GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Traspaso de una BBDD en formato Excel a formato SQL y crear una interfaz para interactuar con la BBDD

Autor: Isidro Carrara Bittini

Director: Marc Baulenas Mir

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

TRANSFORMACIÓN DE BBDD(EXCEL) A BBDD(SQL) Y CREACIÓN DE INTERFAZ PARA INTERACTUAR CON DICHA BBDD

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/2023 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Isidro Carrara Bittini Fecha: 25 / 07 / 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Marc Baulenas Mir Fecha: 26 / 07 / 2023

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Traspaso de una BBDD en formato Excel a formato SQL y crear una interfaz para interactuar con la BBDD

Autor: Isidro Carrara Bittini

Director: Marc Baulenas Mir

Traspaso de una BBDD en formato Excel a formato SQL y crear una interfaz para interactuar con la BBDD

Autor: Carrara Bittini, Isidro
Director: Baulenas Mir, Marc
Entidad Colaboradora: Minsait

RESUMEN DEL PROYECTO

Las bases de datos (BBDD) suelen percibirse como complejas, difícil de entender cómo funcionan, de crearlas e interactuar con ellas. Este es uno de los motivos por lo que al final se acaba usando Excel para crear BBDD, que no es la forma más eficiente ya que como seres humanos podemos olvidarnos de actualizar campos o errores por el estilo. El TFM consiste en hacer una transformación de una BBDD en formato Excel a un formato SQL, para posteriormente crear un interfaz que interactúe con la BBDD. Se usará SQL porque es la forma más eficiente de guardar datos y se puede implementar en muchas herramientas.

La BBDD de Excel ya está creada, por lo que se explicara como cambiar un formato de Excel a formato SQL y se usara también el lenguaje de programación de Python para poder facilitar algunas tareas como traspasar la información.

1. Introducción

En la era digital actual, los datos son el corazón que impulsa la toma de decisiones efectiva en diversas áreas, desde la industria hasta la investigación y la gestión empresarial. Con el crecimiento exponencial de la información, surge la necesidad de transformar y almacenar datos de manera eficiente y segura. En este contexto, el presente Trabajo de Fin de Máster (TFM) se enfoca en abordar un desafío fundamental: la transformación de una base de datos en hoja de cálculo a un formato SQL y el desarrollo de una interfaz interactiva que permita interactuar de manera efectiva con dicha base de datos.

Las hojas de cálculo, como Excel, han sido ampliamente utilizadas para almacenar y manipular datos en diversas disciplinas debido a su facilidad de uso y versatilidad. Sin embargo, a medida que las cantidades de datos aumentan, las hojas de cálculo pueden volverse limitadas en términos de escalabilidad, rendimiento y seguridad. Por otro lado, SQL (Structured Query Language) ha demostrado ser una solución robusta y eficiente para la gestión de bases de datos, permitiendo una estructura organizada y consultas poderosas que facilitan el acceso y análisis de la información.

El objetivo principal de este trabajo es explorar y presentar un enfoque detallado para migrar una base de datos en hoja de cálculo a un formato SQL, abordando los desafíos inherentes a esta transformación y aprovechando las ventajas de SQL para asegurar un almacenamiento de datos óptimo y una gestión más eficiente. Asimismo, se pretende desarrollar una interfaz interactiva y amigable que brinde a los usuarios la capacidad de interactuar con la base de datos de manera intuitiva, permitiendo consultas personalizadas, visualización de resultados y actualización de datos en tiempo real.

En este trabajo, se describirán en detalle las tecnologías y herramientas utilizadas para llevar a cabo la transformación de la base de datos en hoja de cálculo a SQL,

abordando aspectos fundamentales como el diseño de la estructura de la base de datos, la validación y limpieza de datos, así como la optimización del rendimiento. Además, se presentará el proceso de desarrollo de la interfaz, destacando las características implementadas para garantizar una experiencia de usuario fluida y eficiente.

Con esta investigación, se busca contribuir al conocimiento en el ámbito de la gestión de datos y brindar una solución práctica y aplicable para aquellos que enfrentan el desafío de transformar bases de datos en hojas de cálculo a formatos SQL y desean potenciar la interacción con sus datos. Asimismo, se aspira a enriquecer la comprensión de cómo la convergencia entre tecnologías puede mejorar la eficiencia y eficacia en la gestión y análisis de información, allanando el camino hacia una toma de decisiones más informada y precisa en diversos ámbitos profesionales.

2. Definición del proyecto

El Objetivo que tiene este proyecto es poder dar la posibilidad de que la base de datos pueda ser más grande. Excel te permite tener registros, pero a más registros más lento y peor funciona, si es cierto que Excel tiene herramientas graficas que una base de datos o no. Esto para el simple hecho de poder tener lo registros y poder manipularlos son herramientas que no son decisivas a la hora de elegir donde hacerlo. SQL nos permite tener muchos más registros y poder manipularlos de mejor forma y con una rapidez superior a la de Excel.

Una vez creada la base de datos en la que poder tener los registros del cliente es necesario saber SQL para poder interactuar con dicha base. Aparte de hacer el traspaso de información de los registros en Excel a base SQL, se ha decido crear una interfaz que nos permita hacer las tareas básicas y necesarias para poder interactuar. Tareas como actualizar e insertar nevos registros, y aceptar o rechazar leads (que es actualizar datos).

Para resumir el objetivo es hacer un traspaso de unos registros de unos clientes en formato Excel a un formato SQL, que nos permite tener mucha más información y poder manipular esa información de mejor manera. Como segundo objetivo, cera una interfaz para que tanto los partners como nosotros como consultora podamos actualizar datos en la base de datos sin tener que saber SQL. Se creará dos interfaces una para Minsait y otra para cada partner, en este proyecto solo se hará la simulación para un partner, para el resto de partners seria literal solo cobrandizar la aplicación y dar una clave de acceso a cada uno.

3. Descripción del modelo/herramienta

Para el Trabajo Final de Máster (TFM) que tiene como objetivo realizar un traspaso de registros de clientes desde un formato Excel a una base de datos SQL, así como crear una interfaz que permita interactuar con la base de datos sin necesidad de conocer SQL, se pueden utilizar diferentes herramientas y tecnologías para llevar a cabo este proyecto. Aquí hay una propuesta de modelo y herramientas que podrían ser adecuadas para el desarrollo del proyecto:

1. Modelo Propuesto:

- Desarrollo basado en arquitectura cliente-servidor: Donde la base de datos SQL será el servidor y la interfaz de usuario será el cliente que interactúa con la base de datos.

- Uso de un lenguaje de programación para la interfaz de usuario: Para crear la interfaz gráfica que permita a los usuarios realizar las tareas básicas de actualización e inserción de registros, es recomendable utilizar un lenguaje de programación adecuado para interfaces de usuario, como Python, Java, C#, o cualquier otro lenguaje que el equipo esté familiarizado y permita el desarrollo de aplicaciones con interfaces gráficas.

- Uso de una biblioteca o framework para la interfaz gráfica: Dependiendo del lenguaje de programación elegido, existen diversas bibliotecas y frameworks que facilitan la creación de interfaces gráficas, como PyQt, Tkinter para Python, JavaFX para Java, Windows Forms para C#, entre otros.

- Base de datos SQL: Se utilizará una base de datos SQL (por ejemplo, MySQL, PostgreSQL o Microsoft SQL Server) para almacenar los registros de los clientes. Estas bases de datos ofrecen un rendimiento superior y capacidad para manejar grandes volúmenes de datos en comparación con Excel.

- Herramientas de migración de datos: Para facilitar el traspaso de los registros desde el formato Excel a la base de datos SQL, se pueden utilizar herramientas de migración de datos, como DBeaver, MySQL Workbench o herramientas similares, que permiten importar datos desde archivos Excel a la base de datos SQL de forma eficiente.

2. Desarrollo de la interfaz:

- Para la creación de la interfaz, se pueden seguir metodologías ágiles de desarrollo de software, como Scrum o Kanban, para dividir el proyecto en tareas más manejables y permitir una iteración rápida con los usuarios y posibles stakeholders.

- Crear la interfaz de usuario con las funciones necesarias para actualizar e insertar nuevos registros en la base de datos.

- Implementar mecanismos de autenticación y autorización para controlar el acceso y las acciones permitidas en la base de datos por parte de los usuarios y partners.

- Diseñar la interfaz con una apariencia atractiva y amigable para el usuario, teniendo en cuenta la experiencia de usuario (UX) para facilitar la interacción con la aplicación.

3. Consideraciones adicionales:

- Asegurar la seguridad de la base de datos y la información almacenada mediante el uso de técnicas de encriptación, autenticación segura y auditoría de actividades.

- Realizar pruebas exhaustivas para garantizar la integridad y funcionalidad de la aplicación antes de su implementación.

- Documentar adecuadamente el proceso de desarrollo, el diseño de la base de datos, la estructura de la interfaz y cualquier otra información relevante para facilitar futuras actualizaciones y mantenimiento.

4. Resultados

- Se ha llegado al resultado esperado debido que se ha creado una interfaz que nos permite interactuar con la base de datos sin saber SQL.
- Efectivamente es un sistema mucho más eficiente donde la actualización de los datos solo ocurre en una vez y hay menos errores.
- El servicio al cliente es mucho más ágil ya que la información es muchas más rápida.
- Mas participación de los partners debido a la que la información es más completa y de más fácil acceso.
- Se ha conseguido traspasar toda información a la nueva base de datos sin ningún error y sin ningún registro perdido.

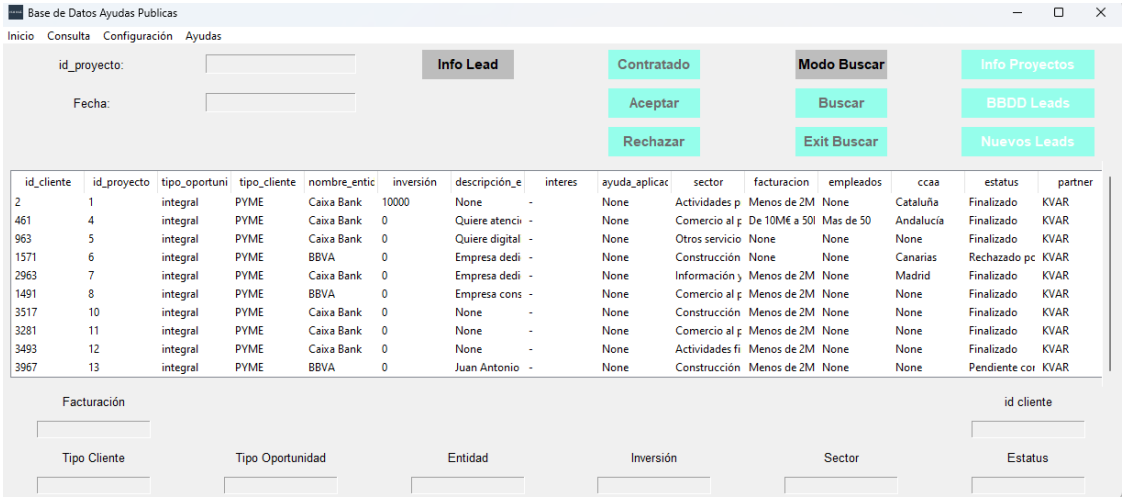


Ilustración 1 – Interfaz para los partners

5. Conclusiones

Este Trabajo de Fin de Máster ha abordado de manera integral y detallada la transformación de una base de datos en hoja de cálculo a un formato SQL, junto con el desarrollo de una interfaz interactiva que permite una interacción efectiva con dicha base de datos. A través de una metodología sólida y una cuidadosa planificación, se ha logrado llevar a cabo este proceso de migración con éxito, obteniendo resultados significativos y de gran utilidad en el ámbito de la gestión y análisis de datos.

La transformación de la base de datos en hoja de cálculo a SQL ha demostrado ser una decisión acertada, ya que ha permitido aprovechar las ventajas de una estructura

organizada y eficiente para el almacenamiento y gestión de los datos. Con la capacidad de utilizar consultas poderosas y realizar análisis más complejos, se ha mejorado la eficiencia y precisión en el tratamiento de la información.

6. Referencias

TRANSFER OF A DATABASE IN EXCEL FORMAT TO SQL FORMAT AND CREATE AN INTERFACE TO INTERACT WITH THE DATABASE.

Author: Carrara Bittini, Isidro
Supervisor: Baulenas Mir, Marc
Collaborating Entity: Minsait

ABSTRACT

Databases are often perceived as complex, difficult to understand how they work, create, and interact with. This is one of the reasons why Excel is used to create databases in the end, which is not the most efficient way as we as humans can forget to update fields or make errors like that. The TFM consists of transforming a database in Excel format into a SQL format, and then creating an interface that interacts with the database. SQL will be used because it is the most efficient way to store data and can be implemented in many tools.

The Excel database is already created, so it will explain how to change from an Excel format to a SQL format, and the Python programming language will also be used to facilitate some tasks such as transferring information.

1. Introducción (traducir todo el resumen al inglés, incluyendo los títulos que se hayan empleado)

In the current digital era, data is the heart driving effective decision-making in various fields, from industry to research and business management. With the exponential growth of information, there arises a need to efficiently and securely transform and store data. In this context, this Master's Thesis (TFM) focuses on addressing a fundamental challenge: the transformation of a spreadsheet-based database into a SQL format, along with the development of an interactive interface to effectively interact with the said database.

Spreadsheets like Excel have been widely used to store and manipulate data across disciplines due to their ease of use and versatility. However, as data volumes increase, spreadsheets can become limited in terms of scalability, performance, and security. On the other hand, SQL (Structured Query Language) has proven to be a robust and efficient solution for database management, offering an organized structure and powerful queries that facilitate data access and analysis.

The main objective of this work is to explore and present a detailed approach to migrate a spreadsheet-based database to a SQL format, addressing the inherent challenges of this transformation and leveraging the advantages of SQL to ensure optimal data storage and more efficient management. Additionally, the aim is to develop an interactive and user-friendly interface that empowers users to interact with the database intuitively, allowing custom queries, result visualization, and real-time data updates.

This work will describe in detail the technologies and tools used to carry out the transformation from a spreadsheet-based database to SQL, addressing fundamental aspects such as database structure design, data validation, cleansing, and performance optimization. Furthermore, the development process of the interface will be presented, highlighting the implemented features to ensure a smooth and efficient user experience.

Through this research, the goal is to contribute to the knowledge in the field of data management and provide a practical and applicable solution for those facing the challenge of transforming spreadsheet-based databases into SQL formats, seeking to enhance data interaction. Moreover, it aspires to enrich the understanding of how the convergence of technologies can improve efficiency and effectiveness in information management and analysis, paving the way towards more informed and precise decision-making in various professional domains.

2. Definición del Proyecto

The objective of this project is to provide the possibility of scaling the database. While Excel allows you to have records, the more records you have, the slower and less efficient it becomes. While it is true that Excel has graphical tools that a database may lack, these tools are not decisive when choosing where to store the records and manipulate them. SQL, on the other hand, allows us to handle many more records efficiently and with superior speed compared to Excel.

Once the database is created to store client records, it becomes necessary to know SQL to interact with the database. In addition to transferring information from Excel records to the SQL database, it has been decided to create an interface that allows performing basic and necessary tasks for interaction. These tasks include updating and inserting new records, as well as accepting or rejecting leads (which involves updating data).

In summary, the main goal is to transfer client records from Excel format to SQL format, which enables us to handle much more information and manipulate it more effectively. As a secondary objective, an interface will be developed to allow both partners and our consulting team to update data in the database without needing to know SQL. Two interfaces will be created, one for Minsait and another for each partner. However, in this project, the simulation will be done for one partner only. For the rest of the partners, it would simply involve customizing the application and providing each one with a unique access key.

3. Descripción del modelo/sistema/herramienta

For the Master's Final Project (TFM) that aims to transfer customer records from an Excel format to a SQL database and create an interface for interacting with the database without requiring knowledge of SQL, different tools and technologies can be used to carry out this project. Here is a proposed model and tools that could be suitable for the project's development:

1. Proposed Model:

- Development based on a client-server architecture: Where the SQL database will act as the server, and the user interface will be the client that interacts with the database.

- Use of a programming language for the user interface: To create the graphical interface allowing users to perform basic tasks like updating and inserting records, it is recommended to use a suitable programming language for user interfaces, such as Python, Java, C#, or any other language the team is familiar with that supports developing applications with graphical interfaces.

- Use of a library or framework for the graphical interface: Depending on the chosen programming language, there are various libraries and frameworks that facilitate the creation of graphical interfaces, such as PyQt, Tkinter for Python, JavaFX for Java, Windows Forms for C#, among others.

- SQL database: A SQL database (e.g., MySQL, PostgreSQL, or Microsoft SQL Server) will be used to store customer records. These databases offer superior performance and the capacity to handle large volumes of data compared to Excel.

- Data migration tools: To facilitate the transfer of records from the Excel format to the SQL database, data migration tools like DBeaver, MySQL Workbench, or similar tools can be used, enabling efficient data importing from Excel files to the SQL database.

2. Interface Development:

    - For creating the interface, agile software development methodologies like Scrum or Kanban can be followed to break the project into manageable tasks and enable quick iteration with users and stakeholders.

    - Design the user interface with the necessary functions to update and insert new records into the database.

    - Implement authentication and authorization mechanisms to control access and actions allowed in the database by users and partners.

    - Design the interface with an appealing and user-friendly appearance, considering the user experience (UX) to facilitate interaction with the application.

3. Additional Considerations:

    - Ensure the security of the database and stored information by using encryption techniques, secure authentication, and activity auditing.

    - Conduct thorough testing to guarantee the integrity and functionality of the application before its implementation.

    - Properly document the development process, database design, interface structure, and any other relevant information to facilitate future updates and maintenance.

4. Resultados

- The expected result has been achieved because an interface has been created that allows us to interact with the database without knowing SQL.
- Indeed, it is a much more efficient system where data updates occur only once, reducing errors significantly.
- Customer service has become much more agile as information retrieval is much faster.
- There is increased participation from partners due to the information being more comprehensive and easily accessible.
- All the information has been successfully transferred to the new database without any errors or lost records.

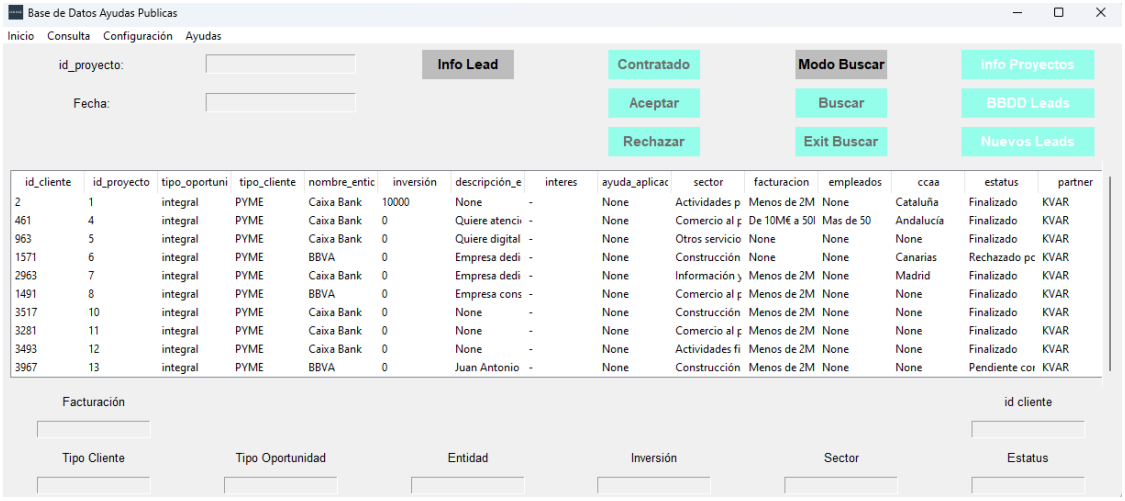


Ilustración 2 - Simulación del bucle completo de la etapa de frecuencias

5. Conclusiones

This Master's Thesis has comprehensively and in detail addressed the transformation of a spreadsheet-based database into an SQL format, along with the development of an interactive interface that enables effective interaction with the said database. Through a robust methodology and careful planning, this migration process has been successfully carried out, yielding significant and valuable results in the field of data management and analysis.

The transformation of the spreadsheet-based database to SQL has proven to be a wise decision, as it has allowed us to leverage the advantages of an organized and efficient structure for data storage and management. With the ability to use powerful queries and perform more complex analyses, efficiency and precision in handling the information have been improved.

6. Referencias

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En la era digital actual, los datos son el corazón que impulsa la toma de decisiones efectiva en diversas áreas, desde la industria hasta la investigación y la gestión empresarial. Con el crecimiento exponencial de la información, surge la necesidad de transformar y almacenar datos de manera eficiente y segura. En este contexto, el presente Trabajo de Fin de Máster (TFM) se enfoca en abordar un desafío fundamental: la transformación de una base de datos en hoja de cálculo a un formato SQL y el desarrollo de una interfaz interactiva que permita interactuar de manera efectiva con dicha base de datos.

Las hojas de cálculo, como Excel, han sido ampliamente utilizadas para almacenar y manipular datos en diversas disciplinas debido a su facilidad de uso y versatilidad. Sin embargo, a medida que las cantidades de datos aumentan, las hojas de cálculo pueden volverse limitadas en términos de escalabilidad, rendimiento y seguridad. Por otro lado, SQL (Structured Query Language) ha demostrado ser una solución robusta y eficiente para la gestión de bases de datos, permitiendo una estructura organizada y consultas poderosas que facilitan el acceso y análisis de la información.

El objetivo principal de este trabajo es explorar y presentar un enfoque detallado para migrar una base de datos en hoja de cálculo a un formato SQL, abordando los desafíos inherentes a esta transformación y aprovechando las ventajas de SQL para asegurar un almacenamiento de datos óptimo y una gestión más eficiente. Asimismo, se pretende desarrollar una interfaz interactiva y amigable que brinde a los usuarios la capacidad de interactuar con la base de datos de manera intuitiva, permitiendo consultas personalizadas, visualización de resultados y actualización de datos en tiempo real.

En este trabajo, se describirán en detalle las tecnologías y herramientas utilizadas para llevar a cabo la transformación de la base de datos en hoja de cálculo a SQL, abordando aspectos fundamentales como el diseño de la estructura de la base de datos, la validación y limpieza de datos, así como la optimización del rendimiento. Además, se presentará el

proceso de desarrollo de la interfaz, destacando las características implementadas para garantizar una experiencia de usuario fluida y eficiente.

Con esta investigación, se busca contribuir al conocimiento en el ámbito de la gestión de datos y brindar una solución práctica y aplicable para aquellos que enfrentan el desafío de transformar bases de datos en hojas de cálculo a formatos SQL y desean potenciar la interacción con sus datos. Asimismo, se aspira a enriquecer la comprensión de cómo la convergencia entre tecnologías puede mejorar la eficiencia y eficacia en la gestión y análisis de información, allanando el camino hacia una toma de decisiones más informada y precisa en diversos ámbitos profesionales.

1.1 MOTIVACIÓN DEL PROYECTO

La motivación para llevar a cabo este Trabajo de Fin de Máster (TFM) puede derivar de varias razones y objetivos personales y profesionales. A continuación, se presentan algunas posibles motivaciones:

1. Interés en la gestión y análisis de datos: Si el estudiante siente pasión por el mundo de los datos y su impacto en la toma de decisiones, realizar este TFM puede brindarle la oportunidad de profundizar en el campo de la gestión de bases de datos y aprender sobre herramientas avanzadas como SQL.
2. Necesidad en el ámbito laboral: En algunos casos, el estudiante puede encontrarse trabajando en una empresa o institución donde actualmente se manejan bases de datos en hojas de cálculo, pero se requiere una solución más escalable y eficiente. El TFM podría ser una forma de abordar esta necesidad y proponer una solución aplicable en su entorno laboral.
3. Importancia de la transformación digital: Con el avance de la transformación digital en todos los sectores, es fundamental adaptarse a las nuevas tecnologías y metodologías. Realizar este TFM puede ser una manera de adquirir habilidades relevantes para la actualización tecnológica en el área de gestión de datos.

4. Explorar nuevas tecnologías y herramientas: Si el estudiante está interesado en expandir sus conocimientos sobre bases de datos y lenguajes como SQL, el TFM puede ser la oportunidad ideal para investigar y aplicar estas tecnologías en un contexto práctico.
5. Impacto en la organización: Algunos estudiantes pueden estar motivados por la idea de que su trabajo académico pueda tener un impacto real y positivo en la organización en la que trabajan o en el ámbito profesional al que aspiran.
6. Desarrollo de habilidades técnicas: La realización de este TFM puede ser una excelente oportunidad para mejorar habilidades técnicas en programación, diseño de bases de datos, análisis de datos y desarrollo de interfaces, entre otras
7. Potencial para futuros proyectos: La experiencia adquirida en el TFM puede sentar las bases para futuros proyectos en el campo de la gestión de datos y el desarrollo de aplicaciones interactivas.
8. Contribución al conocimiento: Algunos estudiantes pueden sentir la motivación de aportar nuevos conocimientos al campo de la informática y la gestión de datos mediante la investigación y la implementación práctica.

En las siguientes secciones de este TFM, se detallará la metodología utilizada, se presentarán las tecnologías empleadas y se expondrán los resultados obtenidos, aportando así una contribución significativa en el ámbito de la gestión de bases de datos y la interacción con la información.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Las hojas de cálculo, como Excel, han sido ampliamente utilizadas para almacenar y manipular datos en diversas disciplinas debido a su facilidad de uso y versatilidad. Sin embargo, a medida que las cantidades de datos aumentan, las hojas de cálculo pueden volverse limitadas en términos de escalabilidad, rendimiento y seguridad. Por otro lado, SQL (Structured Query Language) ha demostrado ser una solución robusta y eficiente para la gestión de bases de datos, permitiendo una estructura organizada y consultas poderosas que facilitan el acceso y análisis de la información. Para poder llevar acabo este proyecto es necesario tener un mejor entendimiento de las tecnologías con la que se van a llevar a cabo y las tecnologías que se usaban anteriormente.

En la sección a continuación se explicará por encima en que consiste el lenguaje o la tecnología y cuál es su funcionamiento. También se explicará la librearías usadas y para que se utilizaran en el proyecto.

2.1 HOJAS DE CÁLCULO EXCEL

Una hoja de cálculo Excel es una herramienta de software desarrollada por Microsoft que permite a los usuarios crear, organizar y manipular datos numéricos en forma de tablas. Es una aplicación ampliamente utilizada en diversas áreas como negocios, contabilidad, finanzas, ciencia, ingeniería, entre otros, debido a su facilidad de uso y versatilidad.

Con Excel, los usuarios pueden realizar operaciones matemáticas, fórmulas complejas, análisis estadísticos, gráficos y tablas dinámicas, lo que facilita la toma de decisiones informadas a partir de la información numérica presentada en la hoja de cálculo.[1]

2.1.1 ESTRUCTURA

Las hojas de cálculo tienen una estructura muy simple de entender, pero con ella se puede hacer cosas muy complejas. Las hojas de cálculo tienen varias hojas que se componen de

filas y columnas, cada hoja es independiente al resto de ellas, pero se puede elegir tener datos relacionados entre ellos si se quiere. Dentro de cada hoja se pueden hacer diversas acciones como crear tablas, graficas o realizar operaciones, como sumar o hacer la desviación típica.

