UPDATE DatosPaciente SET activo = 0 WHERE idPersona = {idPersona}"""
25     try:
26         conexion.cursor.execute(sql)
27         conexion.cerrarConexion()
28         titulo = "Eliminar Paciente"
29         mensaje = "Paciente eliminado exitosamente"
30         messagebox.showinfo(titulo,mensaje)
31     except:
32         titulo = "Eliminar Paciente"
33         mensaje = "Error al eliminar Paciente"
34         messagebox.showwarning(titulo, mensaje)
```

- ANEXO 6:

```
38 def editarDatoPaciente(persona, idPersona):
39     conexion = ConexionDB()
40     sql = f"""UPDATE DatosPaciente SET NombreCompleto = "{persona.NombreCompleto}", ApellidosCompleto =
41         |      DNI = "{persona.DNI}", FechaNacimiento = "{persona.FechaNacimiento}", Edad = "{persona.
42         |      NumeroTelefonico = "{persona.NumeroTelefonico}", CorreoElectronico = "{persona.CorreoEl
43         |      activo = 1 WHERE idPersona = {idPersona}"""
44     try:
45         conexion.cursor.execute(sql)
46         conexion.cerrarConexion()
47         titulo = "Editar Paciente"
48         mensaje = "Paciente Editado Exitosamente"
49         messagebox.showinfo(titulo, mensaje)
50     except:
51         titulo = "Editar Paciente"
52         mensaje = "Error al editar paciente"
53         messagebox.showinfo(titulo, mensaje)
```

- **ANEXO 7:**

```
class DatosPaciente:
    def __init__(self, NombreCompleto, ApellidosCompleto, DNI, FechaNacimiento, Edad, NumeroTelefonico, CorreoElectronico):
        self.idPersona = None
        self.NombreCompleto = NombreCompleto
        self.ApellidosCompleto = ApellidosCompleto
        self.DNI = DNI
        self.FechaNacimiento = FechaNacimiento
        self.Edad = Edad
        self.NumeroTelefonico = NumeroTelefonico
        self.CorreoElectronico = CorreoElectronico

    def __str__(self):
        return f"DatosPaciente[{self.NombreCompleto},{self.ApellidosCompleto}, {self.DNI}]"
```

- **ANEXO 8:**

```
57 def listar():
58     conexion = ConexionDB()
59
60     ListaDatosPaciente = []
61     sql = "SELECT * FROM DatosPaciente WHERE activo = 1"
62
63     try:
64         conexion.cursor().execute(sql)
65         ListaDatosPaciente = conexion.cursor().fetchall()
66         conexion.cerrarConexion()
67     except:
68         title = "Datos"
69         mensaje = "Registros no existen"
70         messagebox.showwarning(title, mensaje)
71     return ListaDatosPaciente
```

- ANEXO 9:

```
73 def listarCondicion(where):
74     conexion = ConexionDB()
75     listaDatosPaciente = []
76     sql = f"SELECT * FROM DatosPaciente {where}"
77
78     try:
79         conexion.cursor.execute(sql)
80         listaDatosPaciente = conexion.cursor.fetchall()
81         conexion.cerrarConexion()
82     except:
83         title = "Datos"
84         mensaje = "Registros no existen"
85         messagebox.showwarning(title, mensaje)
86     return listaDatosPaciente
```

- ANEXO 10:

```
CONEXION > HistorialDao.py > ...
1  from CONEXION.conexion import ConexionDB
2  from tkinter import messagebox
3
4
5
6  class HistoriaMedica:
7      """Clase de la tabla HistoriaMedica"""
8
9      def __init__(self, idPersona, FechaHistoria, MotivoDeLaVisita, Operacion, Tratamiento, DetalleAdicional):
10         """Constructor cuyos parámetros son los nombres de las columnas de la Tabla HistoriaMedica"""
11
12         self.idHistorial = None
13         self.idPersona = idPersona
14         self.FechaHistoria = FechaHistoria
15         self.MotivoDeLaVisita = MotivoDeLaVisita
16         self.Operacion = Operacion
17         self.Tratamiento = Tratamiento
18         self.DetalleAdicional = DetalleAdicional
19
20     def __str__(self):
21         """Método que muestra los objetos"""
22
23         return f"HistoriaMedica[{self.idPersona},{self.FechaHistoria},{self.MotivoDeLaVisita}, {self.Operacion}"
24
```


- **ANEXO 11:**

```
29 def guardarHistoria(idPersona, FechaHistoria, MotivoDeLaVisita, Operacion, Tratamiento, DetalleAdicional):
30     conexion = ConexionDB()
31     sql = f"""INSERT INTO HistoriaMedica (idPersona, FechaHistoria, MotivoDeLaVisita, Operacion, Tratamiento, DetalleAdicional)
32         VALUES ({idPersona}, '{FechaHistoria}', '{MotivoDeLaVisita}', '{Operacion}', '{Tratamiento}', '{DetalleAdicional}');
33     try:
34         conexion.cursor().execute(sql)
35         conexion.cerrarConexion()
36         titulo = "Registro Historia Medica"
37         mensaje = "Historia registrada exitosamente"
38         messagebox.showinfo(titulo, mensaje)
39     except:
40         titulo = "Registro Historia Medica"
41         mensaje = "Error al registrar historia"
42         messagebox.showerror(titulo, mensaje)
```

- **ANEXO 12:**

```
46 def eliminarHistoria(idHistorial):
47     conexion = ConexionDB()
48     sql = f"DELETE FROM HistoriaMedica WHERE idHistorial = {idHistorial}"
49
50     try:
51         conexion.cursor().execute(sql)
52         conexion.cerrarConexion()
53         titulo = "Eliminar Historia"
54         mensaje = "Historia medica eliminada exitosamente"
55         messagebox.showinfo(titulo, mensaje)
56     except:
57         titulo = "Eliminar Historia"
58         mensaje = "Error al eliminar historia medica"
59         messagebox.showerror(titulo, mensaje)
```

- **ANEXO 13:**

```
63 def editarHistoria(fechaHistoria, MotivoDeLaVisita, Operacion, Tratamiento, DetalleAdicional, idHistorial):
64     conexion = ConexionDB()
65     sql = f"""UPDATE HistoriaMedica SET fechaHistoria = '{fechaHistoria}', MotivoDeLaVisita = '{MotivoDeLaVisita}', Operacion = '{Operacion}', Tratamiento = '{Tratamiento}', DetalleAdicional = '{DetalleAdicional}'
66     WHERE idHistorial = {idHistorial};
67     try:
68         conexion.cursor().execute(sql)
69         conexion.cerrarConexion()
70         titulo = "Editar Historia"
71         mensaje = "Historia medica editada exitosamente"
72         messagebox.showinfo(titulo, mensaje)
73     except:
74         titulo = "Editar Historia"
75         mensaje = "Error al editar historia medica"
76         messagebox.showerror(titulo, mensaje)
```

- **ANEXO 14:**

```
79 def listarHistoria(idPersona):
80     conexion = ConexionDB()
81     listaHistoria = []
82
83     sql = f"SELECT H.idHistorial, D.NombreCompleto || ' ' || D.ApellidosCo
84
85
86     try:
87         conexion.cursor.execute(sql)
88         listaHistoria = conexion.cursor.fetchall()
89         conexion.cerrarConexion()
90     except:
91         titulo = "LISTAR"
92         mensaje = "Error al listar historia medica"
93         messagebox.showerror(titulo, mensaje)
94
95     return listaHistoria
```

- **ANEXO 15:**

```
INTERFAZ > GUI.py > ...
1  from CONEXION.PacienteDao import DatosPaciente, editarDatoPaciente, guardarDatoPaciente, listar, listarCondicion, eliminarPaciente
2  from CONEXION.HistorialDao import guardarHistoria, editarHistoria, eliminarHistoria, listarHistoria
3  import tkinter as tk
4  from tkinter import W, ttk, messagebox, Toplevel
5  import tkcalendar as tc
6  import datetime
7  from PIL import Image, ImageTk
8  import fpdf
```

- ANEXO 16:

```
class Frame(tk.Frame):  
  
    def __init__(self, aplicacion):  
  
        super().__init__(aplicacion)  
        self.aplicacion = aplicacion  
        self.pack(fill=tk.BOTH, expand=True)  
        self.config(background="lightseagreen")  
        self.idPersona = None  
        self.idPersonaHistoria=None  
        self.idHistoriaMedica=None  
        self.idHistoriaMedicaEditar=None  
        self.camposPaciente()  
        self.deshabilitar()  
        self.tablaPaciente()
```

- ANEXO 17:

```
def camposPaciente(self):  
  
    self.lblNombre = tk.Label(self, text="NOMBRE COMPLETO: ")  
    self.lblNombre.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblNombre.grid(column=0, row=0, pady=5)  
  
    self.lblApellidos = tk.Label(self, text="APELLIDOS COMPLETOS: ")  
    self.lblApellidos.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblApellidos.grid(column=0,row=1, pady=5)  
  
    self.lblDni = tk.Label(self, text="DNI: ")  
    self.lblDni.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblDni.grid(column=0,row=2, padx=10, pady=5)  
  
    self.lblFechNacimiento = tk.Label(self, text="FECHA DE NACIMIENTO: ")  
    self.lblFechNacimiento.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblFechNacimiento.grid(column=0,row=3, pady=5)  
  
    self.lblEdad = tk.Label(self, text="EDAD: ")  
    self.lblEdad.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblEdad.grid(column=0,row=4, pady=5)  
  
    self.lblTelefono = tk.Label(self, text="TELÉFONO: ")  
    self.lblTelefono.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblTelefono.grid(column=0,row=5, pady=5)  
  
    self.lblCorreo = tk.Label(self, text="CORREO ELECTRÓNICO: ")  
    self.lblCorreo.config(font=("verdana",15,"bold"), background="lightseagreen", anchor = "w")  
    self.lblCorreo.grid(column=0,row=6, pady=5)
```

- ANEXO 18:

```
self.svNombre = tk.StringVar()
self.entryNombre = tk.Entry(self, textvariable=self.svNombre)
self.entryNombre.config(width=50, font=("verdana",15))
self.entryNombre.grid(column=1, row=0, padx=10, pady=5, columnspan=2)

self.svApellidos = tk.StringVar()
self.entryApellidos = tk.Entry(self, textvariable=self.svApellidos)
self.entryApellidos.config(width=50, font=("verdana",15))
self.entryApellidos.grid(column=1, row=1, padx=10, pady=5, columnspan=2)

self.svDni = tk.StringVar()
self.entryDni = tk.Entry(self, textvariable=self.svDni)
self.entryDni.config(width=50, font=("verdana",15))
self.entryDni.grid(column=1, row=2, padx=10, pady=5, columnspan=2)

self.svFecNacimiento = tk.StringVar()
self.entryFecNacimiento = tk.Entry(self, textvariable=self.svFecNacimiento)
self.entryFecNacimiento.config(width=50, font=("verdana",15))
self.entryFecNacimiento.grid(column=1, row=3, padx=10, pady=5, columnspan=2)

self.svEdad = tk.StringVar()
self.entryEdad = tk.Entry(self, textvariable=self.svEdad)
self.entryEdad.config(width=50, font=("verdana",15))
self.entryEdad.grid(column=1, row=4, padx=10, pady=5, columnspan=2)

self.svTelefono = tk.StringVar()
self.entryTelefono = tk.Entry(self, textvariable=self.svTelefono)
self.entryTelefono.config(width=50, font=("verdana",15))
self.entryTelefono.grid(column=1, row=5, padx=10, pady=5, columnspan=2)

self.svCorreo = tk.StringVar()
self.entryCorreo = tk.Entry(self, textvariable=self.svCorreo)
self.entryCorreo.config(width=50, font=("verdana",15))
self.entryCorreo.grid(column=1, row=6, padx=10, pady=5, columnspan=2)
```

- ANEXO 19:

```
self.btnNuevo = tk.Button(self, text="NUEVO", command=self.habilitar)
self.btnNuevo.config(width=20, font=("verdana",12,"bold"),
                      background="firebrick1", cursor="hand2",activebackground="firebrick3")
self.btnNuevo.grid(column=0,row=7, padx=10, pady=5)

self.btnGuardar = tk.Button(self, text="GUARDAR", command=self.guardarPaciente)
self.btnGuardar.config(width=20, font=("verdana",12,"bold"),
                      background="sienna1", cursor="hand2",activebackground="sienna3")
self.btnGuardar.grid(column=1,row=7, padx=10, pady=5)

self.btnCancelar = tk.Button(self, text="CANCELAR")
self.btnCancelar.config(width=20, font=("verdana",12,"bold"),
                      background="khaki1", cursor="hand2",activebackground="khaki3")
self.btnCancelar.grid(column=2,row=7, padx=10, pady=5)
```

- ANEXO 20:

```
self.lblBuscarDni = tk.Label(self, text="Buscar DNI:")
self.lblBuscarDni.config(font=("verdana",15,"bold"), bg="lightseagreen")
self.lblBuscarDni.grid(column=3, row=0, padx=2, pady=5)

self.svBuscarDni = tk.StringVar()
self.entryBuscarDni = tk.Entry(self, textvariable=self.svBuscarDni)
self.entryBuscarDni.config(width=20, font=("verdana",15))
self.entryBuscarDni.grid(column=4, row=0, padx=2, pady=5)

self.btnBuscarCondicion = tk.Button(self, text="BUSCAR", command=self.buscarCondicion)
self.btnBuscarCondicion.config(width=10, font=("verdana",12,"bold"),
                              bg="purple1", cursor="hand2",activebackground="purple3")
self.btnBuscarCondicion.grid(column=4,row=1, padx=2, pady=5)
```

- ANEXO 21:

```
def buscarCondicion(self):  
    if len(self.svBuscarDni.get()) > 0:  
        where = "WHERE 1=1"  
  
        if len(self.svBuscarDni.get()) > 0:  
            where = "WHERE Dni = " + self.svBuscarDni.get() + "  
            self.tablaPaciente(where)  
        else:  
            self.tablaPaciente()  
    else:  
        if len(self.svBuscarDni.get()) == 0:  
            where = "WHERE activo = 0 "  
            self.tablaPaciente(where)
```

- ANEXO 22:

```
def deshabilitar(self):  
  
    self.idPersona = None  
    self.svNombre.set("")  
    self.svApellidos.set("")  
    self.svDni.set("")  
    self.svFecNacimiento.set("")  
    self.svEdad.set("")  
    self.svTelefono.set("")  
    self.svCorreo.set("")  
  
    self.entryNombre.config(state="disabled")  
    self.entryApellidos.config(state="disabled")  
    self.entryDni.config(state="disabled")  
    self.entryFecNacimiento.config(state="disabled")  
    self.entryEdad.config(state="disabled")  
    self.entryCorreo.config(state="disabled")  
    self.entryTelefono.config(state="disabled")  
  
    self.btnCancelar.config(state="disabled")  
    self.btnGuardar.config(state="disabled")
```

- ANEXO 23:

```
def habilitar(self):

    self.svNombre.set("")
    self.svApellidos.set("")
    self.svDni.set("")
    self.svFecNacimiento.set("")
    self.svEdad.set("")
    self.svTelefono.set("")
    self.svCorreo.set("")

    self.entryNombre.config(state="normal")
    self.entryApellidos.config(state="normal")
    self.entryDni.config(state="normal")
    self.entryFecNacimiento.config(state="normal")
    self.entryEdad.config(state="normal")
    self.entryCorreo.config(state="normal")
    self.entryTelefono.config(state="normal")

    self.btnCancelar.config(state="normal")
    self.btnGuardar.config(state="normal")
```

- ANEXO 24:

```
def tablaPaciente(self, where=""):

    if len(where) > 0:
        self.listaDatosPaciente = listarCondicion(where)
    else:
        self.listaDatosPaciente = listar()
        self.listaDatosPaciente.reverse()

    self.tabla = ttk.Treeview(self, column=("NombreCompleto", "ApellidosCompleto", "DNI", "FechaNacimiento",
                                             "Edad", "NumeroTelefonico", "CorreoElectronico"))

    self.tabla.grid(column=0, row=8, columnspan=5, sticky="nswe")

    self.scroll = ttk.Scrollbar(self, orient="vertical", command=self.tabla.yview)
    self.scroll.grid(row=8, column=4, sticky="nse")
    self.tabla.config(height=15)

    self.tabla.configure(yscrollcommand=self.scroll.set)

    self.tabla.tag_configure("evenrow", background="antiquewhite1")
```

- **ANEXO 25:**