Al tener varias hojas podemos hacer registros diferentes en cada una de ellas, o incluso podemos destinar una hoja para cada tarea requerida. Por ejemplo, una hoja puede ser para registrar datos, otra para hacer operaciones y otra más para hacer representaciones graficas. Esto es lo que hace que las hojas de cálculo sean herramientas muy potentes y con mucha versatilidad, con capacidad de realizar muchas tareas.

2.1.2 HERRAMIENTAS

Como se acaba de mencionar la herramienta de hoja de cálculo nos permite hacer muchas cosas mediante las herramientas que están incorporadas. Entre ellas se encuentran las herramientas básicas de crear tablas, gráficas y formulas. Hay herramientas más avanzadas como crear macros (sirve para hacer una tarea y guardarla en un comando para poder repetirla sin hacerla paso a paso). Otra herramienta mas avanzada en VBA, que es una herramienta que nos permite hacer programas y scrips para agilizar las tareas, para poder usar esta herramienta hay que aprender el lenguaje de programación que se usa para ellas.

Algunas herramientas que destacar de esta tecnología son:[1]

1. Cálculos automáticos: Las hojas de cálculo permiten realizar cálculos automáticos mediante fórmulas y funciones. Esto agiliza tareas repetitivas y complejas, como sumas, restas, promedios, entre otras operaciones matemáticas.
2. Gráficos y visualización de datos: La tecnología de hojas de cálculo ofrece la posibilidad de crear gráficos y tablas dinámicas para visualizar los datos de manera más clara y comprensible, lo que facilita la toma de decisiones basadas en la información presentada.

3. Análisis estadístico: Las hojas de cálculo incluyen una amplia gama de funciones estadísticas que permiten analizar datos numéricos y extraer conclusiones relevantes, como desviación estándar, regresión lineal y análisis de tendencias.
4. Automatización: Mediante el uso de macros y scripts, es posible automatizar tareas repetitivas y complejas en las hojas de cálculo, lo que aumenta la eficiencia y reduce el tiempo de trabajo.
5. Colaboración: Las hojas de cálculo pueden ser compartidas y colaboradas por múltiples usuarios simultáneamente, lo que facilita el trabajo en equipo y la revisión conjunta de datos.
6. Modelado financiero: Las hojas de cálculo son ampliamente utilizadas en el ámbito financiero para realizar presupuestos, proyecciones y análisis de rentabilidad.
7. Escalabilidad: Las hojas de cálculo pueden manejar grandes cantidades de datos y se pueden utilizar para crear bases de datos y sistemas de gestión más complejos.

2.1.3 HERRAMIENTAS USADAS EN EL PROYECTO

Las herramientas que se usan en este proyecto son las herramientas más básicas, tablas y formulas que nos permiten relacionar los datos de unas hojas a otras. Hay que tener en cuenta que el objetivo de esta hoja de calculo no es mas que mantener un registro de los clientes y poder actualizar el estado de ellos. También se ha usado herramienta como el SharePoint, es una hoja de calculo que se puede compartir para que los partners puedan acceder y actualizar la información de los clientes.

2.2 PYTHON

Python es un lenguaje de programación de alto nivel, interpretado, de propósito general y con una sintaxis sencilla y legible. Fue creado en la década de 1990 por Guido van Rossum y desde entonces ha ganado una gran popularidad debido a su facilidad de aprendizaje, flexibilidad y amplia comunidad de desarrolladores.

Al ser un lenguaje de propósito general, Python se puede utilizar para una amplia variedad de tareas, como desarrollo web, análisis de datos, inteligencia artificial, automatización, desarrollo de juegos, entre otras aplicaciones.

Una de las principales ventajas de Python es su extensa biblioteca estándar y su ecosistema de librerías de terceros, que brindan una amplia gama de funcionalidades y capacidades adicionales. Algunas de las librerías más populares de Python son:

- Pandas: Esta librería proporciona estructuras de datos flexibles y herramientas para el análisis y manipulación de datos. Es especialmente útil para trabajar con datos tabulares, como los que se encuentran en hojas de cálculo o bases de datos, y permite realizar tareas como filtrado, limpieza, agregación y visualización de datos.
- Tkinter: La librería tkinter es una parte estándar de la biblioteca estándar de Python que se utiliza para crear interfaces gráficas de usuario (GUI, por sus siglas en inglés). Permite a los desarrolladores crear ventanas, cuadros de diálogo, botones, cuadros de texto, etiquetas, menús, barras de desplazamiento y otros componentes interactivos para sus aplicaciones.
- Mysql.connector: es un módulo de Python que proporciona una interfaz para conectarse y manipular bases de datos MySQL desde aplicaciones escritas en Python. Es una librería que facilita la interacción con bases de datos MySQL mediante sentencias SQL y permite realizar operaciones como consultas, inserciones, actualizaciones y eliminaciones de datos.

Estas son solo algunas de las muchas librerías disponibles en Python. La versatilidad y diversidad de su ecosistema hacen que Python sea una excelente elección para una amplia gama de proyectos y aplicaciones de programación.

2.2.1 PANDAS

Toda la información que se expondrá a continuación esta proporcionada directamente de la pagina oficial de pandas, la librería oficial para la gestión de datos de Python. [2]. Pandas

tiene muchísimas funcionalidades, para este proyecto solo se usará para leer y extraer los datos de las hojas de cálculo, por lo tanto, solo se explicarán las funcionalidades que se han usado para el proyecto.

Pandas funciona mediante Data Frames, que es el conjunto de datos en un formato especifico que es como un diccionario, donde coge los headings de las hojas para crear los atributos y luego cada fila tiene una serie de datos que pertenecen a los atributos. Para poder usar esta librería hay que instalarla con pip install y luego importarla en el .py de la siguiente manera:

```
import pandas as pd
```

2.2.1.1 Transformar hoja de cálculo en Data Frame

Pandas tiene una función que sirve para cualquier archivo como Excel, csv , dase de datos. La función se llama pandas.read_ con la extensión de lo que se quiera leer. En este caso se quiere leer un archivo xlsx por lo tanto el formato para poder transformar la hoja de cálculo a data frame es:

```
nombre_hoja = 'Hoja1'
path = 'bbdd.xlsx'
df = pd.read_excel(path,sheet_name=nombre_hoja)
```

Esto lo que hace es generar un data frame de la hoja proporcionado, a la función hay que darle la dirección del archivo y cual es la hoja sobre la quiere trabajar.

2.2.1.2 Obtención de datos del data frame

La estructura del data frame es parecido a la de una matriz o tabla, mediante dos índices nos podemos mover por filas columnas y filas, cada fila hace referencias a un registro que tiene información sobre cada campo del registro.

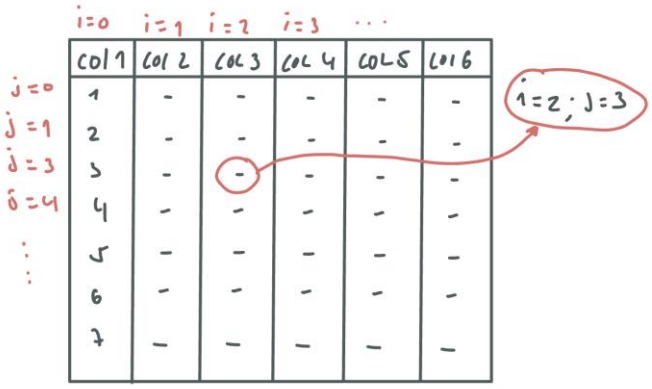


Ilustración 3: Estructura matricial de los data frames

Para poder acceder a un dato en concreto debemos usar una función de pandas que tiene la siguiente estructura `df.iat[fila,columna]` que nos permite sacar un datos en concreto. Siempre se ha de tener en cuenta que los índices empiezan con el valor cero. Como se puede ver en la siguiente imagen, el índice de la columna hace referencia a una columna de la hoja de cálculo.

	A	B	C	D	E	
	0	1	2	3	4	5
	AA_ID	Tipo de Oportunidad	ID-Lead	Fecha de Lead	Fecha de oportunidad	E
	I - 00001	Asesoramiento personalizado integral	2	29/09/2021	01/10/2021	S
	I - 00002	Asesoramiento personalizado integral	407	07/10/2021	07/10/2021	K
	I - 00003	Asesoramiento personalizado integral	549	12/10/2021	14/10/2021	C
	I - 00004	Asesoramiento personalizado integral	461	07/10/2021	14/10/2021	S
	I - 00066	Asesoramiento personalizado integral	963	20/10/2021	28/10/2021	A

Ilustración 4: Relación de las columnas de excel con índice para df.iat[i,j]

Para poder acceder a todos los datos de cada uno de los registros y quedarnos con los campos de interés debemos usar un recurso muy conocido en la programación que es un bucle for, queda demostrado a continuación su funcionamiento.

```
for i  in range(1,rows):

    id_cliente = int(df.iat[i,2])
    tipo_cliente = aux_funciones.tip_cliente_bien(df.iat[i,9])
    partner = aux_funciones.buscar_partner(df.iat[i,28])
    estatus = aux_funciones.buscar_estatus(df.iat[i,32])
    fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))
```

```
tipo_oportunidad = aux_funciones.tip_oportunidad_bien(df.iat[i,1])
facturacion = aux_funciones.facturacion_bien(df.iat[i,21])
numero_empleados = aux_funciones.numero_empleados_bien(df.iat[i,22])
provincia =aux_funciones.provincia(df.iat[i,24])
ccaa = aux_funciones.ccaa_bien(df.iat[i,26])
sector = aux_funciones.sector_bien(df.iat[i,19])
```

Estos junto a la función de `df.axes[0],` son las únicas funciones que se usan del modulo de pandas. `df.axes[0]` nos permite saber cuantos registros (filas) tiene la base de datos, se usa para poder hacer el bucle for.

2.2.2 TKINTER

`tkinter` es una librería de Python que proporciona una interfaz gráfica de usuario (GUI) para crear aplicaciones con ventanas, botones, cuadros de texto, tablas y otros elementos interactivos. Es parte de la biblioteca estándar de Python y es ampliamente utilizada para desarrollar aplicaciones con una interfaz gráfica amigable y atractiva.

Algunos de los comandos básicos para crear elementos de interfaz gráfica con `tkinter` son los siguientes:

- Crear una ventana
- Crear un botón
- Crear una tabla (treeview)
- Crear un entry

2.2.2.1 Crear una venta

En `tkinter`, la creación de una ventana se realiza mediante la clase `Tk`. `Tk` representa la ventana principal de la aplicación y es la base sobre la cual se construye toda la interfaz gráfica. Cuando creamos una instancia de `Tk`, se crea una nueva ventana que será el contenedor principal para todos los elementos gráficos que queramos mostrar.

Al crear la ventana principal con `Tk`, se pueden definir opciones adicionales, como el tamaño de la ventana, el título, el fondo, etc.

La sintaxis para crear una ventana en `tkinter` es bastante sencilla:

```
import tkinter as tk

import tkinter as tk

ventana = tk.Tk()
ventana.title("Mi Ventana")   # Título de la ventana
ventana.geometry("500x300")   # Tamaño de la ventana (ancho x alto)
ventana.configure(bg="white")   # Color de fondo de la ventana

# Aquí irían los widgets y elementos de la interfaz gráfica

ventana.mainloop()
```

Explicación del código:

- Importamos el módulo `tkinter` con el alias `tk`.
- Creamos una instancia de la clase `Tk` y la asignamos a la variable ventana. Esto crea una nueva ventana principal en la aplicación.
- Entre la creación de la ventana (`Tk`) y el bucle principal (`mainloop`()), podemos agregar widgets y elementos para construir la interfaz gráfica. Estos elementos pueden ser botones, etiquetas, campos de texto, tablas, imágenes, etc. La ubicación y apariencia de estos elementos dependerá de cómo los configuremos.
- Finalmente, llamamos al método `mainloop()` en la instancia de `Tk`. Este método inicia el bucle principal de la aplicación, que se encarga de detectar y responder a las interacciones del usuario con la interfaz gráfica. El programa se mantendrá en este bucle hasta que la ventana sea cerrada por el usuario.

Una vez que se haya creado la ventana principal, podemos ir agregando más widgets y elementos para construir la interfaz gráfica, utilizando métodos como `Label`, `Button`, `Entry`, `Canvas`, entre otros, para añadir etiquetas, botones, campos de texto y otras funcionalidades.

En resumen, la creación de una ventana con `tkinter` es el primer paso para desarrollar una interfaz gráfica de usuario y proporcionar una experiencia interactiva para los usuarios de

la aplicación. A partir de esta ventana principal, podemos agregar y configurar diferentes widgets para construir una interfaz más compleja y funcional.

2.2.2.2 Botones

En `tkinter,` los botones son elementos de interfaz gráfica que permiten a los usuarios interactuar con la aplicación al hacer clic en ellos. Los botones pueden realizar acciones específicas cuando se presionan, como ejecutar funciones, cambiar el estado de la interfaz o realizar otras operaciones.

Para crear un botón en `tkinter`, se utiliza la clase `Button`. La sintaxis básica para crear un botón es la siguiente:

```
import tkinter as tk

ventana = tk.Tk()

def funcion_del_boton():
    # Código que se ejecutará cuando se presione el botón
    print("¡Botón presionado!")

boton = tk.Button(ventana, text="Haga clic aquí",
command=funcion_del_boton)
boton.pack()

ventana.mainloop()
```

Explicación del código:

- Importamos el módulo `tkinter` con el alias `tk`.
- Creamos una instancia de la clase `Tk`, que representa la ventana principal de la aplicación.
- Definimos una función llamada `funcion_del_boton`, que se ejecutará cuando el botón sea presionado. En este caso, simplemente muestra un mensaje en la consola.
- Creamos un botón utilizando la clase `Button`. El botón se asocia a la ventana principal (ventana) y tiene el texto "Haga clic aquí". Además, le asignamos la función `funcion_del_boton` para que se ejecute cuando el botón sea presionado.

- Utilizamos el método `pack()` para empaquetar el botón en la ventana principal.
- Llamamos al método `mainloop()` para iniciar el ciclo de eventos de la interfaz gráfica y permitir la interacción con la aplicación.

Cuando se ejecute el programa, se mostrará una ventana con un botón que dice "Haga clic aquí". Al hacer clic en el botón, se imprimirá el mensaje "¡Botón presionado!" en la consola.

Los botones en `tkinter` son una forma fundamental de agregar interacción a las aplicaciones, y se pueden utilizar para realizar diversas tareas, como ejecutar funciones, cambiar valores, abrir ventanas adicionales, enviar datos y mucho más.

2.2.2.3 Tablas

En `tkinter`, el widget `Treeview` es utilizado para crear tablas o árboles de datos, lo que permite mostrar información en formato de filas y columnas de manera organizada. El `Treeview` es especialmente útil cuando se necesita mostrar datos estructurados, como listas jerárquicas o información tabular con varias columnas.

Para crear un `Treeview` en `tkinter`, se utiliza la clase `ttk.Treeview`, que es parte del módulo `ttk` (tema de `tkinter`). La sintaxis básica para crear un `Treeview` es la siguiente:

```
import tkinter as tk
from tkinter import ttk

ventana = tk.Tk()

# Crear un Treeview
tabla = ttk.Treeview(ventana, columns=("Columna1", "Columna2", ...))
tabla.heading("#0", text="ID")  # Crea la columna de índices (opcional)
tabla.heading("Columna1", text="Encabezado1")
tabla.heading("Columna2", text="Encabezado2")
# Otros encabezados de columnas (si es necesario)

tabla.pack()

ventana.mainloop()
```

Explicación del código:

Importamos el módulo `tkinter` con el alias `tk` y el módulo `ttk` para acceder al `Treeview`.

- Creamos una instancia de la clase `Tk` para crear la ventana principal de la aplicación.
- Creamos un `Treeview` utilizando la clase `ttk.Treeview`. El `Treeview` se asocia a la ventana principal (ventana).
- Especificamos las columnas que deseamos mostrar en el `Treeview` utilizando el parámetro `columns`. Cada columna se identifica con un nombre único entre paréntesis. Siempre hay una columna especial llamada #0 que actúa como columna de índices. A continuación, podemos agregar más columnas según sea necesario, como "Columna1", "Columna2", etc.
- Usamos el método `heading`() para establecer los encabezados de las columnas. Podemos proporcionar etiquetas para cada columna, lo que ayudará a comprender qué información se muestra en cada columna.
- Utilizamos el método `pack()` para empaquetar el `Treeview` en la ventana principal y que sea visible para el usuario.
- Finalmente, llamamos al método `mainloop()` para iniciar el bucle principal de la aplicación y permitir la interacción con la interfaz gráfica.

Una vez que se ha creado el `Treeview`, podemos agregar datos a las filas y columnas utilizando métodos como `insert()`, `insert_parent()`, `insert_child()`, etc. Además, es posible agregar funcionalidades adicionales, como selección de filas, ordenamiento, editar celdas, agregar imágenes, entre otras, para personalizar la apariencia y funcionalidad del `Treeview`.

En resumen, el `Treeview` en `tkinter` es una herramienta poderosa para representar datos estructurados y tabulares de manera organizada y visualmente atractiva en una interfaz gráfica de usuario. Es muy útil cuando se necesita mostrar información jerárquica o en forma de tabla para una mejor comprensión y navegación de los datos.

2.2.2.4 Entry

En `tkinter`, el widget `Entry` es utilizado para crear campos de texto de una sola línea, donde el usuario puede ingresar y editar texto. Es similar a un cuadro de texto en el que los usuarios pueden escribir información. Los campos de texto `Entry` son muy útiles para recibir datos del usuario, como nombres, direcciones o cualquier otra información que requiera entrada de texto.

Para crear un campo de texto `Entry` en `tkinter,` se utiliza la clase `Entry`. La sintaxis básica para crear un campo de texto es la siguiente:

```
import tkinter as tk

ventana = tk.Tk()

# Función que se ejecuta cuando se presiona el botón "Aceptar"
def obtener_texto():
    texto = cuadro_texto.get()
    print("Texto ingresado:", texto)

# Crear un campo de texto Entry
cuadro_texto = tk.Entry(ventana)
cuadro_texto.pack()

# Crear un botón para obtener el texto ingresado
boton_aceptar = tk.Button(ventana, text="Aceptar", command=obtener_texto)
boton_aceptar.pack()

ventana.mainloop()
```

Explicación del código:

- Importamos el módulo `tkinter` con el alias `tk`.
- Creamos una instancia de la clase `Tk`, que representa la ventana principal de la aplicación.
- Definimos una función llamada `obtener_texto()`, que se ejecutará cuando se presione el botón "Aceptar". En esta función, obtenemos el texto ingresado en el campo de texto `Entry` utilizando el método `get(),` y luego lo imprimimos en la consola.

- Creamos un campo de texto `Entry` utilizando la clase `Entry`. El campo de texto se asocia a la ventana principal (ventana).
- Utilizamos el método `pack()` para empaquetar el campo de texto y el botón en la ventana principal.
- Creamos un botón con la clase `Button`, que tiene el texto "Aceptar" y está asociado a la función `obtener_texto()` para que se ejecute cuando se presione el botón.
- Utilizamos el `método pack()` para empaquetar el botón en la ventana principal.
- Llamamos al método `mainloop()` para iniciar el ciclo de eventos de la interfaz gráfica y permitir la interacción con la aplicación.

Cuando se ejecute el programa, se mostrará una ventana con un campo de texto y un botón "Aceptar". Cuando el usuario ingrese texto en el campo de texto y presione el botón, se imprimirá el texto ingresado en la consola.

El widget `Entry` en `tkinter` ofrece varias opciones para configurar su comportamiento, como el ancho del campo, opciones de validación, ocultar el texto (por ejemplo, para contraseñas) y más. Es una herramienta útil para recibir entrada de texto del usuario en aplicaciones gráficas.

2.2.2.5 Messagebox

El `messagebox` en `tkinter` es un módulo que proporciona una forma sencilla de mostrar cuadros de diálogo modales, como mensajes de alerta, mensajes de información, preguntas al usuario y mensajes de error. Estos cuadros de diálogo se utilizan para interactuar con el usuario y obtener respuestas o confirmaciones para ciertas acciones en la interfaz gráfica de usuario.

Para utilizar el `messagebox` en `tkinter`, es necesario importar el módulo `messagebox` del paquete `tkinter`. A continuación, se pueden utilizar diferentes funciones de `messagebox` para mostrar diferentes tipos de cuadros de diálogo.

Cuadro de mensaje de información (`showinfo`):

```
def mostrar_informacion():
    messagebox.showinfo("Información", "Este es un cuadro de mensaje de
información.")

btn_info = tk.Button(ventana, text="Mostrar información",
command=mostrar_informacion)
btn_info.pack()
```

Cuadro de mensaje de advertencia (`showwarning`):

```
def mostrar_advertencia():
    messagebox.showwarning("Advertencia", "Se ha detectado una
advertencia en la aplicación.")

btn_advertencia = tk.Button(ventana, text="Mostrar advertencia",
command=mostrar_advertencia)
btn_advertencia.pack()
```

Cuadro de mensaje de error (`showerror`):

```
def mostrar_error():
    messagebox.showerror("Error", "Se ha producido un error en la
aplicación.")

btn_error = tk.Button(ventana, text="Mostrar error",
command=mostrar_error)
btn_error.pack()
```

Cuadro de mensaje de pregunta (`askquestion`):

```
def preguntar():
    respuesta = messagebox.askquestion("Pregunta", "¿Está seguro de
continuar?")
    if respuesta == "yes":
        print("El usuario ha confirmado.")
    else:
        print("El usuario ha cancelado.")

btn_pregunta = tk.Button(ventana, text="Preguntar", command=preguntar)
btn_pregunta.pack()
```

Estas funciones muestran diferentes tipos de cuadros de diálogo según el propósito y la interacción requerida. Los cuadros de diálogo `messagebox` son modales, lo que significa que bloquean la interacción con la ventana principal hasta que se cierran, lo que asegura que el usuario responda antes de continuar con otras acciones en la aplicación.

En resumen, el `messagebox` en `tkinter` es una herramienta útil para mostrar mensajes y obtener interacciones del usuario en forma de cuadros de diálogo modales. Se puede utilizar para proporcionar información, advertencias, preguntas y mensajes de error, mejorando así la experiencia de usuario en la aplicación.

2.2.3 MYSQL.CONNECTOR

El módulo de `mysql.connector` nos permite acceder a la base de datos que esta creada en la herramienta de MySQL y poder interactuar con dicha base de datos. Para poder realizar estas acciones hay una serie de funciones que se describirán a continuación.

La conexión con la base de datos es un comando que tiene la siguiente estructura y los campos a llenar son para poder identificar la base de datos que se quiere manipular. El comando es `myqls.connector.conect():`

```
midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
```

Una vez conectados a la base de datos hay que crear un cursor con el que podremos ejecutar diferentes acciones dentro de mysql.

```
cursos = midb.cursor()
```

Con el curso creado hay dos comandos que nos hace sirven uno para decirle la tarea que quieres realizar y otra que es para ejecutarlo en la base de datos y guardarlo. El ultimo comendo que no es obligatorio, pero si es buena práctica es el close, que nos permite cerrar la conexión con la base datos.

```
sql = f"INSERT INTO proyecto
(id_cliente,tipo_cliente,partner,estatus,tipo_oportunidad,inversión,descr
ipción_empresa,interes,ayuda_aplicada,sector,cnae,facturacion,numero_empl
eados,provincia,ccaa) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"
        values =
(id_cliente,tipo_cliente,partner,estatus,tipo_oportunidad,inversión,descr
ipción_empresa,interes,ayuda_aplicada,sector,cnae,facturacion,numero_empl
eados,provincia,ccaa)

cursos.execute(sql,values)
midb.commit()
midb.close()
```

El comando execute puede hacerse de dos maneras diferentes, una es hacer un comando SQL normal y otro es como se muestra. Se hace con dos valores uno con el comando y luego otro que contiene los valores que se quieren implementar. No solo vale para hacer INSERT sino que también se puede hacer otras cosas como UPDATE, DROP TALBE, CREATABLE Y JOINS, entre otros.

2.3 STRUCTURED QUERY LANGUAGE (SQL)

SQL es un lenguaje que nos permite interactuar y crear bases de datos de una forma rápida y sencilla. Este proyecto solo requiere comandos de crear una base de datos, tablas nuevas, añadir registros, actualizar registro y mostrar los datos. Por lo tanto, en esta sección se expondrá cuáles son los comandos que harán falta en el proyecto y la explicación de como crear una base de datos. Toda la información que se describe a continuación viene dará de tres cursos que se han realizado para aprender sobre SQL, los cursos que se han realizado son los siguientes:

- 100 Days of Code: The Complete Python Pro Bootcamp for 2023 - Udemy[3]
- The Complete SQL Bootcamp: Go from Zero to Hero - Udemy[3]
- Python: HTML, CSS, Flask y MySQL – Hola Mundo[4]

2.3.1 CREACIÓN DE UNA BASE DE DATOS

La creación de una base de datos relacionada consta de una serie de tablas que están relacionadas entre si para poder crear registros de unos clientes, siempre se hace referencia a clientes porque es para lo que se va a usar en este proyecto. A la hora de crear una base de datos relacionada hay que entender dos fundamentos muy sencillos:

- Uno es que en cada tabla tiene que haber un identificador único que es la primary key y nos permite acceder a la información de dicho registro.
- Dos las tablas se relacionan mediante foreign key, que es cuando una tabla adquiere información sobre otra.
- Tres al crear la base de datos hay que saber que el orden de impletacion de la tablas debe ser que nunca se puede hacer una tabla con un foreign key, si la tabla al que lo relaciona no está creado
- Cuatro hay que saber que hay diferentes tipos de variables y que hay que indicarlo a la hora de hacer la tabla e informar si ese algún campo puede estar vació o no.
- Cinco las tablas pueden estar relacionadas de diferentes maneras 1:1 n:1 1:n o n:n.

El comando para crear una tabla es

```
create table tipo_proyecto (
        id_tipo_p int auto_increment primary key,
        tipo_proyecto varchar(150)
);
```

Como se puede ver esta el comando créate seguido del nombre, entre los paréntesis quedan los campos que tendrán la tabla y el tipo de variable que es dicho campo.

El comando primary key hace referencia a esa columna numérica que hará de identificador de cada fila, donde no se puede repetir ningún número y en este caso se ha decido que se autoincremente cada vez que hacemos un registro nuevo y así no tener que ir haciéndolo a mano.

2.3.1.1 Tipos de variables

- INTEGER o INT: Representa valores enteros, como números positivos o negativos sin decimales.
- FLOAT o DOUBLE: Representa valores numéricos de punto flotante, que pueden contener decimales y tienen mayor precisión que los enteros.
- VARCHAR o CHAR: Representa cadenas de caracteres de longitud variable (VARCHAR) o longitud fija (CHAR). Se utilizan para almacenar texto y su longitud puede variar o ser fija.
- DATE: Representa fechas, como "yyyy-mm-dd".
- TIME: Representa horas, minutos y segundos en formato "hh:mm:ss".
- DATETIME o TIMESTAMP: Representa una combinación de fecha y hora en formato "yyyy-mm-dd hh:mm:ss".
- BOOLEAN o BOOL: Representa valores booleanos, es decir, verdadero (TRUE) o falso (FALSE).
- BLOB: Representa datos binarios, como imágenes, archivos o documentos.
- DECIMAL o NUMERIC: Representa números decimales con una precisión fija, ideal para cálculos financieros.
- ENUM: Representa un conjunto de valores posibles, donde solo se puede seleccionar uno de esos valores.