```
self.tabla.heading("#0",text="ID")
self.tabla.heading("#1",text="Nombres")
self.tabla.heading("#2",text="Apellidos")
self.tabla.heading("#3",text="DNI")
self.tabla.heading("#4",text="F. Nacimiento")
self.tabla.heading("#5",text="Edad")
self.tabla.heading("#6",text="Telefono")
self.tabla.heading("#7",text="Correo")

self.tabla.column("#0", anchor=W, width=3)
self.tabla.column("#1", anchor=W, width=120)
self.tabla.column("#2", anchor=W, width=120)
self.tabla.column("#3", anchor=W, width=20)
self.tabla.column("#4", anchor=W, width=100)
self.tabla.column("#5", anchor=W, width=3)
self.tabla.column("#6", anchor=W, width=40)
self.tabla.column("#7", anchor=W, width=120)

for p in self.listaDatosPaciente:
    self.tabla.insert("",0,text=p[0], values=(p[1],p[2],p[3],p[4],p[5],p[6],p[7]), tags=("evenrow",))
```

- **ANEXO 26:**

```
self.btnEditarPaciente = tk.Button(self, text="Editar Paciente", command=self.editarPaciente)
self.btnEditarPaciente.config(width=20,font=("verdana",12,"bold"), bg="orange", activebackground="darkorange", cursor="hand2")
self.btnEditarPaciente.grid(row=9, column=0, padx=10, pady=5)

self.btnEliminarPaciente = tk.Button(self, text="Eliminar Paciente", command=self.eliminarDatoPaciente)
self.btnEliminarPaciente.config(width=20,font=("verdana",12,"bold"), bg="cornflowerblue", activebackground="royalblue2", cursor="hand2")
self.btnEliminarPaciente.grid(row=9, column=1, padx=10, pady=5)

self.btnHistorialPaciente = tk.Button(self, text="Historial Paciente")
self.btnHistorialPaciente.config(width=50,font=("verdana",12,"bold"), bg="cyan2", activebackground="cyan3", cursor="hand2")
self.btnHistorialPaciente.grid(row=9, column=2, columnspan=3, pady=5)
```

- **ANEXO 27:**

```
def MostrarCalendario(self):
    """Función que muestra el calendario en la interfaz"""

    self.topCalendario = Toplevel() #Creamos una ventana flotante
    self.topCalendario.title("FECHA DE NACIMIENTO")
    self.topCalendario.geometry("300x300+1000+80")
    self.topCalendario.resizable(width=False, height=False)
    self.topCalendario.iconbitmap("ICONOS/calendario.ico")
    self.topCalendario.config(background="lightseagreen")

    self.calendar = tc.Calendar(self.topCalendario, selectmode="day", year=1990, month=3, day=20, locale ="es_US", background="lightseagreen")
    self.calendar.pack(fill=tk.BOTH, expand=True)

    self.btnCalendario2 = tk.Button(self.topCalendario, text="INSERTAR")
    self.btnCalendario2.config(width=13, font=("verdana",12,"bold"), command=self.EnviaFecha,
                                background="navajowhite3", cursor="hand2",activebackground="navajowhite4")
    self.btnCalendario2.pack(fill=tk.BOTH)
```


- **ANEXO 28:**

```
def EnviarFecha(self):  
  
    self.svFecNacimiento.set(self.calendar.get_date())  
  
    if len(self.calendar.get_date())>1:  
        self.CalcularEdad()
```

- **ANEXO 29**

```
def CalcularEdad(self):  
  
    self.fechaActual = datetime.date.today()  
    self.date1 = self.calendar.get_date()  
    self.convertidor = datetime.datetime.strptime(self.date1, "%d/%m/%Y")  
  
    self.resultado = self.fechaActual.year - self.convertidor.year  
    self.svEdad.set(self.resultado)
```

- **ANEXO 30:**

```
def guardarPaciente(self):  
    persona = DatosPaciente(self.svNombre.get(), self.svApellidos.get(), self.svDni.get(), self.svFecNacimiento.get(),  
                             self.svEdad.get(), self.svTelefono.get(), self.svCorreo.get())  
  
    if self.idPersona == None:  
        guardarDatoPaciente(persona)  
    else:  
        editarDatoPaciente(persona, self.idPersona)  
  
    self.deshabilitar()  
    self.tablaPaciente()
```

- **ANEXO 31:**

```
def editarPaciente(self):
    try:
        self.idPersona = self.tabla.item(self.tabla.selection())["text"]
        self.NombrePaciente = self.tabla.item(self.tabla.selection())["values"][0]
        self.ApellidosPaciente = self.tabla.item(self.tabla.selection())["values"][1]
        self.DNIPaciente = self.tabla.item(self.tabla.selection())["values"][2]
        self.FecNacimientoPaciente = self.tabla.item(self.tabla.selection())["values"][3]
        self.EdadPaciente = self.tabla.item(self.tabla.selection())["values"][4]
        self.TelefonoPaciente = self.tabla.item(self.tabla.selection())["values"][5]
        self.CorreoPaciente = self.tabla.item(self.tabla.selection())["values"][6]

        self.habilitar()

        self.entryNombre.insert(0, self.NombrePaciente) |
        self.entryApellidos.insert(0, self.ApellidosPaciente)
        self.entryDni.insert(0, self.DNIPaciente)
        self.entryFecNacimiento.insert(0, self.FecNacimientoPaciente)
        self.entryEdad.insert(0, self.EdadPaciente)
        self.entryTelefono.insert(0, self.TelefonoPaciente)
        self.entryCorreo.insert(0, self.CorreoPaciente)

    except:
        title = "Editar Paciente"
        mensaje = "Error al editar paciente"
        messagebox.showerror(title, mensaje)
```

- **ANEXO 32:**

```
def eliminarDatoPaciente(self):
    try:
        self.idPersona = self.tabla.item(self.tabla.selection())["text"]
        eliminarPaciente(self.idPersona)

        self.tablaPaciente()
        self.idPersona = None
    except:
        title = "Eliminar Paciente"
        mensaje = "No se pudo eliminar paciente"
        messagebox.showinfo(title, mensaje)
```

- **ANEXO 33:**

```
def tablaHistoria(self):
    """función que muestra todos los historiales del paciente en un Treeview"""
    try:
        if self.idPersona == None:
            self.idPersona = self.tabla.item(self.tabla.selection())["text"]
            self.idPersonaHistoria = self.idPersona

        if self.idPersona > 0:
            idPersona = self.idPersona

        self.ListaHistoria = listarHistoria(idPersona)
        self.tabla2 = ttk.Treeview(self.topHistoriaMedica, column=("Paciente", "FechaHistoria", "MotivoDe
        self.tabla2.config(height=10)
        self.tabla2.tag_configure("evenrow", background="oldlace")
        self.tabla2.grid(column=0, row=0, columnspan=7, sticky="nse")

        self.scroll2=ttk.Scrollbar(self.topHistoriaMedica, orient="vertical", command=self.tabla2.yview)
        self.scroll2.grid(row=0, column=7, sticky="nse")

        self.tabla2.configure(yscrollcommand=self.scroll2.set)
```

```
self.tabla2.heading("#0",text="ID")
self.tabla2.heading("#1",text="Paciente")
self.tabla2.heading("#2",text="Fecha y Hora")
self.tabla2.heading("#3",text="Motivo de la visita")
self.tabla2.heading("#4",text="Operacion")
self.tabla2.heading("#5",text="Tratamiento")
self.tabla2.heading("#6",text="Detalle adicional")

self.tabla2.column("#0", anchor=W, width=40)
self.tabla2.column("#1", anchor=W, width=200)
self.tabla2.column("#2", anchor=W, width=150)
self.tabla2.column("#3", anchor=W, width=200)
self.tabla2.column("#4", anchor=W, width=150)
self.tabla2.column("#5", anchor=W, width=300)
self.tabla2.column("#6", anchor=W, width=450)

for p in self.ListaHistoria:
    self.tabla2.insert("",0,text=p[0], values=(p[1],p[2],p[3],p[4],p[5],p[6]), tags=("evenrow",))

except:

    titulo = "Historia Medica"
    mensaje = "Error al mostrar historial"
    messagebox.showerror(titulo, mensaje)
    self.idPersona = None
```

- **ANEXO 34:**

```
def historiaMedica(self):
    """Función para generar el Top Level donde estará la tabla de los historiales del paciente"""

    self.topHistoriaMedica = Toplevel()
    self.topHistoriaMedica.title("HISTORIAL MEDICO")
    self.topHistoriaMedica.geometry("1515x280+5+50")
    self.topHistoriaMedica.resizable(width=False, height=False)
    self.topHistoriaMedica.iconbitmap("ICONOS/Historial.ico")
    self.topHistoriaMedica.config(background="azure")

    self.tablaHistoria()

    self.btnGuardarHistoria=tk.Button(self.topHistoriaMedica, text="Agregar Historia", command=self.topAgregarHistoria)
    self.btnGuardarHistoria.config(width=20, font=("Verdana", 12, "bold"),
                                    foreground="gray2", background="seagreen1", activebackground="seagreen3", cursor="hand2")
    self.btnGuardarHistoria.grid(column=0, row=1, padx=10, pady=10)

    self.btnEditarHistoria=tk.Button(self.topHistoriaMedica, text="Editar Historia", command=self.topEditarHistorialMedico)
    self.btnEditarHistoria.config(width=20, font=("Verdana", 12, "bold"),
                                    foreground="gray2", background="salmon2", activebackground="salmon4", cursor="hand2")
    self.btnEditarHistoria.grid(column=1, row=1, padx=10, pady=10)

    self.btnEliminarHistoria=tk.Button(self.topHistoriaMedica, text="Eliminar Historia", command=self.eliminarHistorialMedico)
    self.btnEliminarHistoria.config(width=20, font=("Verdana", 12, "bold"),
                                    foreground="gray2", background="purple2", activebackground="purple4", cursor="hand2")
    self.btnEliminarHistoria.grid(column=2, row=1, padx=10, pady=10)

    self.btnEnviar=tk.Button(self.topHistoriaMedica, text="Generar PDF", command=self.crearPDF)
    self.btnEnviar.config(width=20, font=("Verdana", 12, "bold"), foreground="gray2",
                           background="tomato2", activebackground="tomato3", cursor="hand2")
    self.btnEnviar.grid(column=3, row=1, padx=10, pady=10)

    self.idPersona = None
```

- **ANEXO 35:**

```
def crearPDF(self):
    """función para crear el historial de cada paciente"""

    pdf=fpdf.FPDF(orientation="L",unit="mm",format="A4")
    pdf.add_page()

    id=self.tabla2.item(self.tabla2.selection())["text"]
    n=self.tabla2.item(self.tabla2.selection())["values"][0]
    f=self.tabla2.item(self.tabla2.selection())["values"][1]
    a=self.tabla2.item(self.tabla2.selection())["values"][2]
    b=self.tabla2.item(self.tabla2.selection())["values"][3]
    c=self.tabla2.item(self.tabla2.selection())["values"][4]
    d=self.tabla2.item(self.tabla2.selection())["values"][5]
    g=str(self.tabla.item(self.tabla.selection())["values"][2])

    pdf.set_font("Arial","",14)
    pdf.text(x=230,y=10, txt = "Generado el: "+str(datetime.date.today()))
    pdf.text(x=130, y=200, txt = "Trujillo-PERÚ")