Es importante seleccionar el tipo de dato adecuado para cada columna en la tabla, según el tipo de información que se desea almacenar. Cada tipo de dato tiene sus propias características y limitaciones, y elegir el tipo correcto contribuirá a un diseño eficiente y preciso de la base de datos.

2.3.1.2 Foreing Key

En SQL, una foreign key (clave foránea) es un concepto que se utiliza para establecer una relación entre dos tablas de una base de datos. La foreign key es una columna o un conjunto de columnas en una tabla que hace referencia a la primary key (clave primaria) de otra tabla.

Cuando se crea una foreign key, se establece una restricción de integridad referencial entre las dos tablas. Esto significa que los valores en la columna de la foreign key deben coincidir con los valores en la columna de la primary key de la otra tabla, o deben ser nulos.

Veamos un ejemplo para entender cómo funciona una foreign key:

Supongamos que tenemos dos tablas en nuestra base de datos: "Clientes" y "Pedidos". La tabla "Clientes" tiene una columna llamada "ID_Cliente" como su primary key, que es única para cada cliente registrado. La tabla "Pedidos" tiene una columna llamada "ID_Cliente" que es una foreign key y se refiere a la columna "ID_Cliente" de la tabla "Clientes".

Cuando un nuevo pedido es registrado en la tabla "Pedidos", el valor de la columna "ID_Cliente" en esa fila debe corresponder a un valor existente en la columna "ID_Cliente" de la tabla "Clientes". De lo contrario, la base de datos rechazará la inserción o actualización del registro en la tabla "Pedidos", ya que estaría violando la restricción de integridad referencial.

El uso de foreign keys en SQL asegura que las relaciones entre tablas sean consistentes y precisas, lo que ayuda a mantener la integridad de los datos en la base de datos. Además, las foreign keys permiten realizar consultas que involucren múltiples tablas y obtener información combinada de manera eficiente.

Es importante tener en cuenta que cuando se utiliza una foreign key, se debe tener cuidado al eliminar o actualizar registros en la tabla referenciada (en este caso, "Clientes").

Dependiendo de cómo se configure la foreign key, se puede evitar que se realicen acciones que podrían dejar datos huérfanos o inconsistentes en la base de datos. Por lo tanto, es fundamental definir las acciones apropiadas para "ON DELETE" (cuando se elimina un registro en la tabla referenciada) y "ON UPDATE" (cuando se actualiza el valor de la primary key en la tabla referenciada).

2.3.1.3 Relación entre tablas

En SQL, las relaciones entre tablas se pueden clasificar en tres tipos principales: 1:1 (uno a uno), 1:n (uno a muchos) y n:n (muchos a muchos). Cada tipo de relación describe cómo se relacionan los datos en una tabla con los datos en otra tabla.

RELACIÓN 1:1 (UNO A UNO):

En una relación 1:1, un registro en una tabla está relacionado con exactamente un registro en otra tabla, y viceversa. Es decir, cada fila en la primera tabla está vinculada a una sola fila en la segunda tabla y viceversa. Esta relación se establece mediante el uso de claves primarias y claves foráneas.

Un ejemplo común de relación 1:1 sería una tabla "Persona" con información personal de individuos y otra tabla "Detalles" con detalles adicionales sobre cada persona. Cada persona tendría una única fila en la tabla "Persona" y otra fila única en la tabla "Detalles" que se relaciona con esa persona mediante una clave primaria y una clave foránea.

RELACIÓN 1:N (UNO A MUCHOS):

En una relación 1:n, un registro en una tabla está relacionado con varios registros en otra tabla, pero los registros en la segunda tabla solo están vinculados a un registro en la primera tabla. En otras palabras, un registro en la primera tabla puede estar relacionado con muchos registros en la segunda tabla, pero un registro en la segunda tabla solo puede estar vinculado a un único registro en la primera tabla.

Un ejemplo de relación 1:n sería una tabla "Cliente" con información de clientes y otra tabla "Pedidos" con información de los pedidos realizados por esos clientes. Cada cliente

puede tener múltiples pedidos, pero cada pedido solo pertenece a un cliente específico. La relación se establece mediante el uso de una clave primaria en la tabla "Cliente" y una clave foránea en la tabla "Pedidos".

RELACIÓN N:N (MUCHOS A MUCHOS):

En una relación n:n, varios registros en una tabla están relacionados con varios registros en otra tabla. Es decir, muchos registros en la primera tabla están relacionados con muchos registros en la segunda tabla. Para implementar una relación n:n, se requiere una tabla adicional, llamada tabla de relación o tabla intermedia, que vincule las dos tablas principales a través de claves primarias y foráneas.

Un ejemplo de relación n:n sería una tabla "Estudiantes" con información sobre diferentes estudiantes y otra tabla "Cursos" con información sobre diferentes cursos. Muchos estudiantes pueden estar inscritos en varios cursos, y cada curso puede tener varios estudiantes inscritos. Para representar esta relación, se crearía una tabla intermedia "Inscripciones" que tendría una clave primaria compuesta por las claves primarias de las tablas "Estudiantes" y "Cursos", estableciendo así una relación n:n.

2.3.2 INSERT

El comando `INSERT INTO` se utiliza en SQL para insertar nuevos registros o filas en una tabla existente. Permite agregar datos a una tabla especificando los valores que se desean insertar en cada columna de la tabla. La sintaxis básica del comando `INSERT INTO` es la siguiente:

```
INSERT INTO nombre_de_tabla (columna1, columna2, columna3, ...)
VALUES (valor1, valor2, valor3, ...);
```

Donde:

- nombre_de_tabla: Es el nombre de la tabla en la que se desea insertar los datos.

- columna1, columna2, columna3, ...: Son los nombres de las columnas en la tabla a las que se insertarán los valores.
- valor1, valor2, valor3, ...: Son los valores que se desean insertar en las respectivas columnas.

Ejemplo de uso del comando `INSERT INTO`:

Supongamos que tenemos una tabla llamada "Clientes" con las columnas "ID_Cliente", "Nombre", "Apellido" y "CorreoElectronico". Para agregar un nuevo cliente a la tabla, podríamos usar el siguiente comando:

```
INSERT INTO Clientes (ID_Cliente, Nombre, Apellido, CorreoElectronico)
VALUES (1, 'Juan', 'Pérez', 'juan.perez@example.com');
```

2.3.3 UPDATE

El comando `UPDATE` se utiliza en SQL para modificar o actualizar los valores de uno o más registros existentes en una tabla. Permite cambiar los valores de una o varias columnas en registros específicos que cumplan con ciertas condiciones. La sintaxis básica del comando `UPDATE` es la siguiente:

```
UPDATE nombre_de_tabla
SET columna1 = nuevo_valor1, columna2 = nuevo_valor2, ...
WHERE condicion;
```

Donde:

- nombre_de_tabla: Es el nombre de la tabla en la que se desea actualizar los datos.
- columna1, columna2, ...: Son los nombres de las columnas que se desean actualizar.
- nuevo_valor1, nuevo_valor2, ...: Son los nuevos valores que se asignarán a las respectivas columnas.

- condicion: Es una cláusula opcional que permite especificar qué registros se deben actualizar. Solo los registros que cumplan con la condición serán modificados. Si esta cláusula se omite, todos los registros de la tabla serán actualizados.

Ejemplo de uso del comando `UPDATE`:

Supongamos que tenemos una tabla llamada "Productos" con las columnas "ID_Producto", "Nombre", "Precio" y "Stock". Si queremos actualizar el precio de un producto específico con ID_Producto=1, podríamos usar el siguiente comando:

```
UPDATE Productos
SET Precio = 25.99
WHERE ID_Producto = 1;
```

2.3.4 JOINS

En SQL, la función `JOIN` se utiliza para combinar registros de dos o más tablas en base a una columna común entre ellas. El `JOIN` permite realizar consultas que involucren múltiples tablas y obtener resultados combinados, relacionando registros de manera lógica y eficiente. Hay diferentes tipos de joins, y uno de los más comunes es el `INNER JOIN`.

La sintaxis básica del `INNER JOIN` es la siguiente:

```
SELECT columnas
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun;
```

Donde:

- columnas: Son las columnas que se desean seleccionar en el resultado de la consulta.
- tabla1 y tabla2: Son los nombres de las tablas que se desean combinar.
- columna_comun: Es la columna que actúa como clave de relación entre las dos tablas. Esta columna debe existir en ambas tablas y tener el mismo tipo de dato.

El INNER JOIN devuelve únicamente los registros que tienen coincidencias en ambas tablas. En otras palabras, solo se seleccionan los registros donde los valores de la columna común son iguales en ambas tablas.

Ejemplo de uso del `INNER JOIN`:

Supongamos que tenemos dos tablas: "Clientes" y "Pedidos". La tabla "Clientes" tiene las columnas "ID_Cliente", "Nombre" y "Apellido", mientras que la tabla "Pedidos" tiene las columnas "ID_Pedido", "ID_Cliente" y "Fecha". Queremos obtener una lista de todos los pedidos junto con el nombre y apellido del cliente que realizó cada pedido.

La consulta con `INNER JOIN` sería:

```
SELECT Pedidos.ID_Pedido, Clientes.Nombre, Clientes.Apellido,
Pedidos.Fecha
FROM Pedidos
INNER JOIN Clientes
ON Pedidos.ID_Cliente = Clientes.ID_Cliente;
```

Esta consulta combinará los registros de ambas tablas en base a la columna "ID_Cliente", mostrando solo los pedidos que tienen un cliente asociado. El resultado será una tabla que incluirá el ID del pedido, el nombre y apellido del cliente y la fecha del pedido.

En resumen, el `INNER JOIN` es una función importante en SQL que permite combinar registros de tablas relacionadas y obtener información enriquecida de la base de datos. Al utilizar `INNER JOIN`, se asegura que solo se seleccionen registros con coincidencias en ambas tablas, lo que facilita el análisis de datos relacionados.

Capítulo 3. ESTADO DE LA CUESTIÓN

En este capítulo se explicará de forma detalla cual es el marco teórico del proyecto, definiendo que es una base de datos, los tipos de base de datos que existen, modelos implementados de base de datos, lenguajes que se utilizan para "hablar" con dichas bases de datos y la historia de las bases de datos. También se explicará lo que es una hoja de Excel o hoja de cálculo, para que sirve, como funciona y sus principales diferencias con un sistema de base de datos.

3.1 SISTEMA DE BASE DE DATOS

3.1.1 DEFINICIÓN DE UNA BASE DE DATOS

Una base de datos (BBDD) es una recopilación de información que crean un registro sobre alguna activada de la forma mas organizada posible. Cuando hablamos de BBDD hoy todo el mundo sobre entiende que es un sistema de organización electrónico, pero la realidad es que el nombre base de datos hace referencia a la acumulación de registros de una forma organizada.

Muchas BBDD empiezan siendo hojas de cálculo, ya que no son muy complejas y no contienen mucha información. Según estos registros de información van aumentado, la BBDD se va acomplejando y por lo tanto conviene cambiar el sistema de hoja de cálculo a un sistema de administración de base de datos como Access. [5]

En esencia una base de datos consiste en una serie de tablas con diferentes campos, estas tablas pueden estar relacionadas entre ellas. Al funciona en una base de datos se pueden hacer búsquedas o filtrar información de una formar más sencilla para peticiones más complejas.

3.1.2 HISTORIA DE LAS BASES DE DATOS

3.1.2.1 Introducción

Las bases de datos son una herramienta clave para la gestión de datos. El progreso del almacenamiento, procesamiento y acceso de datos desde los primeros métodos de almacenamiento hasta los modernos sistemas de gestión de bases de datos ha sido impresionante.

En la década de 1960 se creó la primera generación de bases de datos, que eran simples y basadas en archivos, principalmente para almacenar información en cintas magnéticas y discos duros. Sin embargo, estos sistemas eran relativamente limitados y carecían de las complejas características de búsqueda y filtrado necesarias.

En los años 70 se crearon los sistemas de bases de datos relacionales, lo que permitió el almacenamiento de datos en tablas relacionales y lenguajes de consulta estructurados. Actualmente, el modelo relacional de base de datos es el más utilizado y muchos sistemas de bases de datos comerciales se basan en este paradigma.[6]

3.1.2.2 Evolución de las bases de datos

Las bases de datos han experimentado una fascinante evolución desde sus inicios hasta la actualidad. En un principio, las bases de datos eran sistemas manuales basados en archivos de papel o tarjetas perforadas. Con el avance de la tecnología, surgieron las bases de datos electrónicas, que permitían almacenar y gestionar datos de manera más eficiente. A continuación, se destacan algunas etapas clave en la evolución de las bases de datos[7] [8]:

- Sistemas de archivos: Los sistemas de archivos fueron el primer paso hacia la organización de datos en formato electrónico. En esta etapa, los datos se almacenaban en archivos planos y se accedía a ellos a través de programas específicos.
- Bases de datos jerárquicas y en red: A medida que las necesidades de almacenamiento y consulta de datos crecían, surgieron las bases de datos

jerárquicas y en red. Estos sistemas permitían establecer relaciones entre los datos y acceder a ellos de manera más estructurada.

- Modelo relacional: La década de 1970 fue testigo del nacimiento del modelo relacional, propuesto por Edgar Codd. Este modelo introdujo el concepto de tablas interconectadas mediante relaciones basadas en claves primarias y foráneas. Los sistemas de gestión de bases de datos relacionales (RDBMS) se convirtieron en la norma de facto para el almacenamiento y consulta de datos.
- Bases de datos objeto-relacionales: En la década de 1990, surgieron las bases de datos objeto-relacionales, que combinaban características del modelo relacional con la programación orientada a objetos. Esto permitió una mayor flexibilidad y eficiencia al manejar tipos de datos más complejos.
- Bases de datos distribuidas: Con la expansión de la tecnología de redes y la necesidad de acceder a datos de forma global, se desarrollaron las bases de datos distribuidas. Estas bases de datos permiten el almacenamiento de datos en múltiples ubicaciones geográficas y la sincronización entre ellas.
- Bases de datos NoSQL: A medida que los volúmenes de datos y la necesidad de escalabilidad aumentaban, surgieron las bases de datos NoSQL. Estos sistemas se caracterizan por su flexibilidad en el esquema de datos y su capacidad para manejar grandes cantidades de información no estructurada.
- Bases de datos en la nube: Con la popularización de la computación en la nube, las bases de datos se trasladaron a entornos de alojamiento remoto y escalable. Las bases de datos en la nube ofrecen ventajas como la disponibilidad, la escalabilidad y la redundancia.

3.1.2.3 Importancia de las bases de datos

La importancia de las bases de datos radica en su papel fundamental como herramientas para la gestión y organización eficiente de grandes volúmenes de información. Algunos puntos clave que resumen su relevancia son[9]:

- Almacenamiento y Organización: Las bases de datos permiten almacenar una gran cantidad de datos de manera estructurada, lo que facilita su organización y recuperación. Esto asegura que la información esté disponible de forma rápida y precisa, evitando la pérdida o duplicación de datos.
- Toma de Decisiones: En diferentes ámbitos, desde empresas hasta investigación científica, las bases de datos proporcionan datos cruciales para la toma de decisiones informadas. Las consultas y análisis de datos permiten identificar tendencias, patrones y oportunidades de mejora.
- Eficiencia en el Acceso a la Información: Gracias a las bases de datos, múltiples usuarios pueden acceder a la información de manera simultánea y desde distintas ubicaciones. Esto mejora la colaboración y eficiencia en el trabajo en equipo.
- Seguridad y Privacidad: Las bases de datos ofrecen mecanismos para garantizar la seguridad y privacidad de los datos. Se pueden implementar restricciones de acceso, encriptación y auditoría de actividades para proteger la información confidencial.
- Escalabilidad: Las bases de datos están diseñadas para manejar grandes volúmenes de datos y pueden crecer y adaptarse conforme aumenta la cantidad de información almacenada, lo que resulta crucial en un mundo cada vez más digital y con datos en constante crecimiento.
- Integración de Aplicaciones: Las bases de datos permiten la integración de diferentes aplicaciones y sistemas, lo que favorece el flujo de información entre ellos y mejora la eficiencia de los procesos.
- Soporte a la Inteligencia de Negocios: Las bases de datos son fundamentales para el análisis de datos y la generación de informes en inteligencia de negocios. Esto ayuda a identificar oportunidades comerciales, evaluar el rendimiento y tomar decisiones estratégicas.

3.1.3 TIPOS DE BASES DE DATOS

Existen muchos tipos de bases de datos, como se ha mencionado antes las bases tiene diferentes usos y su forma de "hablar" con ellas es diferente y por esos existen diferentes tipos de bases de datos.

El orden de la lista según Oracle va de las bases mas utilizadas a las menos utilizadas [10].

3.1.3.1 Base de datos relacionadas

Un base de datos relacionadas son un conjunto de recopilación de datos y proporciona acceso a los datos almacenados. Este formato tiene como estructuras tablas, que luego están relacionadas entre ellas. Estas tablas están compuestas por filas y por columnas, donde la fila indica un ID nuevo que es único y las columnas a los datos que se quieren almacenar [11].

Ilustración 5: Base de datos relacionadas[12]

Un ejemplo muy común es de la comprade algún cliente en alguna tienda online. Cuando el cliente hace una petición, esta petición se queda registrada y se hace una comparación con el inventario disponible. En el caso que no esté disponible la petición será rechaza, mientras que si el producto esta disponible el pedido se aceptara y la tabla de inventario de deberá actualizar. Esto es un claro ejemplo donde se que la tabla de pedidos se relaciona directamente con la tabla de inventario y viceversa.

Las bases de datos relacionales ofrecen una estructura flexible y escalable que permite la adición, modificación y eliminación de datos sin afectar su integridad. Además, eliminan la redundancia de datos, asegurando una consistencia y precisión en los datos almacenados. La capacidad de realizar consultas complejas y combinadas utilizando múltiples tablas,

brinda información valiosa y relevante de manera más eficiente. En resumen, una base de datos relacional ofrece integridad, flexibilidad, consistencia y eficiencia en la obtención de información.

3.1.3.2 Base de datos orientadas a objetos

Una base de datos orientada a objetos está definida como un encapsulado donde se le guardan todos los atributos de dicho elemento. El elemento puede estar dentro una clases o subclases, lo cual le permite heredas todos los atributos dichas clases, a diferencia de las bases de datos relacionadas los atributos no se distribuyen en tablas, sino que son intrínsecos al elemento. A diferencia de otras bases de datos, para poder interactuar con este tipo de bases de datos se debe hacer por lenguaje de código como: Java o C++ [13].

Un ejemplo de base de datos orienta a objetos sería una colección de ropa de una marca, donde un elemento específico, como unas bermudas de lino de la colección de verano, tendría varios atributos, como color, tipo de prenda, material y talla. Sin embargo, las bermudas también estarían clasificadas como una subclase de pantalones, que a su vez es una clase de prenda de ropa de verano. En lugar de organizarse en tablas, esta base de datos se organiza jerárquicamente en niveles.

Una base de datos orientada a objetos permite la representación de datos en términos de objetos que contienen tanto datos como comportamiento. Esto permite una mayor flexibilidad y escalabilidad en la estructura de datos, así como la posibilidad de una mayor reutilización de código. Además, las bases de datos orientadas a objetos pueden manejar mejor datos complejos y no estructurados. En resumen, los beneficios de una base de datos orientada a objetos incluyen una mayor flexibilidad y escalabilidad en la estructura de datos, una mayor reutilización de código y una mejor capacidad para manejar datos complejos.

3.1.3.3 Base de datos distribuidas

Una base de datos distribuida (DBD) es un tipo de sistema de base de datos en el que los datos se almacenan en múltiples sistemas informáticos interconectados en red, en lugar de

en un solo sistema centralizado. En una DBD, cada nodo de la red tiene una copia local de una parte de los datos, lo que permite que los datos sean procesados en diferentes ubicaciones geográficas y a través de múltiples sistemas informáticos.[14]

Ilustración 6: Base de datos distribuida[15]

Un ejemplo de una base de datos distribuida podría ser una red de tiendas de retail que utiliza una DBD para manejar los datos de inventario y ventas. Cada tienda tendría una copia local de la base de datos, pero todas estarían conectadas en una red, permitiendo que los datos se sincronicen y se actualicen en tiempo real.

Cuando se realiza una venta en una tienda, la información se actualizaría automáticamente en la base de datos de esa tienda, y luego se propagaría a las bases de datos de las otras tiendas. Esto permite que todas las tiendas tengan acceso a la información de inventario y ventas actualizada, lo que es esencial para tomar decisiones informadas sobre el inventario y la logística.

Este enfoque tiene varios beneficios. En primer lugar, permite una mayor escalabilidad, ya que se pueden agregar nuevos nodos a la red para manejar un mayor volumen de datos. En segundo lugar, mejora el rendimiento, ya que los datos pueden procesarse de manera paralela en diferentes nodos, lo que reduce la carga en un solo sistema centralizado.

Además, proporciona una mayor disponibilidad, ya que, si un nodo falla, los otros nodos pueden continuar funcionando y procesando los datos.[14]

3.1.3.4 Bases de datos en memoria

Una base de datos en memoria es un tipo de sistema de base de datos en el que los datos se almacenan en la memoria principal del sistema informático, en lugar de en un disco duro. Esto significa que la base de datos puede acceder y procesar los datos de manera más rápida y eficiente, lo que puede mejorar significativamente el rendimiento y la capacidad de respuesta de la aplicación que utiliza la base de datos.

En una base de datos en memoria, los datos se almacenan en estructuras de datos optimizadas para la memoria, como matrices y árboles. Además, se utilizan técnicas de compresión de datos para reducir el tamaño de los datos almacenados en la memoria, lo que permite que se pueda almacenar una mayor cantidad de datos en la memoria principal.[16]

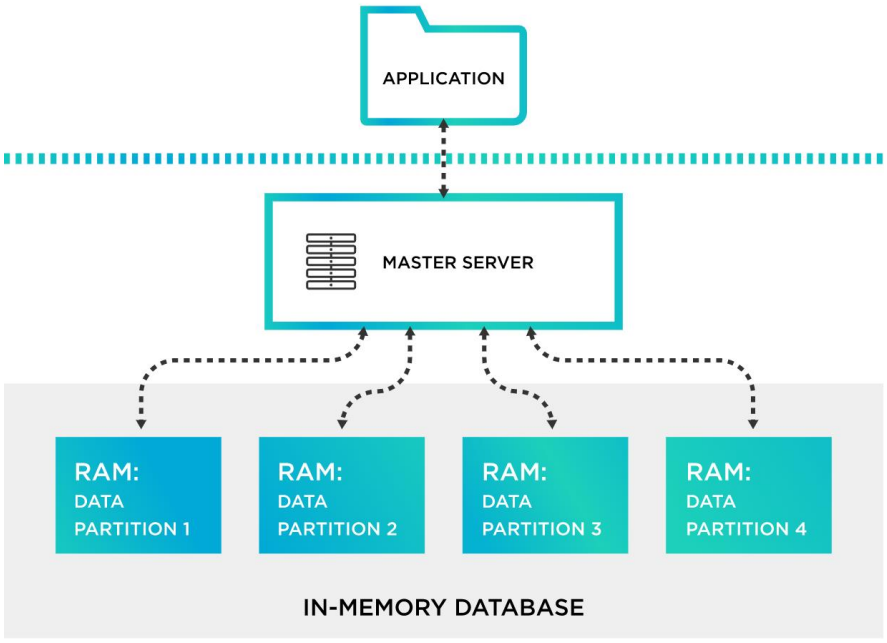


Ilustración 7: Base de datos en memoria[17]

Un ejemplo práctico de una base de datos en memoria podría ser un sistema de monitoreo de sensores en tiempo real para una planta de producción industrial. En este sistema, los datos de los sensores que miden variables como temperatura, presión, humedad, velocidad y otros parámetros críticos se almacenan en la base de datos en memoria.

Dado que estos datos son críticos para la operación segura y eficiente de la planta de producción, es fundamental que el sistema pueda acceder y procesar los datos de manera rápida y eficiente. Si los datos se almacenaran en un disco duro, podría haber un retraso significativo en la adquisición y procesamiento de los datos, lo que podría afectar la capacidad de la planta de producción para detectar y responder rápidamente a cualquier problema.

Las bases de datos en memoria ofrecen importantes mejoras de rendimiento con respecto a las bases de datos tradicionales basadas en disco, a saber, mayor velocidad de lectura y escritura, así como menor latencia de los datos. Sin embargo, estos sistemas presentan varios inconvenientes potenciales, como el elevado coste de la memoria y la necesidad de gestionar la sincronización y persistencia de los datos.[16]

3.1.3.5 Bases de datos tiempo-real

Una base de datos tiempo-real es una base de datos que está diseñada para procesar y almacenar datos en tiempo real, es decir, con una latencia mínima entre la entrada y la salida de los datos. En otras palabras, una base de datos tiempo-real es aquella que está optimizada para manejar datos que cambian rápidamente y necesitan ser procesados inmediatamente, sin ningún tipo de demora.

Las bases de datos tiempo-real se utilizan en una amplia variedad de aplicaciones, como en sistemas de control de procesos industriales, sistemas de monitoreo de sensores, sistemas de seguridad y vigilancia, sistemas de gestión de flotas, entre otros. Estas aplicaciones requieren el procesamiento en tiempo real de grandes volúmenes de datos, lo que significa que cualquier retraso en la entrada o salida de los datos puede ser crítico.[18]

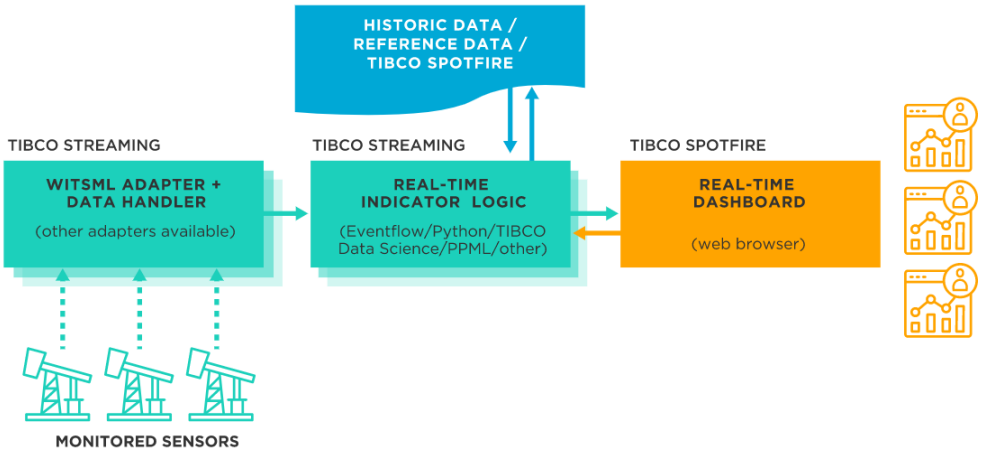


Ilustración 8: Base de datos tiempo-real [19]

Un ejemplo de aplicación de una base de datos tiempo-real podría ser en el monitoreo de la red de una empresa de telecomunicaciones. En este caso, la base de datos tiempo-real se utilizaría para recopilar y analizar datos en tiempo real sobre el tráfico de la red, incluyendo el rendimiento de los dispositivos de red, el ancho de banda utilizado y el número de conexiones activas.

La base de datos tiempo-real permitiría a la empresa de telecomunicaciones monitorear el estado de su red en tiempo real, identificar posibles cuellos de botella y problemas de rendimiento, y tomar medidas proactivas para optimizar la eficiencia y el rendimiento de su red. Esto podría incluir la reconfiguración de dispositivos de red, la asignación de ancho de banda adicional a ciertas conexiones o la implementación de políticas de gestión de tráfico más estrictas para limitar el consumo de ancho de banda por parte de ciertos usuarios o aplicaciones.