    pdf.image("ICONOS/UNT.png", x=250, y=15, w=40, h=35)
    pdf.image("ICONOS/vigo.png", x=45, y=160, w=40, h=20)
    pdf.image("ICONOS/jonathan.png", x=130, y=160, w=40, h=20)
    pdf.image("ICONOS/victor.png", x=215, y=160, w=40, h=20)

    pdf.set_font("Arial","B",20)
    pdf.text(x=125, y=15, txt="CLÍNICA UNT")

    pdf.set_font("Arial","B",16)
    pdf.text(x=10, y=35, txt="Paciente:")
    pdf.text(x=10, y=50, txt="DNI:")
    pdf.text(x=10, y=65, txt="Motivo de la visita a la clínica:")
    pdf.text(x=10, y=80, txt="Fecha y hora de la visita:")
    pdf.text(x=10, y=95, txt="Su operación fue:")
    pdf.text(x=10, y=110, txt="Debe seguir el siguiente tratamiento:")
    pdf.text(x=10, y=125, txt="Detalles adicionales:")

    pdf.set_font("Arial","",16)
    pdf.text(x=40, y=35, txt=n)
    pdf.text(x=30, y=50, txt=g)
    pdf.text(x=100, y=65, txt=a)
    pdf.text(x=90, y=80, txt=f)
    pdf.text(x=70, y=95, txt=b)
    pdf.text(x=120, y=110, txt=c)
    pdf.text(x=80, y=125, txt=d)

    pdf.set_font("Arial","B",14)
    pdf.text(x=35,y=180, txt="Vigo Villar Cristhian A.")
    pdf.text(x=117,y=180, txt="Sanchez Rojas Jonathan A.")
    pdf.text(x=200,y=180, txt="Valdiviezo Jimenez Victor J.")

    pdf.set_font("Arial","",14)
    pdf.text(x=50, y=185, txt="Cirujano")
    pdf.text(x=137, y=185, txt="Pediatra")
    pdf.text(x=220, y=185, txt="Traumatólogo")

    pdf.output(f"HISTORIALES_PDF/Historial_{id}_{n}.pdf")
```

- ANEXO 36:

```
def topAgregarHistoria(self):  
    """Función para generar el Top Level donde agregaremos los datos del historial"""  
  
    self.topAHistoria = tk.Toplevel()  
    self.topAHistoria.title("AGREGAR HISTORIA")  
    self.topAHistoria.geometry("900x450+300+350")  
    self.topAHistoria.resizable(width=False, height=False)  
    self.topAHistoria.iconbitmap("ICONOS/ICONO.ico")  
    self.topAHistoria.config(background="cornsilk2")
```

```
##### FRAME 1 #####  
  
self.frameDatosHistoria = tk.LabelFrame(self.topAHistoria)  
self.frameDatosHistoria.config(background="cornsilk2")  
self.frameDatosHistoria.pack(fill=tk.BOTH, expand=True, pady=10, padx=20)  
  
##### LABELS-F1 #####  
  
self.lblMotivo = tk.Label(self.frameDatosHistoria, text="Motivo de la visita", width=30, font=("Verdana", 15,"bold"), background="cornsilk2")  
self.lblMotivo.grid(row=0, column=0, padx=5, pady=3)  
  
self.lblOperacion = tk.Label(self.frameDatosHistoria, text="Operación", width=20, font=("Verdana", 15,"bold"), background="cornsilk2")  
self.lblOperacion.grid(row=2, column=0, padx=5, pady=3)  
  
self.lblTratamiento = tk.Label(self.frameDatosHistoria, text="Tratamiento", width=20, font=("Verdana", 15,"bold"), background="cornsilk2")  
self.lblTratamiento.grid(row=4, column=0, padx=5, pady=3)  
  
self.lblDetalle = tk.Label(self.frameDatosHistoria, text="Detalle adicional", width=30, font=("Verdana", 15,"bold"), background="cornsilk2")  
self.lblDetalle.grid(row=6, column=0, padx=5, pady=3)  
  
##### ENTRIES-F1 #####  
  
self.svMotivo = tk.StringVar()  
self.Motivo = tk.Entry(self.frameDatosHistoria, textvariable=self.svMotivo)  
self.Motivo.config(width=64, font=("Verdana", 15))  
self.Motivo.grid(row=1, column=0, padx= 5, pady=3, columnspan=2)  
  
self.svOperacion = tk.StringVar()  
self.Operacion = tk.Entry(self.frameDatosHistoria, textvariable=self.svOperacion)  
self.Operacion.config(width=64, font=("Verdana", 15))  
self.Operacion.grid(row=3, column=0, padx= 5, pady=3, columnspan=2)  
  
self.svTratamiento = tk.StringVar()  
self.Tratamiento = tk.Entry(self.frameDatosHistoria, textvariable=self.svTratamiento)  
self.Tratamiento.config(width=64, font=("Verdana", 15))  
self.Tratamiento.grid(row=5, column=0, padx= 5, pady=3, columnspan=2)  
  
self.svDetalle = tk.StringVar()  
self.Detalle = tk.Entry(self.frameDatosHistoria, textvariable=self.svDetalle)  
self.Detalle.config(width=64, font=("Verdana", 15))  
self.Detalle.grid(row=7, column=0, padx= 5, pady=3, columnspan=2)
```

```
##### FRAME 2 #####

self.frameFechaHistoria = tk.LabelFrame(self.topAHistoria)
self.frameFechaHistoria.config(bg="cornsilk2")
self.frameFechaHistoria.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

##### LABELS-F2 #####

self.lblFechaHistoria = tk.Label(self.frameFechaHistoria, text="Fecha y Hora", width=20, font=("Verdana", 15, "bold"), background="cornsilk3")
self.lblFechaHistoria.grid(row=1, column=0, padx=5, pady=3)