En este caso, la base de datos tiempo-real sería esencial para permitir una respuesta rápida y eficaz a los cambios en el tráfico de la red, lo que a su vez permitiría a la empresa de telecomunicaciones mantener un alto nivel de calidad de servicio para sus clientes y maximizar su eficiencia operativa.

Una de las características clave de las bases de datos tiempo-real es la capacidad de procesar datos en tiempo real, es decir, de manera continua, sin pausas ni retrasos. Esto implica que los datos pueden ser ingresados y procesados inmediatamente, lo que permite que la información sea analizada y utilizada para tomar decisiones en tiempo real.

Otra característica importante de las bases de datos tiempo-real es la capacidad de manejar grandes volúmenes de datos, y hacerlo de manera eficiente y escalable. En este sentido, las bases de datos tiempo-real suelen estar diseñadas para funcionar en entornos de alta disponibilidad y para soportar grandes cargas de trabajo.[18]

3.1.4 LENGUAJES DE BASE DE DATOS

La interacción con las bases de datos se hace mediante unos lenguajes específicos. Estos lenguajes nos permiten interactuar con las bases de datos: crear las mismas bases de datos, insertar nuevos registros, actualizar el modelo que se esté implementando, actualizar los datos que ya están en la base de datos e incluso hacer búsquedas de datos específicos mediante la unión de información (tablas).

Los lenguajes más representativos son: Python, SQL, C#, R y PHP. Cada lenguaje tiene sus beneficios y sus desventajas, pero todos tienen la misma finalidad o pueden usarse para

crear e interactuar con las bases de datos. El lenguaje más común para interactuar con las bases de datos y de la cual tengo algo de conocimiento es SQL, por lo tanto, entraremos al detalle sobre este lenguaje.

3.1.4.1 SQL

SQL es el acrónimo para Structured Query Lenguage. Un lenguaje que nos permite manipular y descargar información en un formato de base de datos estructurado. Mediante este lenguaje se puede hacer consultas sobre la base de datos y operaciones algebraicas complejas. Es el lenguaje que mas se usa para en el tipo de bases de datos relacionados. Como se ha comentado en la parte de descripción de tecnologías con SQL se puede hacer:

- Crear tablas y tablas relacionadas
- Insertar datos y registros nuevos
- Rediseñar el tipo de información
- Actualizar datos y registros
- Hacer búsquedas complejas
- Hacer operaciones complejas

3.1.5 LENGUAJE DE PROGRAMACIÓN (PYTHON)

El lenguaje de programación es muy versátil y por lo tanto nos permite hacer muchas cosas. El proyecto usará Python para automatizar y crear el código de una interfaz que permite usar código SQL para hacer todas las operaciones complejas que se tenga que hacer sobre la base de datos.

Una de las ventajas que tiene Python es que tiene librerías que permiten hacer estas tareas de una forma más amena y con mayor facilidad y flexibilidad. Se usarán librerías para crear sistemas de automatización para insertar datos en la base de datos, cera una interfaz y crear códigos para hacer tareas masivas y repetitivas.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Las bases de datos son importantes a nivel personal, pero lo son aún más a nivel empresarial, y se consideran una de las mayores aportaciones que ha hecho la tecnología informática a las empresas. Hoy en día, cualquier organización que se precie, por pequeña que sea, debe tener una Base de Datos para organizar la información de su negocio, pero para que funcione como se necesita, no basta con tener, necesitas saber cómo arreglarlo.

Si no tenemos un responsable de esta tarea en nuestra empresa, es posible contratar a un responsable externo de la empresa.

Las utilidades que tienen las bases de datos son:

- Agrupar y almacenar todos los datos de la empresa en un único lugar.
- Facilitar que se compartan los datos entre los diferentes miembros de la empresa.
- Evitar la redundancia y mejorar la organización de nuestra actividad.
- Visualizar los datos de un cliente o potencial: interacciones, ventas, datos de contacto.
- Conectar los datos de operaciones, facturación e interacciones con cada cliente o potencial
- Activar campañas de marketing o tareas

Si la base de datos se gestiona correctamente, la organización obtendrá varios beneficios. Aumentará su capacidad, habrá tareas que serán rápidas y flexibles por su sencillez, podremos aumentar la seguridad de los datos que almacenamos, y con todo ello aumentaremos el tiempo y por tanto habrá una mejora en el rendimiento.

4.1.1 FLUIDEZ DE LOS DATOS

La estructura anterior es una estructura en formato Excel que nos obliga a tener nuestros registros separados del registro del partner. Esto permite que una vez entre un lead nuevo y se derive el partner lo pueda ver en seguida y aceptar el lead recibiendo la información, de si esta aceptado o no, al instante. Esto hace que tengamos menos errores a al hora de actualizar los datos de derivación del partner.

4.1.1.1 Error en la carga de datos

Como hay que ir actualizando tres archivos de Excel distintas, puede haber algún fallo en este sistema tedioso y manual. Esto lo que hace es reducir ese error, también a veces volvemos a derivar un lead que ya está derivado y esto no puede ser si esta aceptado por otro partner. Esta herramienta permite reducir los siguientes errores:

- Error de actualización en alguna hoja de Excel.
- Rapidez de actualización de información sobre los leads.
- En todo momento los partner y la empresa tienen la misma imagen del estado del lead.

4.1.2 RAPIDEZ DE CONTACTO Y ACEPTACIÓN DE LEADS

El servicio al cliente es una de las cosas mas importantes, si no la más importante, de un servicio, por esta razón la rápida atención al cliente es fundamental para tenerlos contentos. Anteriormente, debido al flujo de datos lento, al cliente se le tardaba en contactar y gracias a un flujo mucho más rápido de datos se le puede contactar al día siguiente.

El funcionamiento de antes era el siguiente:

- Analizar cliente y mandar un correo con la información solo del proyecto al partner.
- El partner analiza la información del cliente y acepta o rechaza, mandando la información por correo.

- Si el partner acepta, actualizar todos los datos en los Excel y manda un correo con la información de contacto del cliente.
- Una vez el partner tenga la información de contacto del cliente se pone en contacto con él.

El funcionamiento que permite la herramienta:

- La información del lead nuevo entra por la mañana una vez que llegue el correo por parte de tecnologías.
- Desde la herramienta podemos ver los nuevos leads y derivarlos al partner con los que mas se ajustan.
- Una vez actualizamos el campo del partner, se actualiza en la BBDD y el partner ya puede ver solo la información del proyecto.
- Una vez acepte podrá ver la información de contacto del cliente, sin nosotros tener que dársela. Se la proporciona la misma herramienta.

Como se puede ver, la información fluye de una forma mucha más rápida. Esto permite que el partner pueda contactar con el cliente de una forma mucha más rápida y así el cliente estar mejor atendido.

4.2 OBJETIVOS

El Objetivo que tiene este proyecto es poder dar la posibilidad de que la base de datos pueda ser más grande. Excel te permite tener registros, pero a más registros más lento y peor funciona, si es cierto que Excel tiene herramientas graficas que una base de datos o no. Esto para el simple hecho de poder tener lo registros y poder manipularlos son herramientas que no son decisivas a la hora de elegir donde hacerlo. SQL nos permite tener muchos más registros y poder manipularlos de mejor forma y con una rapidez superior a la de Excel.

Una vez creada la base da datos en la que poder tener los registros del cliente es necesario saber SQL para poder interactuar con dicha base. Aparte de hacer el traspaso de

información de los registros en Excel a base SQL, se ha decido crear una interfaz que nos permita hacer las tareas básicas y necesarias para poder interactuar. Tareas como actualizar e insertar nevos registros, y aceptar o rechazar leads (que es actualizar datos).

Para resumir el objetivo es hacer un traspaso de unos registros de unos clientes en formato Excel a un formato SQL, que nos permite tener mucha más información y poder manipular esa información de mejor manera. Como segundo objetivo, cera una interfaz para que tanto los partners como nosotros como consultora podamos actualizar datos en la base de datos sin tener que saber SQL. Se creará dos interfaces una para Minsait y otra para cada partner, en este proyecto solo se hará la simulación para un partner, para el resto de partners seria literal solo cobrandizar la aplicación y dar una clave de acceso a cada uno.

4.3 METODOLOGÍA

La forma en la que está estructurada los datos del cliente y la forma en la que se deriva es ineficiente. Esto nos lleva a pensar que se puede crear algo o mejorar algo para poder hacerla mas eficiente. Para ellos se ha seguido la siente metodología para poder analizar los defectos de este sistema y las soluciones correspondientes.

Debido a que se trata de una base de datos, que es lo que se tiene en la hoja de Excel, se ha analizado cual es mejor formato para tener una hoja base de datos. Esto se concluyo que era mediante un formato .db (base de datos) en vez de .xsls (Hoja de cálculo de Excel).

Se ha hecho un estudio de las diferentes hojas que están relacionadas y la información que expone cada una de las tablas y su funcionamiento. Una vez se entienda la información que se muestra, cuando se muestra y bajo que circunstancias; se crea una estructura de base de datos, donde la información se separa en tablas de la forma más eficiente posible.

Cuando se tenga una idea de como estructurar la información de los leads, debemos traspasar toda la información antigua a la nueva base de datos, para no perder ningún registro. Habrá que tener en cuenta que cada día hay clientes (leads) nuevos y por lo tanto también hay que hacer que se inserten en la base de datos.

Se creará unos script con Python que ejecuten comandos SQL para poder hacer esta tarea. El lenguaje Python nos permitirá hacer las actualizaciones de forma automática de la forma mas rápido posible. Mientras que el lenguaje SQL nos permite interactuar con la base de datos para poder ir insertando y actualizando la información.

Una vez se tiene toda la información en la base de datos, solo queda poder mostrarla y crear una herramienta con la que actualizar información sin la necesidad de saber SQL. A esto se le denominara la interfaz con la base de datos NOCODE. Esta interfaz, que es mas bien una herramienta o App, será una combina de códigos y scrips que combinará el lenguaje Python y SQL para poder hacer todas las acciones necesarias.

En el proceso de la herramienta habrá que analizar la lógica que hay detrás de las derivaciones, que campo hay que actualizar, donde se tienen que mostrar si se a aceptado o donde no si se ha rechazado. Toda es lógica deber estar al milímetro y tiene que ser una lógica excluyente, donde dos partners no pueden tener el mismo cliente.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

Semana 1: Investigación y planificación

- Investigar las mejores prácticas para crear e insertar datos en una base de datos SQL desde Python.
- Seleccionar una base de datos SQL para utilizar (por ejemplo, MySQL o PostgreSQL).
- Planificar la estructura de la base de datos y los tipos de datos que se incluirán.

Semana 1-4: Cursos para aprender SQL

- Como estructurar una bbdd
- Relaciones entra tablas y las keys
- Como interactuar con una base de datos para obtener información
- Joins y Updates, para mostrar y actualizar la información

Semana 4-5: Curso de formación de interfaces con Python (Tkinter)

- Aprender a usar la librería
- Aprender a interactuar con bbdd para poder mostrarla
- Funcionamiento de botones y campos para determinar acciones.

Semana 3-4: Desarrollo de la base de datos en SQL

- Crear la base de datos en SQL.
- Configurar las tablas y campos necesarios para almacenar los datos.

Semana 5-7: Creación de código Python para insertar datos en la base de datos

- Crear un script de Python que inserte datos en la base de datos.
- Probar y depurar el código para asegurarse de que funcione correctamente.

Semana 7-8 Desarrollo de la interfaz Python-SQL

- Crear una interfaz en Python que permita a los usuarios actualizar los datos en la base de datos.
- Probar y depurar la interfaz para asegurarse de que funcione correctamente.

Semana 8-9: Documentación y presentación

- Documentar el código y la solución.
- Preparar una presentación para mostrar los resultados del proyecto.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

En este capítulo es donde el alumno debe describir su proyecto. En función del tipo de proyecto la estructura interna variará. El título del mismo, así como sus apartados, son sólo una sugerencia que cada alumno deberá adaptar particularmente a su proyecto.

A su vez, este capítulo podrá extenderse en varios capítulos más, por ejemplo, si mi proyecto consta de varias fases o módulos, lo lógico sería tener varios capítulos donde se describa el desarrollo del proyecto:

- Capítulo 5. Implantación y configuración de la plataforma
- Capítulo 6. Desarrollo del sistema

5.1 ANÁLISIS DEL SISTEMA

5.1.1 ORGANIZACIÓN DE LOS DATOS RECOPILADOS

Para poder trasladar la base de datos en formato hoja de cálculo a base de datos en SQL, tenemos que analizar las tablas que ya existen y ver cuál es la relación entre ellas. Las dos tablas más importantes son: la hoja de oportunidades y la hoja de derivación.

La hoja de oportunidades es donde esta toda la información del cliente, todos sus datos personales, los datos del proyecto y en qué estado esta (en caso de que este derivado, está la información del al último partner que se le derivo su información).

Para poder ordenar un poco la información se podría seccionar la información en tres tablas diferentes: cliente, proyecto y estado. Se debe aclarar que no se añadirán campos nuevos al formulario de los leads, se trabajara sobre la información que hoy en día ese formulario recoge.

El listado de los datos que se recogen son los siguientes, los clasificaremos en tres grupos:

5.1.1.1 NFORMACIÓN CLIENTE

- **ID-lead** → el formulario lleva un contados y registra el numero de la petición como el id del cliente.
- **Tipo oportunidad** → esto depende de la landing por la que entran, hay ayudas que tramitamos de forma masiva (tramitación masiva), proyectos que son un poco flojos (integral) y proyectos fuertes que se acogen a una convocatoria especifica (proyecto)
- **Nombre empresa** → nombre por la que se conoce a la empresa, en caso de que sea un particular será el nombre del cliente.
- **CIF/NIF**
- **Entidad** → cada lead entra por alguna entidad, esto especifica de que banco entra (BBVA, CaixaBank,ect)
- **Fecha lead** → la fecha en la que el cliente a rellenado el formulario
- **Fecha oportunidad** → la fecha en la que la información del lead a sido volcado sobre la hoja de oportunidad
- **Nombre cliente** → el nombre de la persona que se esta poniendo en contacto con la empresa
- **Apellido cliente** → el apellido de la persona que se esta poniendo en contacto con la empresa
- **Correo cliente** → El correo de la persona

Nombre	Tipo variable	Key	tipo de campo
id_cliente	int AUTO_INCREMENT	primary key	-
id_lead	int		R
nombr_ empresa	varchar(30)		R
cif_nif	varchar(30)		R
entidad	int	foreign key	O
nombre_cliente	varchar(30)		R
apellido_cliente	varchar(30)		R
correo_cliente	varchar(30)		R
fecha_lead	date		-

Nombre	Tipo variable	Key	tipo de campo
fecha_oportunidad	date		-

Tabla 1: Tabla para los cilentes

5.1.1.2 INFORMACIÓN POROYECTO

- **Tipo cliente** → si son particulares, pymes, grandes empresas, comunidad de vecinos o ONGS
- **Descuento cliente** → dependiendo de la entidad de la que venga el lead tendrá un descuento sobre el coste de los servicios.
- **Tipología proyecto**
- **Importe de la inversión**
- **Descripción de la empresa**
- **Intereses**
- **Ayuda aplicada**
- **Sector cliente**
- **CNAE**
- **Facturación** → lo que la empresa está consiguiendo facturar en un año, el año anterior para ser exactos
- **Numero empleados** → número de empleados en la empresa, para medir el tamaño de la empresa, son rangos
- **Tamaño empresa** → como se consideran ellos, si son microempresas, pequeñas, medianas o grandes empresas
- **Provincia** → ciudad en la que se encuentra la empresa
- **Domicilio Fiscal** → domicilio fiscal de la empresa o del particular dependiendo del tipo de cliente
- **CCAA** → comunidad autónoma donde se encuentra la empresa

Nombre	Tipo variable	Key	tipo de campo
id_proyecto	int AUTO_INCREMENT	primary key	-
id_cliente	int	foreign key	O

Nombre	Tipo variable	Key	tipo de campo
tipo_cliente	int	foreign key	O
tipo_oportunidad	int	foreign key	O
tipo empresa	int	foreign key	O
inversión	varchar(30)		R
descripción_empresa	varchar(30)		R
interes	varchar(30)		R
ayuda_aplicada	varchar(30)		R
sector	int	foreign key	O
cnae	varchar(30)		R
facturación	int	foreign key	O
numero_empleados	int	foreign key	O
tamaño	int	foreign key	O
provincía	int	foreign key	O
ccaa	int	foreign key	O
Partner	int	foreign key	O
Resultado	int	foreign key	O

Tabla 2: Tabla de los datos de oportunidades

5.1.1.3 Derivación

Como podemos comprobar la mayoría de los campos ya están en alguna de las tablas, por lo tanto, no sería necesario toda esa información para la tabla de derivación. En el caso de que se quiera mostrar esa información se podría hacer un join con los campos que se quiere analizar. Los campos imprescindibles para poder tener el registro de las derivaciones son:

- **id_derivación**
- **id_cliente**
- **partner**
- **fecha_derivación**
- **fecha_finalizado**
- **resultado**

Nombre	Tipo variable	Key	tipo de campo

Nombre	Tipo variable	Key	tipo de campo
id_derivación	int AUTO_INCREMENT	primary key	-
id_cliente	int	foreign key	-
partner	int	foreign key	O
fecha_derivación	date		-
fecha_finalizado	date		-
resultado	int	foreign key	O

Tabla 3:Tabla de los datos de derivación

5.1.1.4 CREACIÓN DE TABLAS BASICAS

Para poder hacer la base de datos de esta forma debemos crear una tabla para cada campo que sea de escoger una opción, esto se hace porque ocupa mucho menos un 1 o 2 que tener la definición completa del campo.

- partners
- estatus
- resultado
- tipo clienet
- tipo de proyecto
- numero empleados
- tamaño empresa
- facturación
- tipo oportunidad
- sector
- provincia
- ccaa

Como se crean estas tablas debido a que son tablas muy simples, se encontrara adjunto en el Anexo I.

5.2 DISEÑO

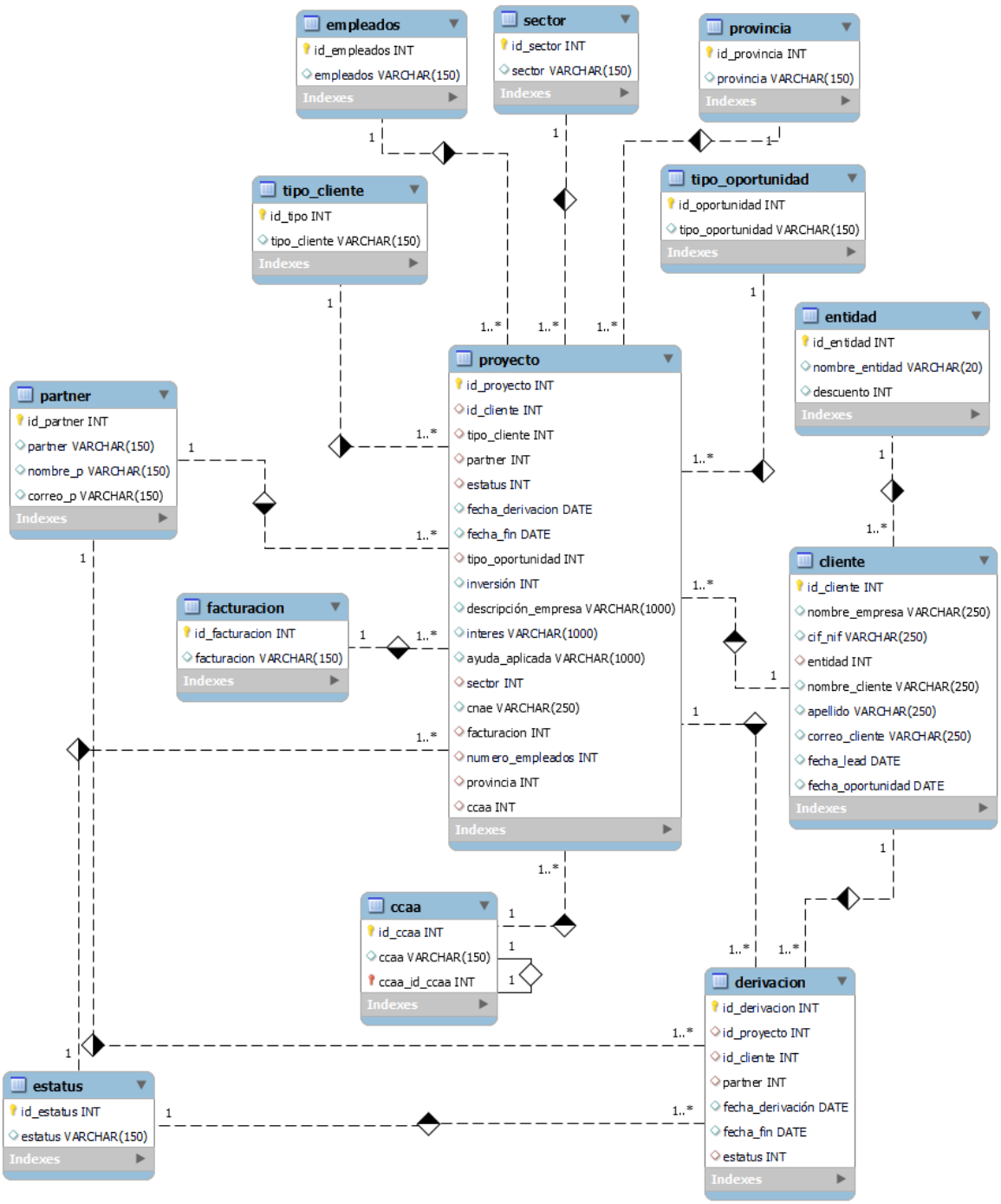


Ilustración 9: Diagrama de relación entra las tablas de SQL

5.3 IMPLEMENTACIÓN

5.3.1 CREACIÓN DE LA BASE DE DATOS

Para poder crear la base de datos sin problemas, debemos seguir una secuencia donde las tablas que tienen foreign key deben hacerse las ultimas. Esto se debe a que deben obtener información de estas tablas, si no se hace en orden no dejara porque la tabla de la que tiene que coger esa información no existe. La creación de estas tablas se puede hacer directamente en la aplicación de MySQL. Los pasos para seguir son los siguientes:

1. Crear las tablas básicas
2. Crear la tabla de clientes
3. Crear la tabla de proyectos
4. Crear la tabla de derivación

Los códigos de las tablas complejas se adjuntan a continuación:

1. El código para poder crear la tabla en SQL es el siguiente:

```
create table cliente(
    id_cliente int auto_increment primary key,
    id_lead int,
    nombre_empresa varchar(50),
    cif_nif varchar(10),
    entidad int, -- foreign key
    nombre_cliente varchar(50),
    apellido varchar(50),
    correo_cliente varchar(100),
    fecha_lead date,
    fecha_oportunidad date,
    foreign key (entidad) references entidad(id_entidad)
);
```

2. El código para poder cera la tabla de

```
create table proyecto(
  id_proyecto int auto_increment primary key,
  id_cliente int, -- foreign key
  tipo_cliente int, -- foreign key (pyme, gran empresa, ...
  partner int, -- foreign key
```

```
  estatus int, -- foreign key
  fecha_derivacion date,
  fecha_fin date,
  tipo_oportunidad int, -- foreign key (masivo, proyecto e integral
  inversión int,
  descripción_empresa varchar(1000),
  interes varchar(1000),
  ayuda_aplicada varchar(1000),
  sector int, -- foreign key
  cnae varchar(250), -- foreign key
  facturacion int, -- foreign key
  numero_empleados int, -- foreign key
  provincia int, -- foreign key
  ccaa int, -- foreign key
  foreign key (id_cliente) references cliente(id_cliente),
  foreign key (tipo_cliente) references tipo_cliente(id_tipo),
  foreign key (tipo_oportunidad)references
tipo_oportunidad(id_oportunidad),
  foreign key (sector) references sector(id_sector),
  foreign key (facturacion) references facturacion(id_facturacion),
  foreign key (numero_empleados) references empleados(id_empleados),
  foreign key (provincia) references provincia(id_provincia),
  foreign key (ccaa) references ccaa(id_ccaa),
  foreign key (partner) references partner(id_partner),
  foreign key (estatus) references estatus(id_estatus)
);
```

3. Código para crear la tabla de derivación

```
create table derivacion(
  id_derivacion int auto_increment primary key,
  id_proyecto int, -- foreing key
  id_cliente int, -- foreign key
  partner int, -- foreign key
  fecha_derivación date,
  fecha_fin date,
  estatus int, -- foreign key
  foreign key(id_proyecto) references proyecto(id_proyecto),
  foreign key (id_cliente) references cliente(id_cliente),
  foreign key (partner) references partner(id_partner),
  foreign key (estatus) references estatus(id_estatus)
);
```

5.3.2 INSERTAR TODOS LOS DATOS ANTIGUOS

Se trata de hacer una transformación de Excel a SQL, por lo tanto, debemos insertar todos los datos que ya están registrados en la hoja de Excel y esto se hace con la combinación de Python y SQL.

5.3.2.1 Conexión con BBDD en Python

Las tablas básicas son las que contienen la información de los campos que se rellenan, para estos crearemos el código de las ejecuciones INSERT. Luego esto lo realizara el código de Python para poder ir más rápido. Los campos posibles están en hojas distintas de la base de datos en el Excel.

Para poder hacer esto debemos crear un código en python que nos permita conectarnos a la base de datos, en este caso como se esta usando la herramienta de MySQL se usa la librería de mysql.connector. El código para poder hacer esto tiene el siguiente formato:

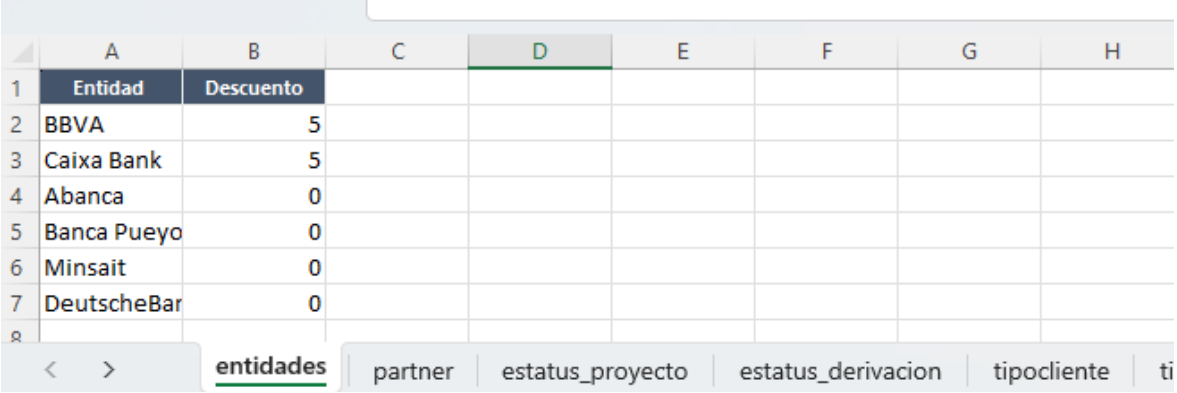
```
import mysql.connector

midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()
cursos.execute(sql,values)
midb.commit()
midb.close()
```

5.3.2.2 Insertar los datos de las tablas simples

La intención de este proyecto es aprender lo máximo posible sobre programación y que todas las tareas por mi simples que sean se hagan con un código y así poder hacerse de forma automática. En la hoja de cálculo hay una serie de tabas sobre la información que se puede seleccionar para algunos campos como entidades, resultados, sector, ect. Por lo tanto, modificando un poco la estructura de donde están las tablas podemos crear una

nueva hoja de calculo donde cada hoja tiene una tabla con las opciones posibles para cada campo como se muestra a continuación.



	A	B	C	D	E	F	G	H
1	Entidad	Descuento						
2	BBVA	5						
3	Caixa Bank	5						
4	Abanca	0						
5	Banca Pueyo	0						
6	Minsait	0						
7	DeutscheBar	0						

entidades | partner | estatus_proyecto | estatus_derivacion | tipocliente | ti

Ilustración 10: Formato de listado.sxls

Teniendo esta tabla creada podemos hacer un código que, con tan solo cambiar un poco, podemos insertar toda información en las tablas simples. El código se hace con Python y usa dos librerías: Pandas y mysql.connector.

```
import pandas as pd
import mysql.connector

midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()

nombre_hoja = 'empleados'
path = 'listado.xlsx'
df = pd.read_excel(path,sheet_name=nombre_hoja)

rows = len(df.axes[0])
col = len(df.axes[1])

for i  in range(rows):
    print(df.iat[i,0])
    sql = f"INSERT INTO empleados (empleados) values (%s);"
    values = (df.iat[i,0],)
    cursos.execute(sql,values)
    midb.commit()
midb.close()
```

En esté código con cambiar el nombre de nombre_hoja podemos ir accediendo a las diferentes hojas y cambiar el comando de INSERT podemos hacer que toda la información se traslade de inmediato. Esto se ha realizado porque hay tablas que tienen mas de 10 campos e ir uno por uno es un proceso muy tedioso que con este código hace que solo se haga un INSERT y el resto los haga de forma automática.

5.3.2.3 Traspaso de datos de los clientes

Este proceso es un poco mas complejo porque cada inserte que tenga una foreign key debe estar escrito por su id no por su nombre. Por ejemplo, si quiero decir que un cliente es del BBVA al hacer el INSERT no puedo decir BBVA, debo decir hacer el INSERT del campo con su id que en este caso seria 1. Este inconveniente nos lleva a crear un código que es corrección de campos, este código evalúa que dato hay que insertar y corrige a su id para que no de error. Ejemplo de una de las muchas funciones que se ha usado es la siguiente:

```
def tip_oportunidad_bien(dato):
    if dato == 'Tramitación masiva':
        return 1
    if dato == 'Asesoramiento personalizado integral':
        return 2
    if dato == 'Asesoramiento personalizado a proyectos':
        return 3
    return 4

def tip_cliente_bien(dato):
    if dato == 'Emprendedor':
        return 1
    if dato == 'Autónomo':
        return 2
    if dato == 'PYME':
        return 3
    if dato == 'Gran Empresa':
        return 4
    if dato == 'Asociaciones empresariales':
        return 5
    if dato == 'ONG':
        return 6
    if dato == 'Centros de investigación':
        return 7
    return 8
```

Cada return hace referencia al id de la entrada real del campo en la hoja de cálculo. Hay muchos registros que por error no tienen completado el campo y por lo tanto hay un return que es para esos casos. Ese id hace referencia siempre a None, implicando que no tenía información anterior.

Una vez que tenemos este archivo con estas funciones podemos hacer el traspaso de información sin que, de error, donde cada campo tiene el id de las tablas simple a las que hacen referencia.

La lógica del archivo .py que se encarga de hacer el traspaso de datos es la siguiente. Primero se conecta como ya hemos visto a la base de datos y también la hoja de calculo con pandas. Segundo es transformar la información de los campos a los id con las funciones creadas, y hacer el código en SQL ya con los id correctos. Tercero es evaluar las fechar y convertirlas a un formato que SQL entiende porque no usa el mismo tipo de registro que Python. Ultimo paso es ejecutar la acción de INSERT. En el código hay un try que se encarga de avisar de los datos que no se han podido pasar porque hay algún fallo.

A continuación, se muestra como es el código de dicho scrip, en este caso se muestra el mas complejo que es el insertar datos en la tabla de proyectos, los otros dos inserts de las tablas complejas se encuentran en el Anexo I.

```
import pandas as pd
import mysql.connector
import aux_funciones

midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()

nombre_hoja = 'Hoja1'
path = 'bbdd.xlsx'
df = pd.read_excel(path,sheet_name=nombre_hoja)

rows = len(df.axes[0])
col = len(df.axes[1])
```

```
#crear tabla solo del cliente
for i  in range(1,rows):

    id_cliente = int(df.iat[i,2])
    tipo_cliente = aux_funciones.tip_cliente_bien(df.iat[i,9])
    partner = aux_funciones.buscar_partner(df.iat[i,28])
    estatus = aux_funciones.buscar_estatus(df.iat[i,32])
    fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))
    tipo_oportunidad = aux_funciones.tip_oportunidad_bien(df.iat[i,1])
    facturacion = aux_funciones.facturacion_bien(df.iat[i,21])
    numero_empleados = aux_funciones.numero_empleados_bien(df.iat[i,22])
    provincia =aux_funciones.provincia(df.iat[i,24])
    ccaa = aux_funciones.ccaa_bien(df.iat[i,26])
    sector = aux_funciones.sector_bien(df.iat[i,19])
    try:
        inversión = int(df.iat[i,15])
    except:
        inversión = 0
    descripción_empresa = df.iat[i,46]
    interes = df.iat[i,17]
    ayuda_aplicada = df.iat[i,18]
    cnae = df.iat[i,20]

    try:
        fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))

        sql = f"INSERT INTO proyecto
(id_cliente,tipo_cliente,partner,estatus,fecha_derivacion,fecha_fin,tipo_
oportunidad,inversión,descripción_empresa,interes,ayuda_aplicada,sector,c
nae,facturacion,numero_empleados,provincia,ccaa) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"
        values =
(id_cliente,tipo_cliente,partner,estatus,fechas[0],fechas[1],tipo_oportun
idad,inversión,descripción_empresa,interes,ayuda_aplicada,sector,cnae,fac
turacion,numero_empleados,provincia,ccaa)
        cursos.execute(sql,values)
    except:
        sql = f"INSERT INTO proyecto
(id_cliente,tipo_cliente,partner,estatus,tipo_oportunidad,inversión,descr
ipción_empresa,interes,ayuda_aplicada,sector,cnae,facturacion,numero_empl
eados,provincia,ccaa) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"
        values =
(id_cliente,tipo_cliente,partner,estatus,tipo_oportunidad,inversión,descr
ipción_empresa,interes,ayuda_aplicada,sector,cnae,facturacion,numero_empl
eados,provincia,ccaa)
        cursos.execute(sql,values)
```

5.3.3 INTERFAZ NOCODE

El objetivo al que se quiere llegar y se ha llega es el siguiente, se explicara para cada una de las acciones que se pueden hacer y como se han creado. Hay dos interfaces, una para la empresa Minsait y otra para cada uno de los partners donde la lógica es un poco distinta ya que solo pueden mostrarse los datos de dicho partner y solo de los que haya aceptado o tenga pendiente de aceptar. La primera imagen muestra la interfaz de la empresa y la segunda de los partners.

Estas interfaces se han realizado con Tkinter, una herramienta que nos permite hacer interfaces y poder costumizarlas voluntad como se ha explicado en la sección de descripción de las tecnologías.

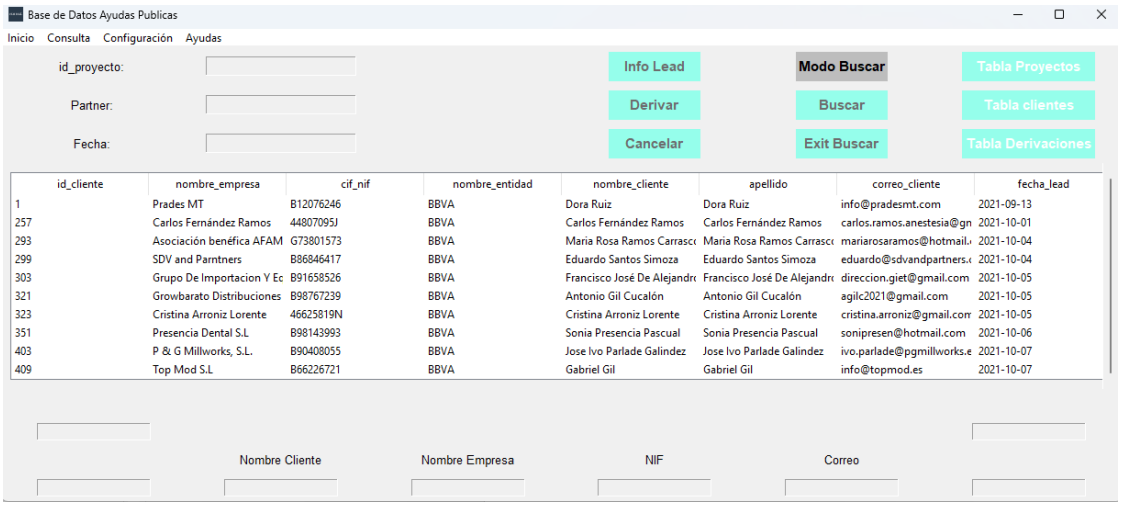


Ilustración 11: Diseño de la interfaz de la empresa

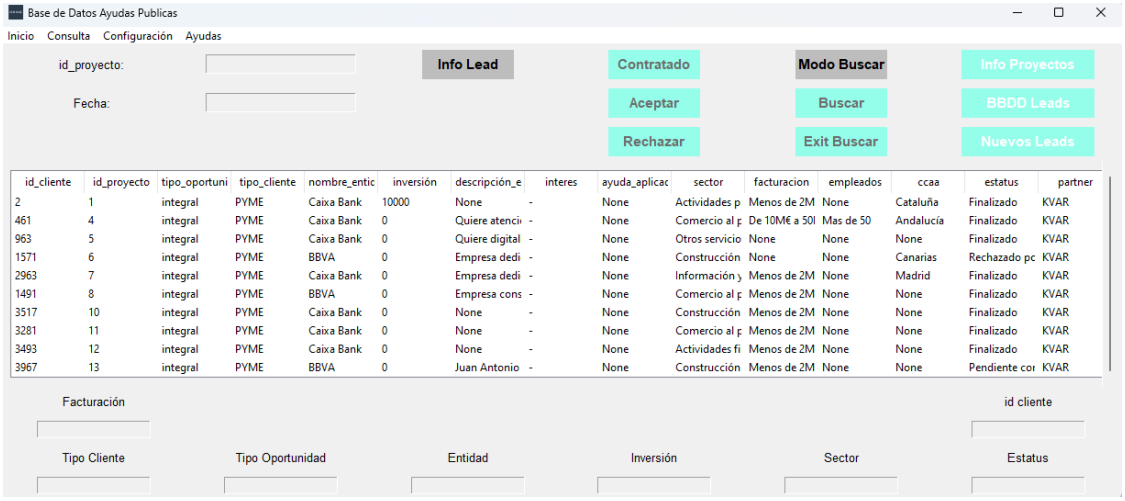


Ilustración 12: Diseño de la interfaz de los partners

5.3.3.1 Distribución de la interfaz de la empresa

La imagen que se muestra a continuación muestra todas las acciones que se pueden hacer y qué información se puede mostrar con cada botón.

5.3.3.2 Distribución de la interfaz de los partners

La imagen que se muestra a continuación muestra todas las acciones que se pueden hacer y qué información se puede mostrar con cada botón.

5.3.4 CREACIÓN INTERFAZ

En la creación de las interfaces, hay cosas en común para como los filtros y el tabal donde se muestra los datos que se han filtrado. Las diferencias son los datos que se muestran y que tablas se muestran. Se explicará las partes comunes en esta sección y luego las especificaciones se harán aparte.

La interfaz está compuesta por cuatro secciones:

1. Sección de actualización
2. Botones de acciones
3. Tabla donde se muestra los datos
4. Filtros para hacer búsqueda de clientes

5.3.4.1 Sección de actualización

La sección de actualización consta de dos herramientas de tkinter, Entry y Lable. La entrada esta hecha para poder mostrar y recoger los datos que se quieran actualizar y el lable se usa para poder dejar claro al usuario a que se refiere el campo.

Esto se hace dentro de la función filtros, ya que consiste en etiquetas y entradas como en el campo de los filtros. Se han situado en otro lugar para que no se confundan con los filtros. Esto campos se activan y se desactivan según los botones que seleccionen.

A continuación, se muestra ejemplo de un campo de actualización:

```
#creación de labels para metar campos
self.label_id_proyecto = tk.Label(self,text='id_proyecto: ',width=20)
self.label_id_proyecto.config(font= ('Arial',10))
self.label_id_proyecto.grid(row=0,column=0,padx=5,pady=5)

#crear entradas
self.mi_id_proyecto = tk.StringVar()
self.entry_id_proyecto = tk.Entry(self,textvariable=self.mi_id_proyecto)
self.entry_id_proyecto.config(width=20,font= ('Arial',11))
self.entry_id_proyecto.grid(row=0,column=1,padx=5,pady=5)
```

Este código crea una sección de interfaz gráfica de usuario (GUI) con dos componentes principales: un label y una entrada (entry). Esto se utiliza para permitir al usuario ingresar datos o información relacionada con un "id_proyecto". Vamos a explicar cada parte del código:

1. Creación del Label:
    - Se define un label denominado self.label_id_proyecto con el texto "id_proyecto: " y una anchura de 20 caracteres (width=20).
    - Se configura la fuente del label con 'Arial' y tamaño de fuente 10 (font= ('Arial',10)).
    - Se coloca el label en la posición de la cuadrícula especificada (grid(row=0, column=0, padx=5, pady=5)), en la fila 0 y columna 0 con márgenes horizontales y verticales de 5 píxeles.
2. Creación de la Entrada (Entry):
    - Se crea una variable llamada self.mi_id_proyecto de tipo tk.StringVar() que se utilizará para almacenar el valor ingresado por el usuario en la entrada.
    - Se define un campo de entrada (self.entry_id_proyecto) vinculado a la variable self.mi_id_proyecto para que cualquier valor ingresado en la entrada se almacene en la variable.
    - Se configura el ancho de la entrada en 20 caracteres (width=20) y se establece la fuente con 'Arial' y tamaño de fuente 11 (font= ('Arial',11)).
    - Se coloca la entrada en la posición de la cuadrícula especificada (grid(row=0, column=1, padx=5, pady=5)), en la fila 0 y columna 1 con márgenes horizontales y verticales de 5 píxeles.

5.3.4.2 Sección Botones

La sección de los botones nos permite activar comandos mediante funciones. Para cada interfaz los botones tienen diferentes acciones. Todos los botones tienen una finalidad ya sea de actualizar, mostrar o activas otros botones y campos de filtro. A continuación, se

muestra como se ha creado unos de los botones y a que función (comando) esta enlazado, para que a la hora de pulsarlo se ejecute una tarea.

```
#Botones activos
self.boton_proyecto = tk.Button(self,text='Tabla
Proyectos',command=self.select_tabla_proyecto)
self.boton_proyecto.config(width=15,font= ('Arial',11,'bold'),fg =
'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat')
self.boton_proyecto.grid(row=0,column=5,padx=5,pady=5)
```

Este código crea un botón en una interfaz gráfica de usuario (GUI) utilizando la biblioteca Tkinter. A continuación, se explica cada parte del código:

1. Creación del Botón:
   - Se define un botón llamado self.boton_proyecto con el texto "Tabla Proyectos".
   - El botón está asociado a un comando o función llamada self.select_tabla_proyecto, que se ejecutará cuando el botón sea clicado. Es decir, cuando el usuario haga clic en el botón "Tabla Proyectos", la función self.select_tabla_proyecto será llamada.
2. Configuración del Botón:
   - Se configura el ancho del botón en 15 caracteres (width=15).
   - Se establece la fuente del texto del botón con 'Arial' y tamaño de fuente 11 (font= ('Arial',11,'bold')).
   - El color del texto del botón se establece en blanco (fg = 'white') y el color de fondo se establece en '#95FEEB' (bg = '#95FEEB').
   - El cursor del ratón se cambia al estilo de "mano" cuando se pasa por encima del botón (cursor='hand2').
   - El color de fondo activo del botón se establece en '#BDBDBD' (activebackground='#BDBDBD'), lo que indica cómo se ve el botón cuando se hace clic en él.
   - El estilo de relieve del botón se establece en 'flat', lo que significa que el botón no tendrá un aspecto en relieve (relief='flat').

3. Posicionamiento del Botón:
    - Finalmente, el botón se coloca en la posición de la cuadrícula especificada (grid(row=0, column=5, padx=5, pady=5)), en la fila 0 y columna 5 con márgenes horizontales y verticales de 5 píxeles.

El comando de select table por ejemplo lo que hace es llamar a una función de treeview que nos proporciona la tabla de proyectos, dentro de esta función se llama mediante mysql.connector a seleccionar la tabla de proyecto y trasladarla a la tabla de interfaz.

5.3.4.3 Sección de Tabla

Esta situada en la parte central y lo que se muestra es lo que se ha buscado en la base de datos. Dependiendo del botón al que se pulse, se mostrara una tabla o otra, por ejemplo, cuando se pulsa el botón de tabla proyecto se ejecuta el siguientes código SQL:

```
SELECT
  proyecto.id_cliente,
  proyecto.id_proyecto,
  tipo_oportunidad.tipo_oportunidad,
  tipo_cliente.tipo_cliente,
  entidad.nombre_entidad,
  proyecto.inversión,
  proyecto.descripción_empresa,
  royecto.interes,
  proyecto.ayuda_aplicada,
  sector.sector,
  facturacion.facturacion,
  empleados.empleados,
  ccaa.ccaa,
  estatus.estatus,
  partner.partner
FROM proyecto
JOIN tipo_oportunidad ON proyecto.tipo_oportunidad =
tipo_oportunidad.id_oportunidad
JOIN tipo_cliente ON proyecto.tipo_cliente = tipo_cliente.id_tipo
JOIN sector ON proyecto.sector = sector.id_sector
JOIN ccaa ON proyecto.ccaa = ccaa.id_ccaa
JOIN facturacion ON proyecto.facturacion = facturacion.id_facturacion
JOIN empleados ON proyecto.numero_empleados = empleados.id_empleados
JOIN cliente ON proyecto.id_cliente = cliente.id_cliente
JOIN entidad ON cliente.entidad = entidad.id_entidad
JOIN estatus ON proyecto.estatus = estatus.id_estatus
JOIN partner ON partner.id_partner = proyecto.partner
```

El código que muestra la tabla esta dentro de la siguiente función:

```
    def tabla_cliente(self,tabla,busqueda,filtros = 'None'):

        conexion = Conexion_DB()

        #recuperar lista

        self.lista_datos = listar(tabla,busqueda,filtros)
        self.encabezado = columnas_tabla(tabla,busqueda,filtros)

        try:
            self.tabla.destroy()
            self.tabla =
ttk.Treeview(self,columns=self.encabezado,selectmode='browse',show='headi
ngs')

self.tabla.grid(row=4,column=0,columnspan=6,sticky='nse',padx=10,pady=10)
            self.scroll_y =
ttk.Scrollbar(self,orient='vertical',command=self.tabla.yview)
            self.scroll_y.grid(row=4,column=5,sticky='nse',padx=5)
        except:
            self.tabla =
ttk.Treeview(self,columns=self.encabezado,selectmode='browse',show='headi
ngs')

self.tabla.grid(row=4,column=0,columnspan=6,sticky='nse',padx=10,pady=10)
            self.scroll_y =
ttk.Scrollbar(self,orient='vertical',command=self.tabla.yview)
            self.scroll_y.grid(row=4,column=5,sticky='nse',padx=5)

        dim_col = int(1200/len(self.encabezado))

        for i in self.encabezado:
            self.tabla.column(i,anchor='w',width=dim_col )
            self.tabla.heading(i,text=i)

        for i in self.lista_datos:
            self.tabla.insert('','end',iid=i[0],text=i[0],values=list(i))

        contar = len(self.lista_datos)
```

Este código define una función llamada tabla_cliente dentro de una clase, que se utiliza para crear y mostrar una tabla de datos en una interfaz gráfica de usuario (GUI). Aquí está la explicación de cada parte del código:

1. Establecer una conexión a la base de datos:
    - Se crea una instancia de la clase Conexion_DB, que se asume que contiene la lógica para establecer una conexión a la base de datos.
2. Recuperar datos de la base de datos:
    - La función llama a las funciones listar y columnas_tabla, que obtienen los datos de la tabla y los encabezados de columna, respectivamente. Estos datos se almacenan en las variables self.lista_datos y self.encabezado, respectivamente.
3. Creación o actualización de la tabla en la GUI:
    - Se intenta destruir la tabla actual (si existe) utilizando self.tabla.destroy(). Luego, se crea una nueva instancia de ttk.Treeview llamada self.tabla con las columnas y el modo de selección especificados (selectmode='browse').
    - La tabla se coloca en la posición de la cuadrícula especificada (grid(row=4, column=0, columnspan=6, sticky='nse', padx=10, pady=10)), en la fila 4 y columnas 0 a 5, con márgenes horizontales y verticales de 10 píxeles.
    - Se crea una barra de desplazamiento vertical (ttk.Scrollbar) llamada self.scroll_y para desplazarse por la tabla.
    - La barra de desplazamiento se coloca en la posición de la cuadrícula especificada (grid(row=4, column=5, sticky='nse', padx=5)), en la fila 4 y columna 5, con un margen horizontal de 5 píxeles.
4. Configuración de las columnas de la tabla:
    - El ancho de las columnas se calcula dividiendo 1200 (ancho total deseado de la tabla) por la cantidad de encabezados de columna (dim_col = int(1200/len(self.encabezado))).
    - Se configura cada columna de la tabla con su respectivo ancho y el encabezado de columna correspondiente (self.tabla.column(i, anchor='w', width=dim_col) y self.tabla.heading(i, text=i)).
5. Rellenar la tabla con datos:
    - Se utiliza un bucle for para recorrer los datos de la lista self.lista_datos.

- Para cada conjunto de datos en self.lista_datos, se inserta una nueva fila en la tabla con el identificador iid igual al primer elemento de ese conjunto (self.tabla.insert('', 'end', iid=i[0], text=i[0], values=list(i))).
- La fila contiene los datos del conjunto (values=list(i)), mientras que el primer elemento del conjunto se usa también como texto visible en la tabla.

6. Contar la cantidad de datos en la tabla:
   - La variable contar contiene la cantidad de filas de datos en la tabla (contar = len(self.lista_datos)).

5.3.4.4 Sección de filtros

Al igual que la sección de actualización, solo se usan las herramientas de entry y lable, que nos permite avisar al usuario de que campo se trata y luego un campo donde poder insertar la información que se quiere buscar. A continuación, se muestra un ejemplo de un campo de búsqueda.

```
#Label
self.label_tipo_cliente = tk.Label(self,text='Tipo Cliente',width=20)
self.label_tipo_cliente.config(font= ('Arial',10))
self.label_tipo_cliente.grid(row=7,column=0,padx=5,pady=5)

#Desplegables
self.mi_tipo_cliente = tk.StringVar()
self.desple_tipo_cliente =
tk.Entry(self,width=20,textvariable=self.mi_tipo_cliente)
self.desple_tipo_cliente.grid(row=8,column=0,padx=5,pady=5)
```

Este código crea un conjunto de elementos relacionados con la manipulación de datos sobre el tipo de cliente en una interfaz gráfica de usuario (GUI) usando la biblioteca Tkinter. A continuación, se explica cada parte del código:

1. Creación del Label:
   - Se define un label denominado self.label_tipo_cliente con el texto "Tipo Cliente" y una anchura de 20 caracteres (width=20).

- Se configura la fuente del label con 'Arial' y tamaño de fuente 10 (font=('Arial',10)).
- El label se coloca en la posición de la cuadrícula especificada (grid(row=7, column=0, padx=5, pady=5)), en la fila 7 y columna 0 con márgenes horizontales y verticales de 5 píxeles.

2. Creación del Desplegable (Entry):
   - Se crea una variable llamada self.mi_tipo_cliente de tipo tk.StringVar(), que se utilizará para almacenar el valor ingresado por el usuario en el desplegable.
   - Se define un campo de entrada (self.desple_tipo_cliente) vinculado a la variable self.mi_tipo_cliente para que cualquier valor ingresado en el desplegable se almacene en la variable.
   - Se configura el ancho del desplegable en 20 caracteres (width=20).
   - El desplegable se coloca en la posición de la cuadrícula especificada (grid(row=8, column=0, padx=5, pady=5)), en la fila 8 y columna 0 con márgenes horizontales y verticales de 5 píxeles.

5.3.4.5 Lógica de las acciones de la interfaz

La interfaz consta de dos herramientas, una para los partners y otra para empresa. Se hará la explicación primero de la empresa y luego el de los partners. Se explicará que es lo que hace cada botón que se muestra en pantalla y que se habilita, y como las dos herramientas están relacionadas.

LOGICA DE BOTONES PARA EMPRESA.

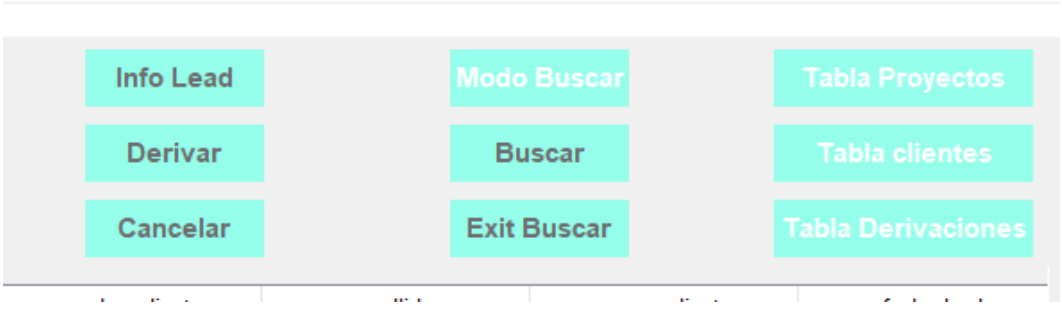
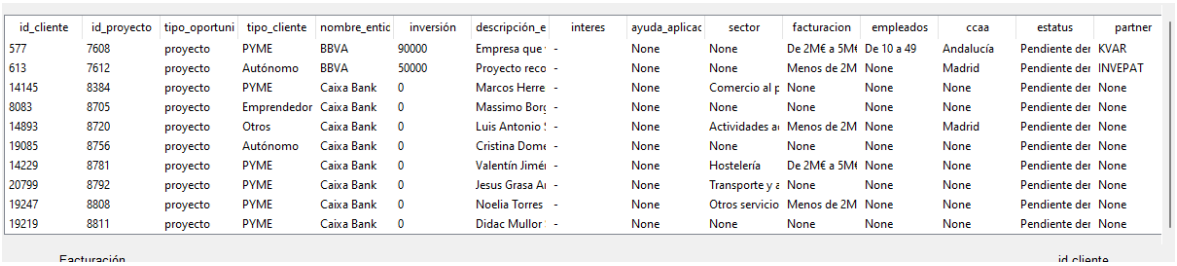


Ilustración 13: Botones interfaz empresa

Los botones que están habilitados son los que están con las letras en blanco, y así es como carga la pantalla desde el principio. Los tres botones de la derecha se encargan de enseñar las tres tablas que se mencionan en el botón.