##### ENTRIES-F2 #####

self.svFechaHistoria = tk.StringVar()
self.entryFechaHistoria = tk.Entry(self.frameFechaHistoria, textvariable=self.svFechaHistoria)
self.entryFechaHistoria.config(width=20, font=("Verdana", 15))
self.entryFechaHistoria.grid(row=1, column=1, padx=5, pady=3)

##### FECHA Y HORA-F2 #####

xFecha=str(datetime.date.today().strftime("%d/%m/%Y, "))
xHora=str(datetime.datetime.now().strftime("%I:%M %p"))
self.svFechaHistoria.set(xFecha+xHora)

##### BUTTONS-F2 #####

self.btnAgregarHistoria = tk.Button(self.frameFechaHistoria, text="Agregar", command=self.agregaHistorialMedico)
self.btnAgregarHistoria.config(width=20, font=("Verdana", 12, "bold"), foreground="ghostwhite", background="lightsalmon", cursor="hand2", acti
self.btnAgregarHistoria.grid(row=2, column=0, padx=10, pady=5)

self.btnSalirAgregarHistoria = tk.Button(self.frameFechaHistoria, text="Salir", command=self.topAHistoria.destroy)
self.btnSalirAgregarHistoria.config(width=20, font=("Verdana", 12, "bold"), foreground="ghostwhite", background="gray6", cursor="hand2", acti
self.btnSalirAgregarHistoria.grid(row=2, column=3, padx=10, pady=5)

self.idPersona = None
```

• ANEXO 37:

```
def agregaHistorialMedico(self):
    """Función que agrega los historiales"""

    try:
        if self.idHistoriaMedica == None:
            guardarHistoria(self.idPersonaHistoria, self.svFechaHistoria.get(), self.svMotivo.get(), self.svOperacion.get(), self
            self.topAHistoria.destroy()
            self.topHistoriaMedica.destroy()
            self.historiaMedica()
            self.idPersona = None

    except:
        titulo = "Agregar Historia"
        mensaje = "Error al agregar historia Medica"
        messagebox.showerror(titulo, mensaje)
```

- **ANEXO 38:**

```
def eliminarHistorialMedico(self):  
    """función que elimina los historiales, de forma definitiva"""  
  
    try:  
        self.idHistoriaMedica = self.tabla2.item(self.tabla2.selection())['text']  
        eliminarHistoria(self.idHistoriaMedica)  
        self.topHistoriaMedica.destroy()  
        self.historiaMedica()  
        self.idHistoriaMedica = None  
  
    except:  
        titulo = "Eliminar Historia"  
        mensaje = "Error al eliminar"  
        messagebox.showerror(titulo, mensaje)
```

- **ANEXO 39:**

```
def topEditarHistorialMedico(self):  
  
    try:  
        self.idHistoriaMedica = self.tabla2.item(self.tabla2.selection())["text"]  
        self.fechaHistoriaEditar = str(datetime.date.today().strftime("%d/%m/%Y, ") + datetime.datetime.now().strftime("%I:%M %p"))  
        self.motivoHistoriaEditar = self.tabla2.item(self.tabla2.selection())["values"][2]  
        self.operacionHistoriaEditar = self.tabla2.item(self.tabla2.selection())["values"][3]  
        self.tratamientoHistoriaEditar = self.tabla2.item(self.tabla2.selection())["values"][4]  
        self.detalleHistoriaEditar = self.tabla2.item(self.tabla2.selection())["values"][5]  
  
        self.topEditarHistoria = tk.Toplevel()  
        self.topEditarHistoria.geometry("910x450+300+350")  
        self.topEditarHistoria.title("EDITAR HISTORIA MEDICA")  
        self.topEditarHistoria.resizable(width=False, height=False)  
        self.topEditarHistoria.iconbitmap("ICONOS/ICONO.ico")  
        self.topEditarHistoria.config(background="palegreen")  
  
        #FRAME EDITAR DATOS HISTORIA  
        self.frameEditarHistoria = tk.LabelFrame(self.topEditarHistoria)  
        self.frameEditarHistoria.config(background="palegreen")  
        self.frameEditarHistoria.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)  
  
        #LABEL EDITAR HISTORIA  
  
        self.lblMotivoEditar = tk.Label(self.frameEditarHistoria, text="Motivo de la visita", width=30, font=("Verdana", 15,"bold"), background="palegreen")  
        self.lblMotivoEditar.grid(row=0, column=0, padx=5, pady=3)  
  
        self.lblOperacionEditar = tk.Label(self.frameEditarHistoria, text="Operación", width=30, font=("Verdana", 15,"bold"), background="palegreen")  
        self.lblOperacionEditar.grid(row=2, column=0, padx=5, pady=3)
```



```

self.lblTratamientoEditar = tk.Label(self.frameEditarHistoria, text="Tratamiento", width=30, font=("Verdana", 15,"bold"), background="palegreen")
self.lblTratamientoEditar.grid(row=4, column=0, padx=5, pady=3)

self.lblDetalleEditar = tk.Label(self.frameEditarHistoria, text="Detalle adicional", width=30, font=("Verdana", 15,"bold"), background="palegreen")
self.lblDetalleEditar.grid(row=6, column=0, padx=5, pady=3)