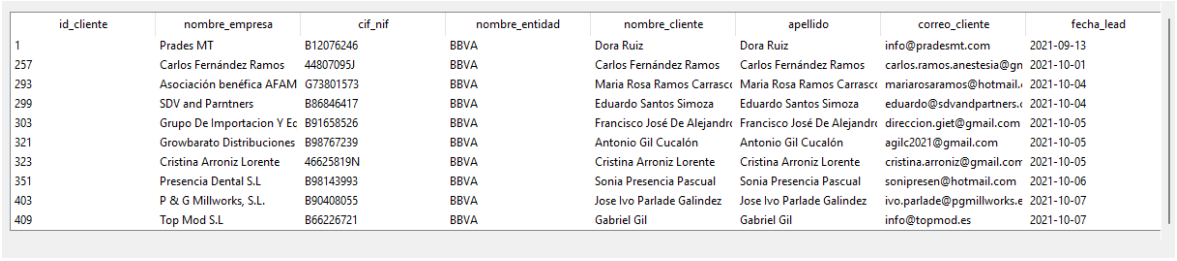
1. Tabla proyecto: esta nos enseña la información de cada proyecto, la inversión, tipo de cliente, tamaño de empresa, interés, facturación, etc.



id_cliente	id_proyecto	tipo_oportuni	tipo_cliente	nombre_entic	inversión	descripción_e	interes	ayuda_aplicac	sector	facturacion	empleados	ccaa	estatus	partner
577	7608	proyecto	PYME	BBVA	90000	Empresa que	-	None	None	De 2M€ a 5M	De 10 a 49	Andalucía	Pendiente de	KVAR
613	7612	proyecto	Autónomo	BBVA	50000	Proyecto reco	-	None	None	Menos de 2M	None	Madrid	Pendiente de	INVEPAT
14145	8384	proyecto	PYME	Caixa Bank	0	Marcos Herre	-	None	Comercio al p	None	None	None	Pendiente de	None
8083	8705	proyecto	Emprendedor	Caixa Bank	0	Massimo Bor	-	None	None	None	None	None	Pendiente de	None
14893	8720	proyecto	Otros	Caixa Bank	0	Luis Antonio	-	None	Actividades a	Menos de 2M	None	Madrid	Pendiente de	None
19085	8756	proyecto	Autónomo	Caixa Bank	0	Cristina Dome	-	None	None	None	None	None	Pendiente de	None
14229	8781	proyecto	PYME	Caixa Bank	0	Valentín Jiméi	-	None	Hostelería	De 2M€ a 5M	None	None	Pendiente de	None
20799	8792	proyecto	PYME	Caixa Bank	0	Jesus Grasa Al	-	None	Transporte y a	None	None	None	Pendiente de	None
19247	8808	proyecto	PYME	Caixa Bank	0	Noelia Torres	-	None	Otros servicio	Menos de 2M	None	None	Pendiente de	None
19219	8811	proyecto	PYME	Caixa Bank	0	Didac Mullor	-	None	None	None	None	None	Pendiente de	None

Ilustración 14: Representación de tabla proyecto en la interfaz

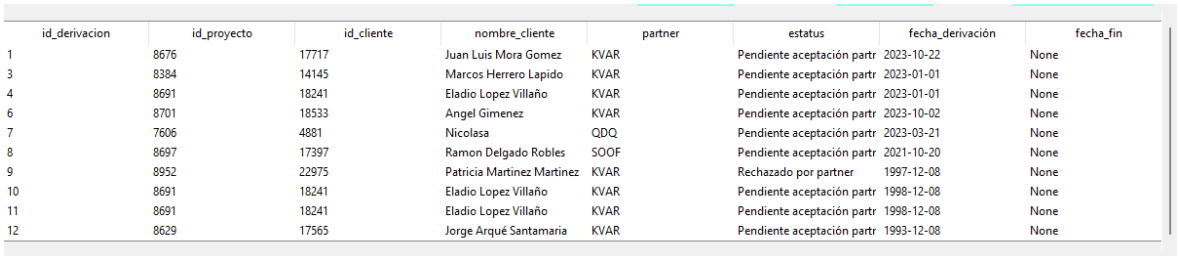
2. Tabla de cliente: se recogen los datos personales del cliente, nombre, correo, número de teléfono, etc. La información para poder contactar con ellos.



id_cliente	nombre_empresa	cif_nif	nombre_entidad	nombre_cliente	apellido	correo_cliente	fecha_lead
1	Prades MT	B12076246	BBVA	Dora Ruiz	Dora Ruiz	info@pradesmt.com	2021-09-13
257	Carlos Fernández Ramos	44807095J	BBVA	Carlos Fernández Ramos	Carlos Fernández Ramos	carlos.ramos.anestesia@gn	2021-10-01
293	Asociación benéfica AFAM	G73801573	BBVA	Maria Rosa Ramos Carrasc	Maria Rosa Ramos Carrasc	mariarosaramos@hotmail.(	2021-10-04
299	SDV and Parntners	B86846417	BBVA	Eduardo Santos Simoza	Eduardo Santos Simoza	eduardo@sdvandpartners.(	2021-10-04
303	Grupo De Importacion Y Ec	B91658526	BBVA	Francisco José De Alejandr	Francisco José De Alejandr	direccion.giet@gmail.com	2021-10-05
321	Growbarato Distribuciones	B98767239	BBVA	Antonio Gil Cucalón	Antonio Gil Cucalón	agilc2021@gmail.com	2021-10-05
323	Cristina Arroniz Lorente	46625819N	BBVA	Cristina Arroniz Lorente	Cristina Arroniz Lorente	cristina.arroniz@gmail.com	2021-10-05
351	Presencia Dental S.L	B98143993	BBVA	Sonia Presencia Pascual	Sonia Presencia Pascual	sonipresen@hotmail.com	2021-10-06
403	P & G Millworks, S.L.	B90408055	BBVA	Jose Ivo Parlade Galindez	Jose Ivo Parlade Galindez	ivo.parlade@pgmillworks.e	2021-10-07
409	Top Mod S.L	B66226721	BBVA	Gabriel Gil	Gabriel Gil	info@topmod.es	2021-10-07

Ilustración 15:Representación de tabla clientea en la interfaz

3. Tabla de derivación: nos permite ver a quien se le esta derivando al lead el tiempo que ha estado en derivación y cuantas veces se ha tenido que derivar.



id_derivacion	id_proyecto	id_cliente	nombre_cliente	partner	estatus	fecha_derivación	fecha_fin
1	8676	17717	Juan Luis Mora Gomez	KVAR	Pendiente aceptación partr	2023-10-22	None
3	8384	14145	Marcos Herrero Lapido	KVAR	Pendiente aceptación partr	2023-01-01	None
4	8691	18241	Eladio Lopez Villaño	KVAR	Pendiente aceptación partr	2023-01-01	None
6	8701	18533	Angel Gimenez	KVAR	Pendiente aceptación partr	2023-10-02	None
7	7606	4881	Nicolasa	QDQ	Pendiente aceptación partr	2023-03-21	None
8	8697	17397	Ramon Delgado Robles	SOOF	Pendiente aceptación partr	2021-10-20	None
9	8952	22975	Patricia Martinez Martinez	KVAR	Rechazado por partner	1997-12-08	None
10	8691	18241	Eladio Lopez Villaño	KVAR	Pendiente aceptación partr	1998-12-08	None
11	8691	18241	Eladio Lopez Villaño	KVAR	Pendiente aceptación partr	1998-12-08	None
12	8629	17565	Jorge Arqué Santamaria	KVAR	Pendiente aceptación partr	1993-12-08	None

Ilustración 16: Representación de tabla derivación en la interfaz

El único botón aparte de las tablas que este activo siempre es el botón de búsqueda, en todas las tablas se pueden hacer búsquedas filtradas. Dependiendo de la tabla en la que se está habrá unos filtros u otro. Una vez se pulsa el botón de modo busqeuda, se activa exit busqeuda para dejar de buscar y limpiar los campos. El otro botón que se activa es el de buscar que nos permite aplicar una seria de filtros.

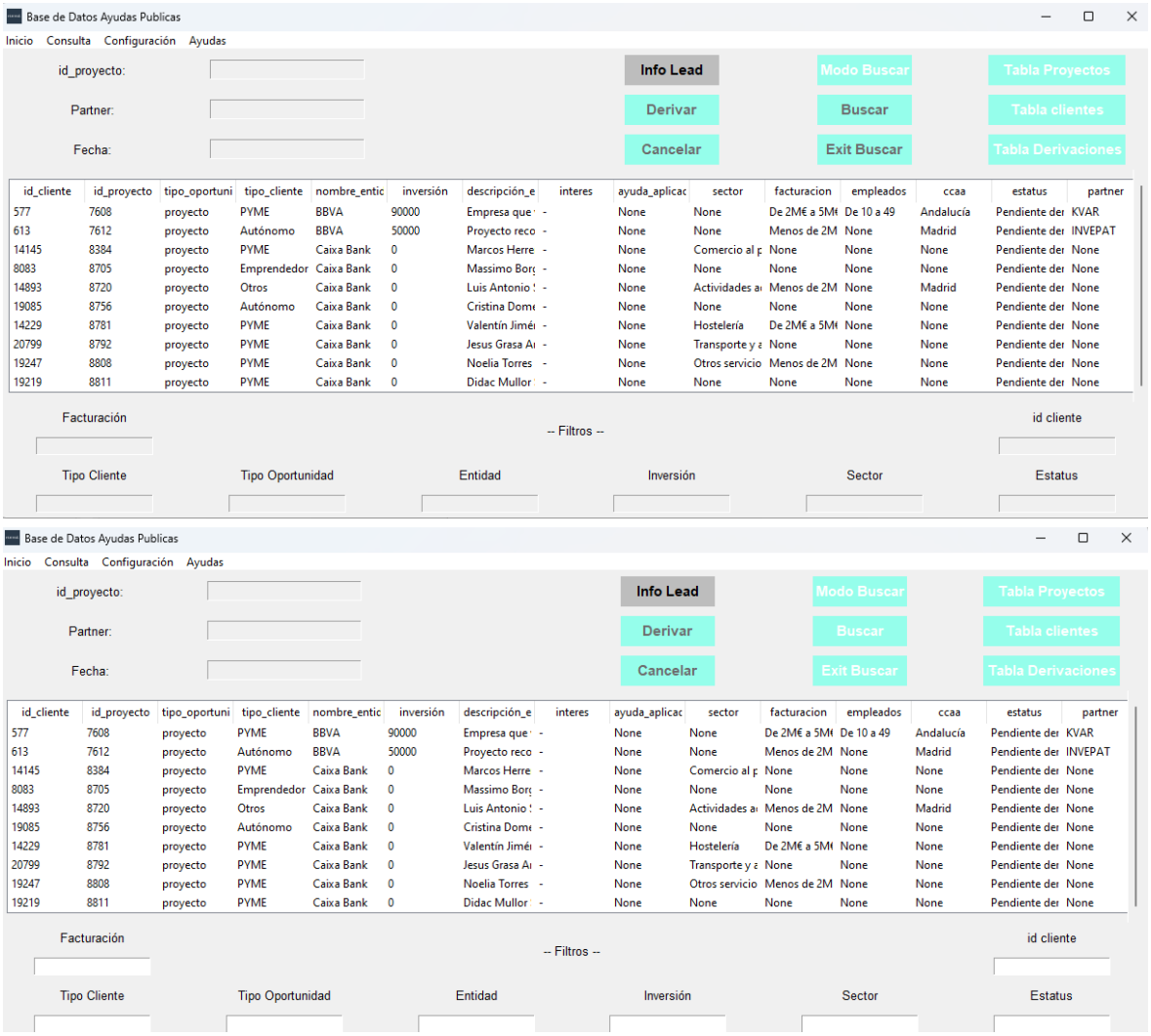


Ilustración 17: Activación del modo busqueda

Como se puede ver en la imagen los campos están desactivados y una vez que se pulsa el botón de modo búsqueda se activan los dos botones de debajo y los campos se habilitan para poder escribir en ellos y hacer búsquedas. A continuación, se hace un ejemplo de búsqueda.

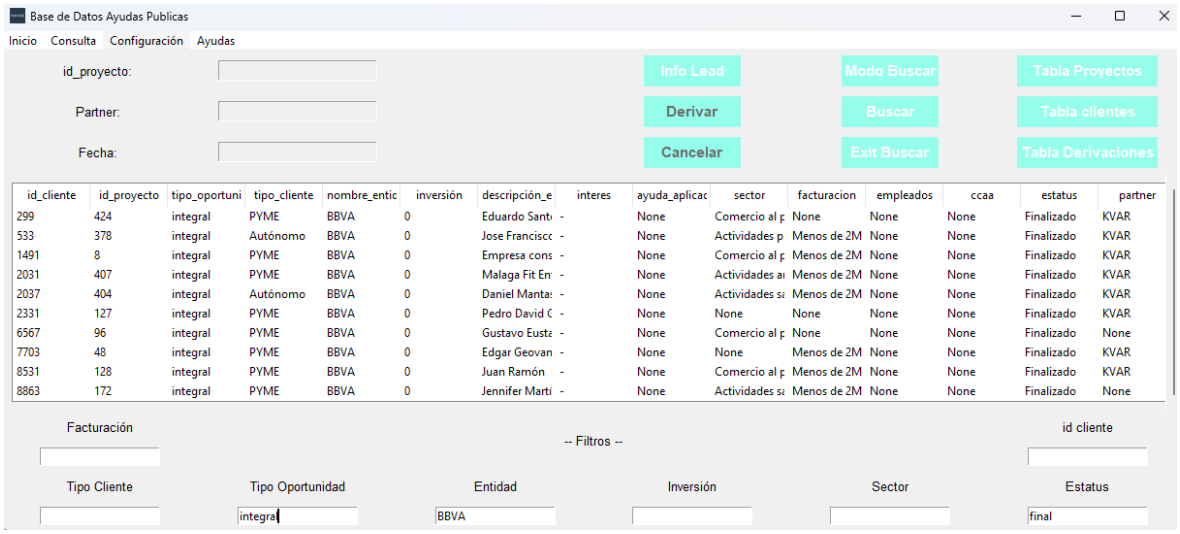
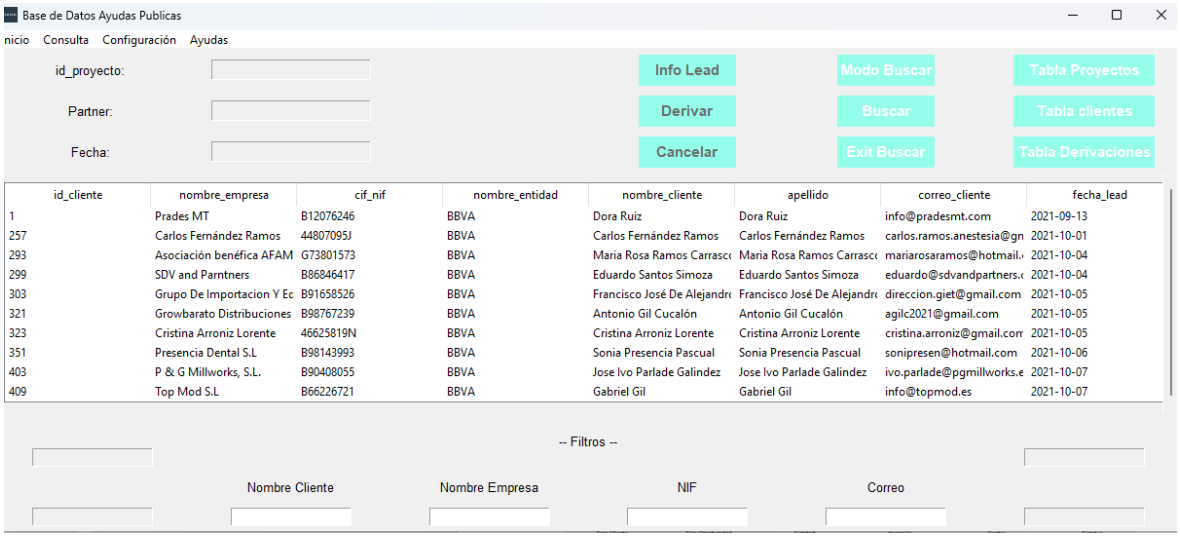


Ilustración 18: Ejemplo de búsqueda en tabla proyecto

Esto se puede hacer en las tres tablas, pero en cada una de ellas se pueden filtrar cosas diferentes, como se muestra en las imágenes en la sección de debajo



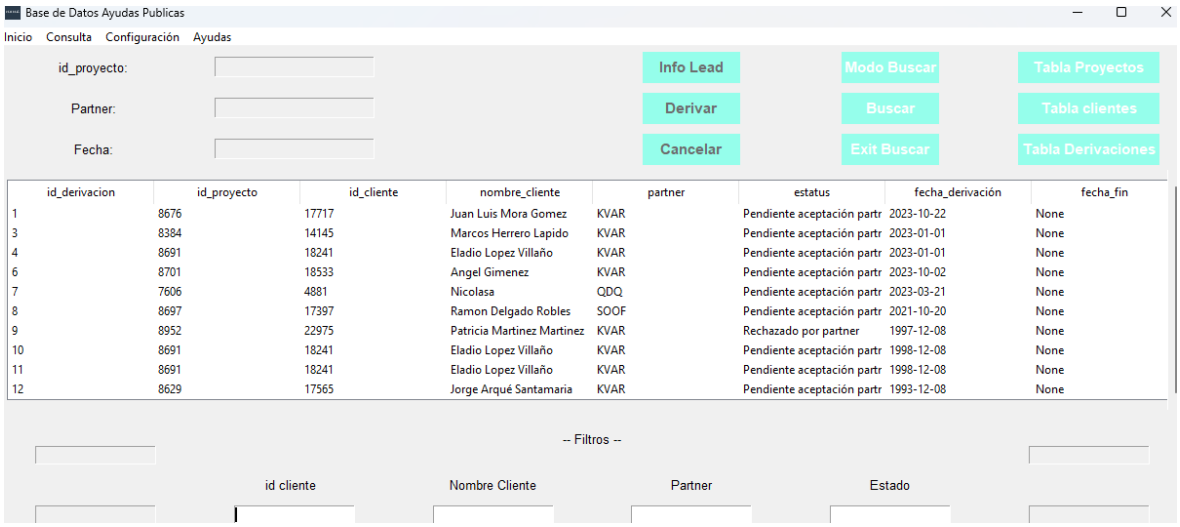


Ilustración 19: Activación de campos para la búsqueda en el resto de las tablas

Actualizar en cambio solo se puede usar en la tabla de proyectos que donde se evalúa a que partner hay que derivar. Esto implica que ese botón solo estará activo una vez que hagamos click en la tabla de proyecto como se puede en la sección de abajo.

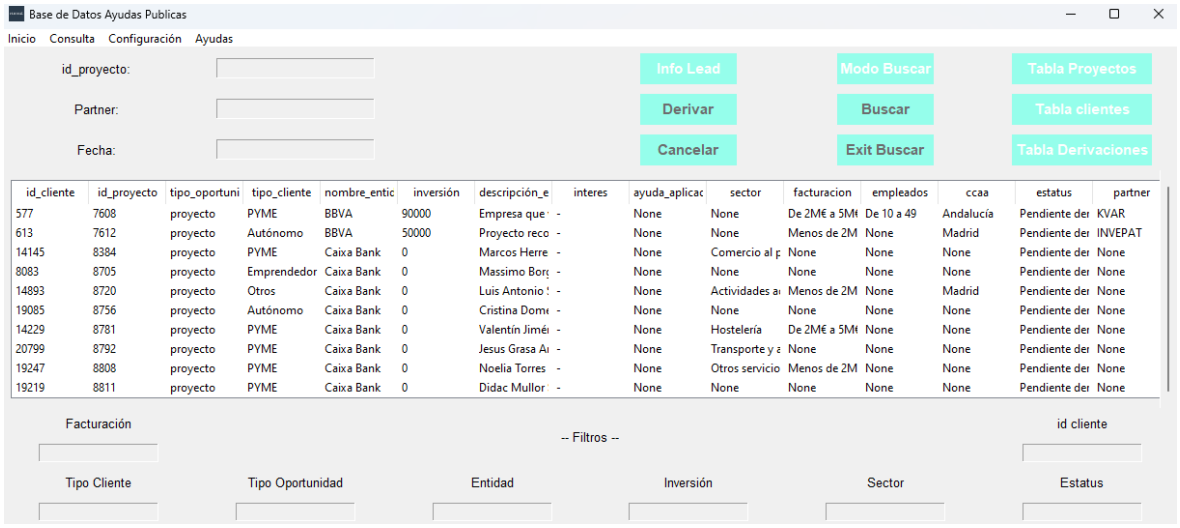


Ilustración 20: Botones que se activan con el botón tabla proyecto

Una vez veamos o filtremos un lead que queremos derivar es tan sencillo como pinchar en la línea donde esta el lead y darle al botón de info lead. Esto habilitar el botón de derivar y el de cancelar.

El botón de derivar lo que hacer es obtener la información de los campos y hacer una nueva derivación a un partner, para esto hace falta completar los campos de partner y fecha. Esto hará que se actualizan todos los campos, los de partners, fecha y el estado del proyecto.

La secuencia es que se crea un registro nuevo en la tabla de derivación para que quede registrado que se ha derivada y a que partner, pero también actualiza el estado del proyecto para que sepamos que estamos esperando respuesta del partner.

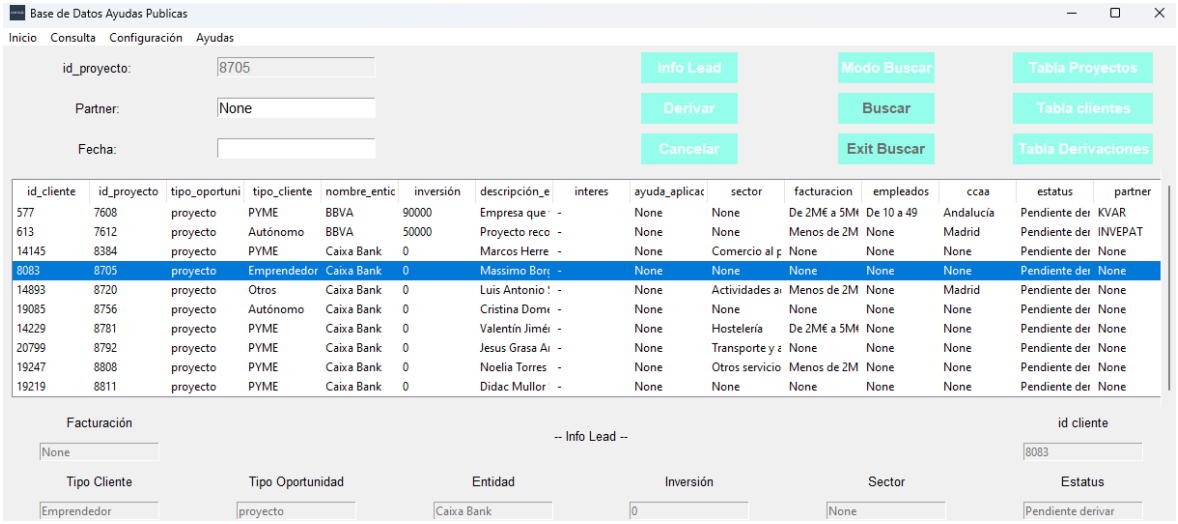


Ilustración 21: Ejemplo de derivación

Esta es la lógica de los botones que activan y que hacen, código completo de como este hecho que funciones activas y los códigos SQL que ejecutan están adjuntos en el Anexo I.

LOGICA DE BOTONES PARA EMPRESA.

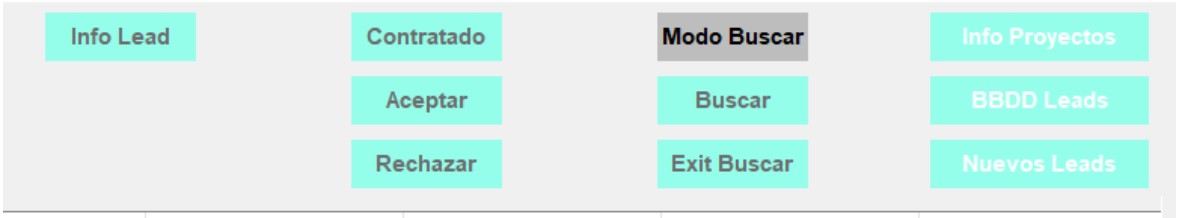


Ilustración 22: Botones de la interfaz partners

Los botones que están habilitados son los que están con las letras en blanco, y así es como carga la pantalla desde el principio. Los tres botones de la derecha se encargan de enseñar las tres tablas que se mencionan en el botón. La única diferencia es que es este caso solo se muestran los leads que le correspondan al partner

1. Tabla info proyecto: esta nos enseña la información de cada proyecto, la inversión, tipo de cliente, tamaño de empresa, interés, facturación, etc.

id_cliente	id_proyecto	tipo_oportuni	tipo_cliente	nombre_entic	inversión	descripción_e	interes	ayuda_aplicac	sector	facturacion	empleados	ccaa	estatus	partner
577	7608	proyecto	PYME	BBVA	90000	Empresa que	-	None	None	De 2M€ a 5M	De 10 a 49	Andalucía	Pendiente de	KVAR
613	7612	proyecto	Autónomo	BBVA	50000	Proyecto reco	-	None	None	Menos de 2M	None	Madrid	Pendiente de	INVEPAT
14145	8384	proyecto	PYME	Caixa Bank	0	Marcos Herre	-	None	Comercio al p	None	None	None	Pendiente de	None
8083	8705	proyecto	Emprendedor	Caixa Bank	0	Massimo Bor	-	None	None	None	None	None	Pendiente de	None
14893	8720	proyecto	Otros	Caixa Bank	0	Luis Antonio	-	None	Actividades a	Menos de 2M	None	Madrid	Pendiente de	None
19085	8756	proyecto	Autónomo	Caixa Bank	0	Cristina Dome	-	None	None	None	None	None	Pendiente de	None
14229	8781	proyecto	PYME	Caixa Bank	0	Valentín Jiméi	-	None	Hostelería	De 2M€ a 5M	None	None	Pendiente de	None
20799	8792	proyecto	PYME	Caixa Bank	0	Jesus Grasa A	-	None	Transporte y a	None	None	None	Pendiente de	None
19247	8808	proyecto	PYME	Caixa Bank	0	Noelia Torres	-	None	Otros servicio	Menos de 2M	None	None	Pendiente de	None
19219	8811	proyecto	PYME	Caixa Bank	0	Didac Mullor	-	None	None	None	None	None	Pendiente de	None

Ilustración 23: Representación de tabla proyecto en la interfaz

2. Tabla de cliente: se recogen los datos personales del cliente, nombre, correo, número de teléfono, etc. La información para poder contactar con ellos. Para salir en esta tabla tiene que ser leads del partner y además estar en estado de aceptado

id_cliente	nombre_empresa	cif_nif	nombre_entidad	nombre_cliente	apellido	correo_cliente	fecha_lead
1	Prades MT	B12076246	BBVA	Dora Ruiz	Dora Ruiz	info@pradesmt.com	2021-09-13
257	Carlos Fernández Ramos	44807095J	BBVA	Carlos Fernández Ramos	Carlos Fernández Ramos	carlos.ramos.anestesia@gn	2021-10-01
293	Asociación benéfica AFAM	G73801573	BBVA	Maria Rosa Ramos Carrasc	Maria Rosa Ramos Carrasc	mariarosaramos@hotmail.	2021-10-04
299	SDV and Parntners	B86846417	BBVA	Eduardo Santos Simoza	Eduardo Santos Simoza	eduardo@sdvandpartners.	2021-10-04
303	Grupo De Importacion Y Ec	B91658526	BBVA	Francisco José De Alejandr	Francisco José De Alejandr	direccion.giet@gmail.com	2021-10-05
321	Growbarato Distribuciones	B98767239	BBVA	Antonio Gil Cucalón	Antonio Gil Cucalón	agilc2021@gmail.com	2021-10-05
323	Cristina Arroniz Lorente	46625819N	BBVA	Cristina Arroniz Lorente	Cristina Arroniz Lorente	cristina.arroniz@gmail.com	2021-10-05
351	Presencia Dental S.L	B98143993	BBVA	Sonia Presencia Pascual	Sonia Presencia Pascual	sonipresen@hotmail.com	2021-10-06
403	P & G Millworks, S.L.	B90408055	BBVA	Jose Ivo Parlade Galindez	Jose Ivo Parlade Galindez	ivo.parlade@pgmillworks.e	2021-10-07
409	Top Mod S.L	B66226721	BBVA	Gabriel Gil	Gabriel Gil	info@topmod.es	2021-10-07

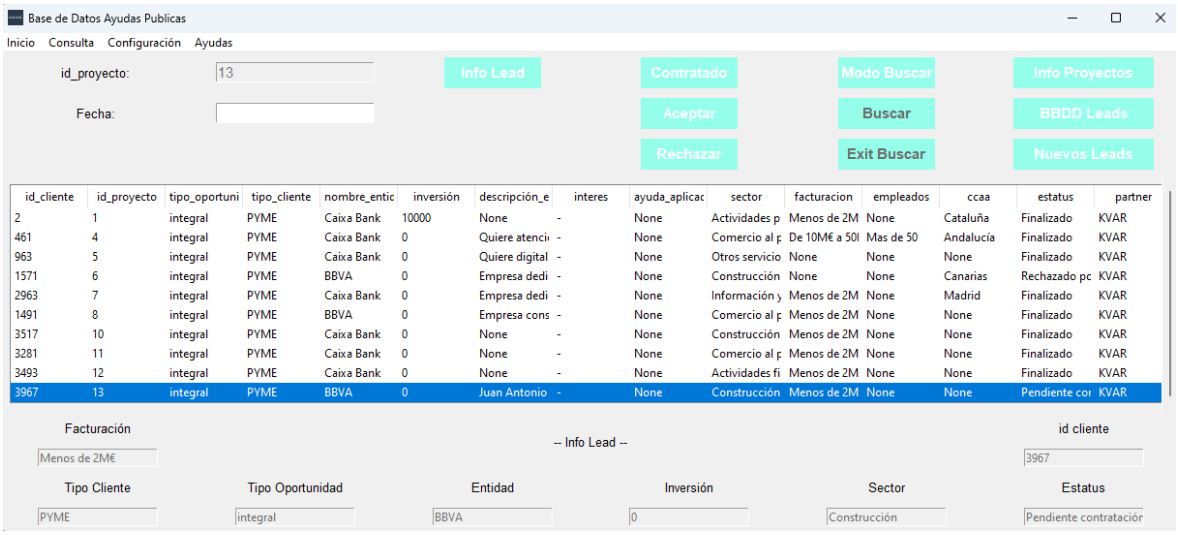
Ilustración 24:Representación de tabla clientea en la interfaz

3. Tabla de nuevos leads: nos permite ver a quien se le está derivando al lead el tiempo que ha estado en derivación y cuantas veces se ha tenido que derivar.

id_derivacion	id_proyecto	id_cliente	partner	estatus	fecha_derivación	fecha_fin
1	8676	17717	KVAR	Pendiente aceptación partner	2023-10-22	None
3	8384	14145	KVAR	Pendiente aceptación partner	2023-01-01	None
4	8691	18241	KVAR	Pendiente aceptación partner	2023-01-01	None
6	8701	18533	KVAR	Pendiente aceptación partner	2023-10-02	None
9	8952	22975	KVAR	Rechazado por partner	1997-12-08	None
10	8691	18241	KVAR	Pendiente aceptación partner	1998-12-08	None
11	8691	18241	KVAR	Pendiente aceptación partner	1998-12-08	None
12	8629	17565	KVAR	Pendiente aceptación partner	1993-12-08	None
13	8703	18177	KVAR	Pendiente aceptación partner	1898-01-01	None
14	8911	17481	KVAR	Rechazado por partner	2050-12-03	2023-10-22

Ilustración 25: Representación de tabla derivación en la interfaz

la lógica del modo buscar es la misma que para la interfaz de empresa, por lo tanto, explicarla no será necesario. La dinámica del botón info leads es parecida, si seleccionas una fila puedes obtener toda información necesaria y se activan los botones que están justo a su derecha. Estos botones lo que nos permiten es aceptar el lead, rechazarlo y confirmar que han firmado contrato.



id_cliente	id_proyecto	tipo_oportuni	tipo_cliente	nombre_entic	inversión	descripción_e	interes	ayuda_aplicac	sector	facturacion	empleados	ccaa	estatus	partner
2	1	integral	PYME	Caixa Bank	10000	None	-	None	Actividades p	Menos de 2M	None	Cataluña	Finalizado	KVAR
461	4	integral	PYME	Caixa Bank	0	Quiere atenci	-	None	Comercio al p	De 10M€ a 50	Mas de 50	Andalucía	Finalizado	KVAR
963	5	integral	PYME	Caixa Bank	0	Quiere digital	-	None	Otros servicio	None	None	None	Finalizado	KVAR
1571	6	integral	PYME	BBVA	0	Empresa dedi	-	None	Construcción	None	None	Canarias	Rechazado pc	KVAR
2963	7	integral	PYME	Caixa Bank	0	Empresa dedi	-	None	Información y	Menos de 2M	None	Madrid	Finalizado	KVAR
1491	8	integral	PYME	BBVA	0	Empresa cons	-	None	Comercio al p	Menos de 2M	None	None	Finalizado	KVAR
3517	10	integral	PYME	Caixa Bank	0	None	-	None	Construcción	Menos de 2M	None	None	Finalizado	KVAR
3281	11	integral	PYME	Caixa Bank	0	None	-	None	Comercio al p	Menos de 2M	None	None	Finalizado	KVAR
3493	12	integral	PYME	Caixa Bank	0	None	-	None	Actividades fi	Menos de 2M	None	None	Finalizado	KVAR
3967	13	integral	PYME	BBVA	0	Juan Antonio	-	None	Construcción	Menos de 2M	None	None	Pendiente cor	KVAR

Ilustración 26: Ejemplo botón info lead para interfaz partner

La lógica es muy fácil de seguir y es muy intuitivo, si le dan a aceptar la base de datos les da acceso a la información personal de contacto del lead. Esto permite que sea mas rápido y que no tengan que esperar que nosotros les pasemos la información, directamente ya la tienen. El botón de rechazar lo que hace es devolver al lead al estado de pendiente de derivar y se queda sin partner asignado y por lo tanto ya no esta en la base de datos del partner. El botón de contratado lo que hace es avisar a todos de que ya se han puesto en contacto con el cliente y ha aceptado a seguir adelante con el partner. En el caso que el

partner o el cliente se echen para atrás pueden darle a botón de rechazado siempre y cuando se inserte una fecha de finalizado.

Las dos interfaces están conectadas porque realmente usan la misma base de datos solo que la de los partners tiene un filtro personal para ellos. La interfaz deja que se puedan actualizar con las dos herramientas a la vez.

Capítulo 6. ANÁLISIS DE RESULTADOS

Tras haber terminado el proyecto y haber visto la aplicación de la herramienta y la rapidez de cómo se mueven los datos y la facilidad de poder acceder a los datos podemos mostrar los siguientes resultados muy favorables:

- Eficiencia de la transformación: La transformación es muy rápida y literal se puede hacer todo en 2minutos calculados, se ha comprobado que con los códigos se puede recrear la base de datos muy rápido.
- Rendimiento de la base de datos SQL: El rendimiento de la base de datos ha sido muy satisfactorio, no hay ningún tipo de lag entre las interfaces y la información se actualiza de forma instantánea. No hace falta cerrar la herramienta para ver las actualizaciones que hace el contrario.
- Usabilidad de la interfaz: La interfaz es muy intuitiva y simple para que no haya perdida. La gran ventaja es que podemos actualizar una base de datos y sacar información sobre ella sin saber SQL, lo cual la hace muy versátil.
- Precisión de los resultados: A lo largo de las pruebas se ha evaluado la precisión de que no haya errores de carga o de actualización, se puede decir que es muy precisa ya que está programado para que no se permita hacer errores.
- Impacto en la toma de decisiones: Por muy simple que parece la herramienta tiene un impacto gigante, ya que soluciona mucho errores que cometemos como errores, un punto muy importante es el servicio rápido que se le puede dar a los clientes y a los partners.
- Comparativa de recursos y costos: Debido a que esto ha sido desarrollado por una persona, el único coste que ha tenido ha sido el de la formación de la persona que lo crea. Debido a los cursos realizado el coste de la herramienta es 40€ en cursos y luego mucho tiempo programando y resolviendo problemas de lógica y código. En todo caso puede ser una reducción de coste en una empresa porque te ahorras una suscripción a una herramienta de CRM.

- Feedback del usuario: La herramienta no está muy desarrollada, se puede hacer poco, pero es lo esencial, ya que en la hoja de cálculo no se hacía nada más que esto. Hay mucho espacio para mejoras, para poder integrar más acciones y hacerla la interfaz más bonita al ojo.

En general, se puede afirmar que el proyecto a sido un éxito y se ha conseguido lograr que que se buscaba. El único inconveniente es el tiempo que hay que dedicarle y todas las mejoras que todavía se le pueden hacer a la herramienta, como alertas al correo para que los partners solo entren en la herramienta cuando hay cosas nuevas y una parte de usuario donde cada cliente puede ver cuál es el estado de su proyecto.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIÓN

En conclusión, este Trabajo de Fin de Máster ha abordado de manera integral y detallada la transformación de una base de datos en hoja de cálculo a un formato SQL, junto con el desarrollo de una interfaz interactiva que permite una interacción efectiva con dicha base de datos. A través de una metodología sólida y una cuidadosa planificación, se ha logrado llevar a cabo este proceso de migración con éxito, obteniendo resultados significativos y de gran utilidad en el ámbito de la gestión y análisis de datos.

La transformación de la base de datos en hoja de cálculo a SQL ha demostrado ser una decisión acertada, ya que ha permitido aprovechar las ventajas de una estructura organizada y eficiente para el almacenamiento y gestión de los datos. Con la capacidad de utilizar consultas poderosas y realizar análisis más complejos, se ha mejorado la eficiencia y precisión en el tratamiento de la información.

La creación de una interfaz interactiva ha sido un logro destacado en este proyecto, ya que ha facilitado la interacción de los usuarios con la base de datos de manera intuitiva y amigable. Gracias a esta interfaz, los usuarios pueden realizar consultas personalizadas, visualizar resultados y actualizar datos en tiempo real, lo que ha mejorado significativamente la experiencia de usuario y ha agilizado la toma de decisiones basadas en la información disponible.

Es importante resaltar que la implementación de este proyecto ha requerido el uso de diversas tecnologías y librerías, como Python, SQL, ttkinter, y otras, que han demostrado ser herramientas poderosas y versátiles para el desarrollo de esta solución. La combinación de estas tecnologías ha sido clave para alcanzar los objetivos propuestos y lograr un resultado robusto y de calidad.

En conclusión, este Trabajo de Fin de Máster ha sido un ejercicio valioso en la aplicación de conocimientos teóricos y prácticos en el campo de la gestión de datos y el desarrollo de interfaces interactivas. Los resultados obtenidos han demostrado el impacto positivo que una adecuada transformación de la base de datos y la implementación de una interfaz eficiente pueden tener en la eficacia y eficiencia de la toma de decisiones en diversas áreas profesionales. Este proyecto sienta las bases para futuros desarrollos y mejoras en el ámbito de la gestión de datos y la interacción con la información, en busca de una mayor optimización y mejora continua en la toma de decisiones informadas.

7.2 TRABAJOS FUTUROS

Tras completar exitosamente el Trabajo de Fin de Máster sobre la transformación de una base de datos en hoja de cálculo a un formato SQL y la creación de una interfaz interactiva, existen varias oportunidades para futuros trabajos y mejoras en este ámbito. Algunas líneas de investigación y desarrollo que podrían considerarse son las siguientes:

- Mejora de la interfaz de usuario: Realizar estudios de usabilidad y obtener comentarios de los usuarios para identificar posibles áreas de mejora en la interfaz. Con base en los resultados, se podrían implementar cambios para optimizar la experiencia del usuario, haciéndola más intuitiva, atractiva y fácil de usar.
- Funcionalidades adicionales: Considerar la incorporación de nuevas funcionalidades a la interfaz, como filtros avanzados, gráficos interactivos, herramientas de exportación de datos o funciones de análisis más avanzadas. Estas adiciones podrían aumentar la utilidad y el valor de la interfaz para los usuarios
- Seguridad y gestión de permisos: Implementar medidas de seguridad para garantizar que solo los usuarios autorizados puedan acceder y modificar ciertos datos en la base de datos. La gestión de permisos y roles es esencial para proteger la integridad y confidencialidad de la información almacenada.

- Sincronización de datos: Investigar y desarrollar soluciones para la sincronización bidireccional entre la base de datos SQL y la hoja de cálculo. Esto permitiría mantener actualizados los datos en ambas plataformas, garantizando coherencia y precisión en toda la información.
- Integración con otras herramientas: Explorar la posibilidad de integrar la interfaz con otras herramientas y plataformas, como sistemas de análisis de datos, generación de informes o soluciones de inteligencia empresarial. La integración podría ampliar las capacidades de la interfaz y permitir un flujo de trabajo más completo.
- Migración de datos a la nube: Investigar la posibilidad de migrar la base de datos a una plataforma en la nube, como Amazon Web Services (AWS) o Microsoft Azure. Esto podría mejorar la escalabilidad, el rendimiento y la disponibilidad de los datos, así como facilitar el acceso desde múltiples ubicaciones.
- Estudio comparativo de tecnologías: Realizar un análisis comparativo entre diferentes sistemas de gestión de bases de datos SQL para determinar cuál es el más adecuado para la aplicación en cuestión. Se podrían evaluar aspectos como el rendimiento, la facilidad de administración y la compatibilidad con las necesidades específicas del proyecto.

En resumen, el Trabajo de Fin de Máster sobre la transformación de una base de datos en hoja de cálculo a un formato SQL y la creación de una interfaz interactiva es solo el punto de partida para futuras exploraciones y avances en este campo. Los trabajos futuros mencionados anteriormente podrían contribuir a mejorar la funcionalidad, usabilidad y eficiencia del sistema, ampliando así su potencial aplicación en diversas áreas profesionales y académicas.

Capítulo 8. BIBLIOGRAFÍA

[1] «Software de hojas de cálculo Microsoft Excel | Microsoft 365». https://www.microsoft.com/es-es/microsoft-365/excel (accedido 24 de julio de 2023).
[2] «User Guide — pandas 2.0.3 documentation». https://pandas.pydata.org/docs/user_guide/index.html (accedido 25 de julio de 2023).
[3] «Online Courses - Learn Anything, On Your Schedule», *Udemy*. https://www.udemy.com/ (accedido 26 de julio de 2023).
[4] «Python: HTML, CSS, Flask y MySQL - Hola Mundo». https://academia.holamundo.io/courses/take/python-html-css-flask-y-mysql/lessons/35133070-6-insert-y-actualizamos-tabla-usuario (accedido 26 de julio de 2023).
[5] «Conceptos básicos sobre bases de datos - Soporte técnico de Microsoft». https://support.microsoft.com/es-es/office/conceptos-b%C3%A1sicos-sobre-bases-de-datos-a849ac16-07c7-4a31-9948-3c8c94a7c204 (accedido 25 de marzo de 2023).
[6] «Historia de las Bases de Datos – Historia de la Informática», 4 de enero de 2011. https://histinf.blogs.upv.es/2011/01/04/historia-de-las-bases-de-datos/ (accedido 25 de julio de 2023).
[7] «Topics | IBM». https://www.ibm.com/topics (accedido 26 de julio de 2023).
[8] «What is a database?» https://www.oracle.com/database/what-is-database/ (accedido 26 de julio de 2023).
[9] «Microsoft Data Platform | Microsoft». https://www.microsoft.com/en-us/sql-server (accedido 26 de julio de 2023).
[10] «¿Qué es una base de datos?» https://www.oracle.com/es/database/what-is-database/ (accedido 25 de marzo de 2023).
[11] «¿Qué es una base de datos relacional?» https://www.oracle.com/es/database/what-is-a-relational-database/ (accedido 25 de marzo de 2023).
[12] «Diagrama_Empleado.jpeg (790×151)». https://upload.wikimedia.org/wikipedia/commons/e/ed/Diagrama_Empleado.jpeg (accedido 19 de abril de 2023).
[13] «Base de datos orientadas a objetos ¿Qué son? | AyudaLey», *Ayuda Ley Protección Datos*. https://ayudaleyprotecciondatos.es/bases-de-datos/orientas-a-objetos/ (accedido 25 de marzo de 2023).
[14] «Base de datos distribuida. ¿Qué es? Características», *Ayuda Ley Protección Datos*. https://ayudaleyprotecciondatos.es/bases-de-datos/distribuida/ (accedido 19 de abril de 2023).
[15] «Bases de Datos Distribuidas: Popularidad, Uso y Tipos». https://www.tecnologias-informacion.com/distribuidas.html (accedido 19 de abril de 2023).
[16] «¿Qué es una base de datos en memoria?», *TIBCO Software*. https://www.tibco.com/es/reference-center/what-is-an-in-memory-database (accedido 19 de abril de 2023).

[17] «https://www.tibco.com/sites/tibco/files/media_entity/2022-02/real-time-data-01.svg». Accedido: 19 de abril de 2023. [En línea]. Disponible en: https://www.tibco.com/sites/tibco/files/media_entity/2022-02/real-time-data-01.svg

[18] «¿Qué son los datos en tiempo real?», *TIBCO Software*. https://www.tibco.com/es/reference-center/what-is-real-time-data (accedido 19 de abril de 2023).

[19] https://www.tibco.com/sites/tibco/files/media_entity/2022-02/real-time-data-01.svg (Accedido 19 de abril de 2023).

Capítulo 9. ANEXO I : ALINEACIÓN ODS

El Trabajo Final de Máster (TFM) sobre el traspaso de una base de datos en formato Excel a formato SQL y la creación de una interfaz para interactuar con dicha base de datos se alinea principalmente con los siguientes Objetivos de Desarrollo Sostenible (ODS):

1. ODS 9 - Industria, innovación e infraestructura: Al migrar la base de datos a formato SQL y crear una interfaz para interactuar con ella, se busca mejorar la infraestructura tecnológica y promover la innovación en el manejo de datos. Esto puede llevar a una mayor eficiencia y optimización de procesos, lo que es esencial para el desarrollo de una industria sostenible.

2. ODS 11 - Ciudades y comunidades sostenibles: La creación de una interfaz para interactuar con la base de datos puede mejorar el acceso a la información y la toma de decisiones más informadas, lo que contribuye a una gestión más eficiente y sostenible de los recursos en las comunidades.

3. ODS 12 - Producción y consumo responsables: La migración de una base de datos a un formato más eficiente, como SQL, puede ayudar a reducir el consumo innecesario de recursos y energía, promoviendo así prácticas de producción y consumo más responsables y sostenibles.

5. ODS 17 - Alianzas para lograr los objetivos: La colaboración y el trabajo conjunto entre diferentes actores involucrados en el proyecto, como académicos, industrias y comunidades, es esencial para alcanzar los objetivos de desarrollo sostenible, en este caso, a través de la mejora en la gestión de datos.

Estos ODS reflejan la importancia y el impacto potencial del TFM en la promoción del desarrollo sostenible a través de la optimización y accesibilidad de la información, así como el fomento de prácticas más responsables en la gestión de datos.

Capítulo 10. ANEXO II: CÓDIGOS

10.1 CODIGOS PARA CREAR LAS TABLAS SIMPLE

El código para poder crear todas las tablas simples se adjunta a continuación:

```
create table entidad(
  id_entidad int auto_increment primary key,
  nombre_entidad varchar(20),
  descuento int
);

create table partner(
  id_partner int auto_increment primary key,
  partner varchar(150),
  nombre_p varchar(150),
  correo_p varchar(150)
);

create table estatus(
  id_estatus int auto_increment primary key,
  estatus varchar(150)
);

create table resultado(
  id_resultado int auto_increment primary key,
  des_resultado varchar(150)
);

create table tipo_cliente (
  id_tipo int auto_increment primary key,
  tipo_cliente varchar(150)
);

create table tipo_oportunidad (
  id_oportunidad int auto_increment primary key,
  tipo_oportunidad varchar(150)
);

create table tipo_proyecto (
  id_tipo_p int auto_increment primary key,
  tipo_proyecto varchar(150)
);

create table sector(
  id_sector int auto_increment primary key,
```

```
  sector varchar(150)
);

create table empleados(
  id_empleados int auto_increment primary key,
  empleados varchar(150)
);

create table cnae(
  id_cnae int auto_increment primary key,
  codigo int,
  cnae varchar(150)
);

create table tam(
  id_tam int auto_increment primary key,
  tam varchar(150)
);

create table provincia(
  id_provincia int auto_increment primary key,
  provincia varchar(150)
);

create table facturacion(
  id_facturacion int auto_increment primary key,
  facturacion varchar(150)
);

create table ccaa(
  id_ccaa int auto_increment primary key,
  ccaa varchar(150)
);
```

10.2 CÓDIGO DE INSERT INTO TABLAS SIMPLES

```
import mysql.connector

midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()

for i in range(rows):
        sql = "INSERT INTO tabla (col1,col2,col3) values (%s,%s,%s);"
```

```
        values = (df.iat[i,0],df.iat[i,1],df.iat[i,2])
        cursos.execute(sql,values)
midb.commit()
```

10.2.1 INSERT INTO DE LA TABLAS SIMPLES

```
INSERT INTO entidad (nombre_entidad,descuento) values (%s,%s);
INSERT INTO partner (partner,nombre_p,correo_p) values (%s,%s,%s);
INSERT INTO estatus (estatus) values (%s);
INSERT INTO resultado (resultado) values (%s);
INSERT INTO tipo_cliente (tipo_cliente) values (%s);
INSERT INTO tipo_oportunidad (tipo_oportunidad) values (%s);
INSERT INTO sector (sector) values (%s);
INSERT INTO empleados (empleados) values (%s);
INSERT INTO cnae (codigo,cnae) values (%s,%s);
INSERT INTO provincia (provincia) values (%s);
INSERT INTO facturacion (facturacion) values (%s);
INSERT INTO ccaa (ccaa) values (%s);
```

10.3 CÓDIGO TABLAS COMPLEJAS

10.3.1 CLIENTES

```
create table cliente(
    id_cliente int primary key,
    nombre_empresa varchar(50),
    cif_nif varchar(10),
    entidad int, -- foreign key
    nombre_cliente varchar(50),
    apellido varchar(50),
    correo_cliente varchar(100),
    fecha_lead date,
    fecha_oportunidad date,
    foreign key (entidad) references entidad(id_entidad)
);
```

10.3.2 PROYECTO

```
create table proyecto(
        id_proyecto int auto_increment primary key,
        id_cliente int, -- foreign key
        tipo_cliente int, -- foreign key (pyme, gran empresa, ...
        partner int, -- foreign key
        estatus int, -- foreign key
        fecha_derivacion date,
        fecha_fin date,
```

```
        tipo_oportunidad int, -- foreign key (masivo, proyecto e integral
        inversión int,
        descripción_empresa varchar(1000),
        interes varchar(1000),
        ayuda_aplicada varchar(1000),
        sector int, -- foreign key
        cnae varchar(250), -- foreign key
        facturacion int, -- foreign key
        numero_empleados int, -- foreign key
        provincia int, -- foreign key
        ccaa int, -- foreign key
        foreign key (id_cliente) references cliente(id_cliente),
        foreign key (tipo_cliente) references tipo_cliente(id_tipo),
        foreign key (tipo_oportunidad) references
tipo_oportunidad(id_oportunidad),
        foreign key (sector) references sector(id_sector),
        foreign key (facturacion) references facturacion(id_facturacion),
        foreign key (numero_empleados) references
empleados(id_empleados),
        foreign key (provincia) references provincia(id_provincia),
        foreign key (ccaa) references ccaa(id_ccaa),
        foreign key (partner) references partner(id_partner),
        foreign key (estatus) references estatus(id_estatus)
);
```

10.3.3 DERIVACIÓN

```
create table derivacion(
        id_derivacion int auto_increment primary key,
        id_proyecto int, -- foreing key
        id_cliente int, -- foreign key
        partner int, -- foreign key
        fecha_derivación date,
        fecha_fin date,
        estatus int, -- foreign key
        foreign key(id_proyecto) references proyecto(id_proyecto),
        foreign key (id_cliente) references cliente(id_cliente),
        foreign key (partner) references partner(id_partner),
        foreign key (estatus) references estatus(id_estatus)
)
```

10.3.4 CÓDIGO DE INSERT INTO TABLAS COMPLEJAS

10.3.5 CLIENTES

```
import pandas as pd
import mysql.connector

def buscar_banco(entidad):
```

```
    if entidad == 'BBVA':
        return 1
    if entidad == 'CaixaBank':
        return 2
    if entidad == 'Abanca':
        return 3
    if entidad == 'Banca Pueyo':
        return 4

def fechas_bien(lead,oport):
    lead = lead.split(' ')
    oport = oport.split(' ')
    return(lead[0],oport[0])


midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()

nombre_hoja = 'Hoja1'
path = 'bbdd.xlsx'
df = pd.read_excel(path,sheet_name=nombre_hoja)

rows = len(df.axes[0])
col = len(df.axes[1])

#crear tabla solo del cliente
for i  in range(rows):
    print(df.iat[i,4])
    entidad = buscar_banco(df.iat[i,12])
    fechas = fechas_bien(str(df.iat[i,3]),str(df.iat[i,3]))
    print(fechas[0])
    try:
        sql = f"INSERT INTO cliente
(id_cliente,nombre_empresa,cif_nif,entidad,nombre_cliente,apellido,correo
_cliente,fecha_lead,fecha_oportunidad) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s);"
        values =
(int(df.iat[i,2]),df.iat[i,5],df.iat[i,6],entidad,df.iat[i,7],df.iat[i,8]
,df.iat[i,10],fechas[0],fechas[1])
        cursos.execute(sql,values)
    except:
        print(df.iat[i,2])


midb.commit()
```