#ENTRYS EDITAR HISTORIA

self.svMotivoEditar = tk.StringVar()
self.entryMotivoEditar = tk.Entry(self.frameEditarHistoria, textvariable=self.svMotivoEditar)
self.entryMotivoEditar.config(width=65, font=("Verdana", 15))
self.entryMotivoEditar.grid(row = 1, column=0, pady=3, padx=5, columnspan=2)

self.svOperacionEditar = tk.StringVar()
self.entryOperacionEditar = tk.Entry(self.frameEditarHistoria, textvariable=self.svOperacionEditar)
self.entryOperacionEditar.config(width=65, font=("Verdana", 15))
self.entryOperacionEditar.grid(row = 3, column=0, pady=3, padx=5, columnspan=2)

self.svTratamientoEditar = tk.StringVar()
self.entryTratamientoEditar = tk.Entry(self.frameEditarHistoria, textvariable=self.svTratamientoEditar)
self.entryTratamientoEditar.config(width=65, font=("Verdana", 15))
self.entryTratamientoEditar.grid(row = 5, column=0, pady=3, padx=5, columnspan=2)

self.svDetalleEditar = tk.StringVar()
self.entryDetalleEditar = tk.Entry(self.frameEditarHistoria, textvariable=self.svDetalleEditar)
self.entryDetalleEditar.config(width=65, font=("Verdana", 15))
self.entryDetalleEditar.grid(row = 7, column=0, pady=3, padx=5, columnspan=2)

#FRAME FECHA EDITAR
self.frameFechaEditar = tk.LabelFrame(self.topEditarHistoria)
self.frameFechaEditar.config(background="palegreen")
self.frameFechaEditar.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

#LABEL FECHA EDITAR
self.lblFechaHistoriaEditar = tk.Label(self.frameFechaEditar, text="Fecha y Hora", width=30, font=("Verdana", 15,"bold"), background="palegreen")
self.lblFechaHistoriaEditar.grid(row=0, column=0, padx=5, pady=3)

# ENTRY FECHA EDITAR
self.svFechaHistoriaEditar = tk.StringVar()
self.entryFechaHistoriaEditar = tk.Entry(self.frameFechaEditar, textvariable=self.svFechaHistoriaEditar)
self.entryFechaHistoriaEditar.config(width=20, font=("Verdana", 15))
self.entryFechaHistoriaEditar.grid(row = 0, column=1, pady=3, padx=5)

#INSERTAR LOS VALORES A LOS ENTRYS

self.entryMotivoEditar.insert(0, self.motivoHistoriaEditar) # 0 es la locación
self.entryOperacionEditar.insert(0, self.operacionHistoriaEditar)
self.entryTratamientoEditar.insert(0, self.tratamientoHistoriaEditar)
self.entryDetalleEditar.insert(0, self.detalleHistoriaEditar)
self.entryFechaHistoriaEditar.insert(0, self.fechaHistoriaEditar)

#BUTTON EDITAR HISTORIA
self.btnEditarHistoriaMedica = tk.Button(self.frameFechaEditar, text="Editar Historia", command = self.historiaMedicaEditar)
self.btnEditarHistoriaMedica.config(width=20, font=("Verdana", 12,"bold"), foreground="snow", background="royalblue2", cursor="hand2")
self.btnEditarHistoriaMedica.grid(row=1, column=0, padx=10, pady=5)

self.btnSalirEditarHistoriaMedica = tk.Button(self.frameFechaEditar, text="Salir", command=self.topEditarHistoria.destroy)
self.btnSalirEditarHistoriaMedica.config(width=20, font=("Verdana", 12,"bold"), foreground="gray99", background="gray25", cursor="hand2")
self.btnSalirEditarHistoriaMedica.grid(row=1, column=1, padx=10, pady=5)

if self.idHistoriaMedicaEditar == None:
    self.idHistoriaMedicaEditar = self.idHistoriaMedica

self.idHistoriaMedica = None

except:
    titulo = "Editar Historia"
    mensaje = "Error al editar historia"
    messagebox.showerror(titulo, mensaje)

```

- **ANEXO 40:**

```
def historiaMedicaEditar(self):  
  
    try:  
        editarHistoria([self.svFechaHistoriaEditar.get(), self.svMotivoEditar.get(), self.svOperacionEditar.get(),  
                        | self.svTratamientoEditar.get(), self.svDetalleEditar.get(), self.idHistoriaMedicaEditar])  
        self.idHistoriaMedicaEditar = None  
        self.idHistoriaMedica = None  
        self.topEditarHistoria.destroy()  
        self.topHistoriaMedica.destroy()  
        self.historiaMedica()  
  
    except:  
        titulo = "Editar Historia"  
        mensaje = "Error al editar historia"  
        messagebox.showerror(titulo, mensaje)  
        self.topEditarHistoria.destroy()
```

- **ANEXO 41:**

```
def error():  
  
    AppError = tk.Tk()  
    AppError.title("¡Vaya!")  
    AppError.geometry("350x200+600+200") |  
    AppError.iconbitmap("ICONOS/error2.ico")  
    AppError.resizable(width=False, height=False)  
    AppError.configure(background="steelblue1")  
  
    LbError = tk.Label(AppError,  
                      text="HA OCURRIDO UN ERROR",  
                      font=("Arial", 18, "bold"),  
                      foreground="red2",  
                      background="steelblue1"  
    )  
    LbError.place(x=25, y=60)  
  
    LbError2 = tk.Label(AppError,  
                      text="Vuelva a ejecutar la aplicación",  
                      font=("Arial", 12),  
                      foreground="gray30",  
                      background="steelblue1"  
    )  
    LbError2.place(x=74, y=97)  
  
    AppError.mainloop()
```

- **ANEXO 42:**

- Link del repositorio:

https://github.com/hdjdujixjy/MecatronicaUNT_Prog1_DisenodeUnSistemaDeControlParaHistorialesMedicos.git

- Link de la carpeta drive:

<https://drive.google.com/drive/folders/1JX5QQ2yP3X0bsZH21BrMSycBMdzGrBNI?usp=sharing>