10.3.6 PROYECTO

```
import pandas as pd
import mysql.connector
import aux_funciones

midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()

nombre_hoja = 'Hoja1'
path = 'bbdd.xlsx'
df = pd.read_excel(path,sheet_name=nombre_hoja)

rows = len(df.axes[0])
col = len(df.axes[1])

#crear tabla solo del cliente
for i  in range(1,rows):

    id_cliente = int(df.iat[i,2])
    tipo_cliente = aux_funciones.tip_cliente_bien(df.iat[i,9])
    partner = aux_funciones.buscar_partner(df.iat[i,28])
    estatus = aux_funciones.buscar_estatus(df.iat[i,32])
    fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))
    tipo_oportunidad = aux_funciones.tip_oportunidad_bien(df.iat[i,1])
    facturacion = aux_funciones.facturacion_bien(df.iat[i,21])
    numero_empleados = aux_funciones.numero_empleados_bien(df.iat[i,22])
    provincia =aux_funciones.provincia(df.iat[i,24])
    ccaa = aux_funciones.ccaa_bien(df.iat[i,26])
    sector = aux_funciones.sector_bien(df.iat[i,19])
    try:
        inversión = int(df.iat[i,15])
    except:
        inversión = 0
    descripción_empresa = df.iat[i,46]
    interes = df.iat[i,17]
    ayuda_aplicada = df.iat[i,18]
    cnae = df.iat[i,20]

    try:
        fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))
```

```
        sql = f"INSERT INTO proyecto
(id_cliente,tipo_cliente,partner,estatus,fecha_derivacion,fecha_fin,tipo_
oportunidad,inversión,descripción_empresa,interes,ayuda_aplicada,sector,c
nae,facturacion,numero_empleados,provincia,ccaa) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"
        values =
(id_cliente,tipo_cliente,partner,estatus,fechas[0],fechas[1],tipo_oportun
idad,inversión,descripción_empresa,interes,ayuda_aplicada,sector,cnae,fac
turacion,numero_empleados,provincia,ccaa)
        cursos.execute(sql,values)
    except:
        sql = f"INSERT INTO proyecto
(id_cliente,tipo_cliente,partner,estatus,tipo_oportunidad,inversión,descr
ipción_empresa,interes,ayuda_aplicada,sector,cnae,facturacion,numero_empl
eados,provincia,ccaa) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"
        values =
(id_cliente,tipo_cliente,partner,estatus,tipo_oportunidad,inversión,descr
ipción_empresa,interes,ayuda_aplicada,sector,cnae,facturacion,numero_empl
eados,provincia,ccaa)
        cursos.execute(sql,values)

midb.commit()
```

10.3.7 DERIVACIÓN

```
import pandas as pd
import mysql.connector
import aux_funciones

midb = mysql.connector.connect(
    host='localhost',
    user='pincho',
    password='Natacion01-',
    database='tfm'
)
cursos = midb.cursor()

nombre_hoja = 'Hoja1'
path = 'bbdd.xlsx'
df = pd.read_excel(path,sheet_name=nombre_hoja)

rows = len(df.axes[0])
col = len(df.axes[1])

#crear tabla solo del cliente
for i  in range(1,rows):

    id_cliente = int(df.iat[i,2])
```

```
    partner = aux_funciones.buscar_partner(df.iat[i,28])
    if partner != 'NULL':


        print(partner)
        estatus = aux_funciones.buscar_estatus(df.iat[i,32])
        fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))

        sql = f"SELECT id_proyecto FROM proyecto WHERE id_cliente =
'{df.iat[i,2]}'"
        print(sql)
        cursos.execute(sql)
        id_pro = cursos.fetchall()
        x = list(id_pro[0])
        id_proyecto = x[0]

        try:
            fechas =
aux_funciones.fechas_bien(str(df.iat[i,33]),str(df.iat[i,34]))

            sql = f"INSERT INTO derivacion
(id_proyecto,id_cliente,partner,estatus,fecha_derivacion,fecha_fin)
values (%s,%s,%s,%s,%s,%s);"
            values =
(id_proyecto,id_cliente,partner,estatus,fechas[0],fechas[1])
            cursos.execute(sql,values)
        except:
            sql = f"INSERT INTO derivacion
(id_proyecto,id_cliente,partner,estatus) values (%s,%s,%s,%s);"
            values = (id_proyecto,id_cliente,partner,estatus)
            cursos.execute(sql,values)

midb.commit()
```

10.4 CÓDIGO PARA LÓGICA GUI

```
import tkinter as tk
from tkinter import ttk
from model.cliente_consultas import
create_table,borrar_tabla,Cliente,guardar,listar,columnas_tabla,derivacio
n
from model.conexion_db import Conexion_DB

def barra_menu(root):
    barra_menu = tk.Menu(root)
    root.config(menu = barra_menu, width=300,height = 300)

    menu_inicio = tk.Menu(barra_menu,tearoff=0)
```

```
    barra_menu.add_cascade(label='Inicio',menu=menu_inicio)
    menu_inicio.add_command(label='Crear registro en
BD',command=create_table)
    menu_inicio.add_command(label='Eliminar registro en
BD',command=borrar_tabla)
    menu_inicio.add_command(label='Actualizar registro en BD')
    menu_inicio.add_command(label='Salir',command=root.destroy)

    menu_consulta = tk.Menu(barra_menu,tearoff=0)
    barra_menu.add_cascade(label='Consulta',menu = menu_consulta)

    menu_config = tk.Menu(barra_menu,tearoff=0)
    barra_menu.add_cascade(label='Configuración',menu=menu_config)

    menu_ayuda = tk.Menu(barra_menu,tearoff=0)
    barra_menu.add_cascade(label='Ayudas',menu=menu_ayuda)

class Frame(tk.Frame):

    def __init__(self,root= None):
        super().__init__(root,width=1000,height=200)
        self.root = root
        self.pack()
        #self.config(bg= 'green')
        self.botones()
        self.filtros()
        self.select_tabla_clientes()
        self.tabla_select = 1
        self.query_busqueda = 0



    def botones(self):

        #crear botones
        self.boton_modo_buscar = tk.Button(self,text='Modo
Buscar',command=self.modo_busqueda)
        self.boton_modo_buscar.config(width=10,font=
('Arial',11,'bold'),fg = 'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat',state='disabled')
        self.boton_modo_buscar.grid(row=0,column=4,padx=5,pady=5)

        self.boton_exit_buscar = tk.Button(self,text='Exit
Buscar',command=self.exit_buscar)
        self.boton_exit_buscar.config(width=10,font=
('Arial',11,'bold'),fg = 'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat',state='disabled')
        self.boton_exit_buscar.grid(row=2,column=4,padx=5,pady=5)

        self.boton_buscar =
tk.Button(self,text='Buscar',command=self.busqueda)
```

```
        self.boton_buscar.config(width=10,font= ('Arial',11,'bold'),fg =
'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat',state='disabled')
        self.boton_buscar.grid(row=1,column=4,padx=5,pady=5)

        self.boton_actualizar = tk.Button(self,text='Info
Lead',command=self.habilitar_actualizar)
        self.boton_actualizar.config(width=10,font=
('Arial',11,'bold'),fg = 'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat',state='disabled')
        self.boton_actualizar.grid(row=0,column=3,padx=5,pady=5)


        self.boton_guardar =
tk.Button(self,text='Derivar',command=self.derivar)
        self.boton_guardar.config(width=10,font= ('Arial',11,'bold'),fg =
'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat',state='disabled')
        self.boton_guardar.grid(row=1,column=3,padx=5,pady=5)

        self.boton_cancelar =
tk.Button(self,text='Cancelar',command=self.deshabilitar_actualizar)
        self.boton_cancelar.config(width=10,font= ('Arial',11,'bold'),fg
= 'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat',state='disabled')
        self.boton_cancelar.grid(row=2,column=3,padx=5,pady=5)


        #Botones activos
        self.boton_proyecto = tk.Button(self,text='Tabla
Proyectos',command=self.select_tabla_proyecto)
        self.boton_proyecto.config(width=15,font= ('Arial',11,'bold'),fg
= 'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat')
        self.boton_proyecto.grid(row=0,column=5,padx=5,pady=5)

        self.boton_cliente = tk.Button(self,text='Tabla
clientes',command=self.select_tabla_clientes)
        self.boton_cliente.config(width=15,font= ('Arial',11,'bold'),fg =
'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat')
        self.boton_cliente.grid(row=1,column=5,padx=5,pady=5)

        self.boton_cliente = tk.Button(self,text='Tabla
Derivaciones',command=self.select_tabla_derivacion)
        self.boton_cliente.config(width=15,font= ('Arial',11,'bold'),fg =
'white',bg = '#95FEEB',cursor='hand2',
activebackground='#BDBDBD',relief='flat')
        self.boton_cliente.grid(row=2,column=5,padx=5,pady=5)
```

```
    def filtros(self):

        #creación de labels para metar campos
        self.label_id_proyecto = tk.Label(self,text='id_proyecto:
',width=20)
        self.label_id_proyecto.config(font= ('Arial',10))
        self.label_id_proyecto.grid(row=0,column=0,padx=5,pady=5)

        self.label_partner = tk.Label(self,text='Partner: ',width=20)
        self.label_partner.config(font= ('Arial',10))
        self.label_partner.grid(row=1,column=0,padx=5,pady=5)

        self.label_fecha = tk.Label(self,text='Fecha: ',width=20)
        self.label_fecha.config(font= ('Arial',10))
        self.label_fecha.grid(row=2,column=0,padx=5,pady=5)

        #crear entradas
        self.mi_id_proyecto = tk.StringVar()
        self.entry_id_proyecto =
tk.Entry(self,textvariable=self.mi_id_proyecto)
        self.entry_id_proyecto.config(width=20,font= ('Arial',11))
        self.entry_id_proyecto.grid(row=0,column=1,padx=5,pady=5)

        self.mi_partner = tk.StringVar()
        self.entry_partner = tk.Entry(self,textvariable=self.mi_partner)
        self.entry_partner.config(width=20,font= ('Arial',11))
        self.entry_partner.grid(row=1,column=1,padx=5,pady=5)

        self.mi_fecha =tk.StringVar()
        self.entry_fecha = tk.Entry(self,textvariable=self.mi_fecha)
        self.entry_fecha.config(width=20,font= ('Arial',11))
        self.entry_fecha.grid(row=2,column=1,padx=5,pady=5)

        #filtros
        self.label_filtro = tk.Label(self,text='')
        self.label_filtro.config(font= ('Arial',10))

self.label_filtro.grid(row=5,column=2,columnspan=2,rowspan=2,padx=5,pady=
5)

        #Label
        self.label_tipo_cliente = tk.Label(self,text='Tipo
Cliente',width=20)
        self.label_tipo_cliente.config(font= ('Arial',10))
        self.label_tipo_cliente.grid(row=7,column=0,padx=5,pady=5)

        self.label_tipo_oportu = tk.Label(self,text='Tipo
Oportunidad',width=20)
        self.label_tipo_oportu.config(font= ('Arial',10))
        self.label_tipo_oportu.grid(row=7,column=1,padx=5,pady=5)
```

```
        self.label_entidad = tk.Label(self,text='Entidad',width=20)
        self.label_entidad.config(font= ('Arial',10))
        self.label_entidad.grid(row=7,column=2,padx=5,pady=5)

        self.label_inversion = tk.Label(self,text='Inversion',width=20)
        self.label_inversion.config(font= ('Arial',10))
        self.label_inversion.grid(row=7,column=3,padx=5,pady=5)

        self.label_sector = tk.Label(self,text='Sector',width=20)
        self.label_sector.config(font= ('Arial',10))
        self.label_sector.grid(row=7,column=4,padx=5,pady=5)

        self.label_estatus = tk.Label(self,text='Estatus',width=20)
        self.label_estatus.config(font= ('Arial',10))
        self.label_estatus.grid(row=7,column=5,padx=5,pady=5)

        self.label_facturacion =
tk.Label(self,text='Facturacion',width=20)
        self.label_facturacion.config(font= ('Arial',10))
        self.label_facturacion.grid(row=5,column=0,padx=5,pady=5)

        self.label_id_cliente = tk.Label(self,text='id cliente',width=20)
        self.label_id_cliente.config(font= ('Arial',10))
        self.label_id_cliente.grid(row=5,column=5,padx=5,pady=5)

        #Desplegables
        self.mi_tipo_cliente = tk.StringVar()
        self.desple_tipo_cliente =
tk.Entry(self,width=20,textvariable=self.mi_tipo_cliente)
        self.desple_tipo_cliente.grid(row=8,column=0,padx=5,pady=5)

        self.mi_tipo_opor = tk.StringVar()
        self.entry_tipo_opor=
tk.Entry(self,width=20,textvariable=self.mi_tipo_opor)
        self.entry_tipo_opor.grid(row=8,column=1,padx=5,pady=5)

        self.mi_tipo_entidad = tk.StringVar()
        self.desple_tipo_entidad=
tk.Entry(self,width=20,textvariable=self.mi_tipo_entidad)
        self.desple_tipo_entidad.grid(row=8,column=2,padx=5,pady=5)

        self.mi_inversion = tk.StringVar()
        self.desple_inversion =
tk.Entry(self,width=20,textvariable=self.mi_inversion)
        self.desple_inversion.grid(row=8,column=3,padx=5,pady=5)

        self.mi_sector = tk.StringVar()
        self.desple_sector =
tk.Entry(self,width=20,textvariable=self.mi_sector)
        self.desple_sector.grid(row=8,column=4,padx=5,pady=5)
```

```
        self.mi_estatus = tk.StringVar()
        self.desple_estatus =
tk.Entry(self,width=20,textvariable=self.mi_estatus)
        self.desple_estatus.grid(row=8,column=5,padx=5,pady=5)

        self.mi_facturacion = tk.StringVar()
        self.desple_facturacion =
tk.Entry(self,width=20,textvariable=self.mi_facturacion)
        self.desple_facturacion.grid(row=6,column=0,padx=5,pady=5)

        self.mi_id_cliente = tk.StringVar()
        self.entry_id_cliente=
tk.Entry(self,width=20,textvariable=self.mi_id_cliente)
        self.entry_id_cliente.grid(row=6,column=5,padx=5,pady=5)

    def habilitar_actualizar(self):
        self.entry_partner.config(state='normal')
        self.entry_fecha.config(state='normal')
        self.boton_guardar.config(state='normal')
        self.boton_cancelar.config(state='normal')
        self.label_filtro.config(text='-- Info Lead --')
        self.boton_modo_buscar.config(state='normal')
        self.boton_exit_buscar.config(state='disabled')
        self.boton_buscar.config(state='disabled')
        self.info_lead()
        self.deshabilitar_campo()

    def deshabilitar_actualizar(self):
        self.mi_partner.set('')
        self.mi_fecha.set('')
        self.mi_id_proyecto.set('')

        self.entry_id_proyecto.config(state='disabled')
        self.entry_partner.config(state='disabled')
        self.entry_fecha.config(state='disabled')
        self.boton_guardar.config(state='disabled')
        self.boton_cancelar.config(state='disabled')

        self.vacio_busqueda()

    def habilitar_campos(self):

        self.label_filtro.config(text='-- Filtros --')
        self.desple_estatus.config(state='normal')
        self.desple_tipo_cliente.config(state='normal')
        self.desple_tipo_entidad.config(state='normal')
        self.desple_inversion.config(state='normal')
        self.desple_sector.config(state='normal')
        self.desple_facturacion.config(state='normal')
        self.entry_tipo_opor.config(state='normal')
        self.entry_id_cliente.config(state='normal')
```

```
        self.boton_exit_buscar.config(state='normal')
        self.boton_buscar.config(state='normal')

    def deshabilitar_campo(self):

        self.desple_estatus.config(state='disabled')
        self.desple_tipo_cliente.config(state='disabled')
        self.desple_tipo_entidad.config(state='disabled')
        self.desple_inversion.config(state='disabled')
        self.desple_sector.config(state='disabled')
        self.desple_facturacion.config(state='disabled')
        self.entry_tipo_opor.config(state='disabled')
        self.entry_id_cliente.config(state='disabled')
        self.boton_exit_buscar.config(state='disabled')
        self.boton_buscar.config(state='disabled')

    def lable_buscar_clientes(self):

        self.label_tipo_cliente.config(text='')
        self.label_tipo_oportu.config(text='Nombre Cliente')
        self.label_entidad.config(text='Nombre Empresa')
        self.label_inversion.config(text='NIF')
        self.label_sector.config(text='Correo')
        self.label_estatus.config(text='')
        self.label_facturacion.config(text='')
        self.label_id_cliente.config(text='')


    def lable_buscar_derivación(self):

        self.label_tipo_cliente.config(text='')
        self.label_tipo_oportu.config(text='id cliente  ')
        self.label_entidad.config(text='Nombre Cliente')
        self.label_inversion.config(text='Partner')
        self.label_sector.config(text='Estado')
        self.label_estatus.config(text='')
        self.label_facturacion.config(text='')
        self.label_id_cliente.config(text='')


    def habilitar_busqueda_clientes(self):

        self.label_filtro.config(text='-- Filtros --')
        self.desple_estatus.config(state='disable')
        self.desple_tipo_cliente.config(state='disable')
        self.desple_tipo_entidad.config(state='normal')
        self.desple_inversion.config(state='normal')
        self.desple_sector.config(state='normal')
        self.desple_facturacion.config(state='disable')
        self.entry_tipo_opor.config(state='normal')
        self.entry_id_cliente.config(state='disable')
```

```
        self.boton_exit_buscar.config(state='normal')
        self.boton_buscar.config(state='normal')
        self.boton_actualizar.config(state='disabled')


    def label_buscar_proyecto(self):
        self.label_tipo_cliente.config(text='Tipo Cliente')
        self.label_tipo_oportu.config(text='Tipo Oportunidad')
        self.label_entidad.config(text='Entidad')
        self.label_inversion.config(text='Inversión')
        self.label_sector.config(text='Sector')
        self.label_estatus.config(text='Estatus')
        self.label_facturacion.config(text='Facturación')
        self.label_id_cliente.config(text='id cliente')

    def select_tabla_clientes(self):
        self.tabla_select = 1
        self.tabla_cliente(tabla=self.tabla_select,busqueda=0)
        self.deshabilitar_campo()
        self.lable_buscar_clientes()
        self.deshabilitar_actualizar()
        self.boton_actualizar.config(state='disable')
        self.boton_modo_buscar.config(state='active')

    def select_tabla_proyecto(self):
        self.tabla_select=2
        self.deshabilitar_campo()
        self.deshabilitar_actualizar()
        self.label_buscar_proyecto()
        self.tabla_cliente(tabla=self.tabla_select,busqueda=0)
        self.boton_modo_buscar.config(state='active')
        self.boton_actualizar.config(state='active')

    def select_tabla_derivacion(self):
        self.tabla_select = 3
        self.deshabilitar_campo()
        self.exit_buscar()
        self.deshabilitar_actualizar()
        self.boton_actualizar.config(state='disable')
        self.boton_modo_buscar.config(state='active')
        self.lable_buscar_derivación()
        self.tabla_cliente(tabla=self.tabla_select,busqueda=0)

    def modo_busqueda(self):
        if self.tabla_select == 1:
            self.habilitar_busqueda_clientes()
        elif self.tabla_select == 2:
            self.habilitar_campos()
        elif self.tabla_select == 3:
            self.habilitar_busqueda_clientes()
        self.deshabilitar_actualizar()
```

```
        self.vacio_busqueda()

    def exit_buscar(self):
        if self.tabla_select == 2:
            self.boton_actualizar.config(state='active')
        self.deshabilitar_campo()
        self.vacio_busqueda()

    def info_lead(self):

self.mi_id_proyecto.set(self.tabla.item(self.tabla.selection())['values']
[1])

self.mi_facturacion.set(self.tabla.item(self.tabla.selection())['values']
[10])

self.mi_tipo_cliente.set(self.tabla.item(self.tabla.selection())['values'
][3])

self.mi_tipo_opor.set(self.tabla.item(self.tabla.selection())['values'][2
])

self.mi_tipo_entidad.set(self.tabla.item(self.tabla.selection())['values'
][4])

self.mi_inversion.set(self.tabla.item(self.tabla.selection())['values'][5
])

self.mi_sector.set(self.tabla.item(self.tabla.selection())['values'][9])

self.mi_id_cliente.set(self.tabla.item(self.tabla.selection())['values'][
0])

self.mi_estatus.set(self.tabla.item(self.tabla.selection())['values'][13]
)

self.mi_partner.set(self.tabla.item(self.tabla.selection())['values'][14]
)

    def vacio_busqueda(self):
        self.mi_id_proyecto.set('')
        self.mi_facturacion.set('')
        self.mi_tipo_cliente.set('')
        self.mi_tipo_opor.set('')
        self.mi_tipo_entidad.set('')
        self.mi_inversion.set('')
        self.mi_sector.set('')
        self.mi_id_cliente.set('')
        self.mi_estatus.set('')

    def busqueda(self):
```

```
        if self.tabla_select == 1:
            self.query_busqueda = 1
        elif self.tabla_select == 2:
            self.query_busqueda = 2
        elif self.tabla_select == 3:
            self.query_busqueda = 3

        print(self.tabla_select,self.query_busqueda)

        filtros =
[self.desple_estatus.get(),self.desple_tipo_cliente.get(),self.desple_tip
o_entidad.get(),self.desple_inversion.get(),self.desple_sector.get(),self
.desple_facturacion.get(),self.entry_tipo_opor.get(),self.entry_id_client
e.get()]

self.tabla_cliente(tabla=self.tabla_select,busqueda=self.query_busqueda,f
iltros=filtros)


    def tabla_cliente(self,tabla,busqueda,filtros = 'None'):

        conexion = Conexion_DB()

        #recuperar lista

        self.lista_datos = listar(tabla,busqueda,filtros)
        self.encabezado = columnas_tabla(tabla,busqueda,filtros)

        try:
            self.tabla.destroy()
            self.tabla =
ttk.Treeview(self,columns=self.encabezado,selectmode='browse',show='headi
ngs')

self.tabla.grid(row=4,column=0,columnspan=6,sticky='nse',padx=10,pady=10)
            self.scroll_y =
ttk.Scrollbar(self,orient='vertical',command=self.tabla.yview)
            self.scroll_y.grid(row=4,column=5,sticky='nse',padx=5)
        except:
            self.tabla =
ttk.Treeview(self,columns=self.encabezado,selectmode='browse',show='headi
ngs')

self.tabla.grid(row=4,column=0,columnspan=6,sticky='nse',padx=10,pady=10)
            self.scroll_y =
ttk.Scrollbar(self,orient='vertical',command=self.tabla.yview)
            self.scroll_y.grid(row=4,column=5,sticky='nse',padx=5)

        dim_col = int(1200/len(self.encabezado))
```

```
        for i in self.encabezado:
            self.tabla.column(i,anchor='w',width=dim_col )
            self.tabla.heading(i,text=i)

        for i in self.lista_datos:
            self.tabla.insert('','end',iid=i[0],text=i[0],values=list(i))

        contar = len(self.lista_datos)


    def derivar(self):
        info =
[self.mi_id_proyecto.get(),self.mi_id_cliente.get(),self.mi_partner.get()
,2,self.mi_fecha.get()]
        derivacion(campos_derivacion= info)
```

10.5 CÓDIGO DE CONSULTAS SQL

```
from .conexion_db import Conexion_DB
from tkinter import messagebox
from aux_funciones import buscar_partner

def create_table():
    conexion = Conexion_DB()

    sql = """
    CREATE TABLE prueba(
        id_cliente int auto_increment primary key,
        nombre varchar(200),
        apellido varchar(200),
        fecha varchar(200)

    );
    """
    try:
        conexion.cursor.execute(sql)
        conexion.cerrar()
        titulo = 'Crear Registro'
        mensaje = 'Se creo la tabla correctamente'
        messagebox.showinfo(title=titulo,message=mensaje)
    except:
        titulo = 'Crear Registro'
        mensaje = 'La tabla ya existe'
        messagebox.showwarning(title=titulo,message=mensaje)

def borrar_tabla():
    conexion = Conexion_DB()
```

```
    sql="""DROP TABLE prueba"""

    try:
        conexion.cursor.execute(sql)
        conexion.cerrar()
        titulo = 'Borrar Registro'
        mensaje = 'Se borro la tabla correctamente'
        messagebox.showinfo(title=titulo,message=mensaje)
    except:
        titulo = 'Borrar Registro'
        mensaje = 'La tabla no existe'
        messagebox.showwarning(title=titulo,message=mensaje)

class Cliente:
    def __init__(self,nombre,apellido,fecha) -> None:
        self.id = None
        self.nombre = nombre
        self.apellido = apellido
        self.fecha = fecha

    def __str__(self) -> str:
        return f'cliente {self.nombre}. {self.apellido}, {self.fecha}'

def guardar(cliente):

    conexion = Conexion_DB()

    sql = f"""
    INSERT INTO prueba (nombre,apellido,fecha)
    VALUES (%s,%s,%s);
    """
    values = (cliente.nombre,cliente.apellido,cliente.fecha)

    try:
        conexion.cursor.execute(sql,values)
        conexion.cerrar()
    except:
        titulo = 'Error de insert dato'
        mensaje = 'La tabla no existe, no puedes añadir datos'
        messagebox.showwarning(title=titulo,message=mensaje)

def listar(tabla,busqueda,filtros = 'None'):
    conexion = Conexion_DB()

    lista_cliente = []

    sql = selector_sql(tabla=tabla,busqueda=busqueda,filtros=filtros)

    try:
        conexion.cursor.execute(sql)
```

```
        lista_cliente = conexion.cursor.fetchall()
        conexion.cerrar()
    except:
        titulo = 'Error de mostrar dato'
        mensaje = 'La tabla no existe, no puedes mostrase los datos'
        messagebox.showwarning(title=titulo,message=mensaje)

    return lista_cliente


def columnas_tabla(tabla,busqueda,filtros = 'None'):
    conexion = Conexion_DB()

    x=[]

    sql = selector_sql(tabla=tabla,busqueda=busqueda,filtros = filtros)

    try:
        conexion.cursor.execute(sql)
        lista_heads = conexion.cursor.column_names
        lista_heads = list(lista_heads)
        for i in lista_heads:
            x.append(i)
    except:
        titulo = 'Error de mostrar dato'
        mensaje = 'La tabla no existe, 3 '
        messagebox.showwarning(title=titulo,message=mensaje)

    return x

def editar(cliente,id_proyecto):
    conexion = Conexion_DB()

    sql = f"""
    UPDATE proyecto
    SET partner = {cliente.partner}, fecha_derivacion = {cliente.fecha}
    WHERE id_proyecto = {id_proyecto};
    """
    try:
        conexion.cursor.execute(sql)
        conexion.cerrar()
    except:
        titulo = 'Error de actualizacion'
        mensaje = 'Los datos son erroneos '
        messagebox.showwarning(title=titulo,message=mensaje)


def selector_sql(tabla,busqueda,filtros = 'None'):

    if tabla == 1 and busqueda == 0:
        sql = """
```

```
            SELECT
            id_cliente,
            nombre_empresa,
            cif_nif,
            nombre_entidad,
            nombre_cliente,
            apellido,
            correo_cliente,
            fecha_lead
            FROM tfm.cliente
            inner join entidad ON cliente.entidad = entidad.id_entidad
        """
    elif tabla == 2 and busqueda == 0:
        sql = """
            SELECT
                proyecto.id_cliente,
                proyecto.id_proyecto,
                tipo_oportunidad.tipo_oportunidad,
                tipo_cliente.tipo_cliente,
                entidad.nombre_entidad,
                proyecto.inversión,
                proyecto.descripción_empresa,
                proyecto.interes,
                proyecto.ayuda_aplicada,
                sector.sector,
                facturacion.facturacion,
                empleados.empleados,
                ccaa.ccaa,
                estatus.estatus,
                partner.partner
                FROM proyecto
                JOIN tipo_oportunidad ON proyecto.tipo_oportunidad =
tipo_oportunidad.id_oportunidad
                JOIN tipo_cliente ON proyecto.tipo_cliente =
tipo_cliente.id_tipo
                JOIN sector ON proyecto.sector = sector.id_sector
                join ccaa on proyecto.ccaa = ccaa.id_ccaa
                JOIN facturacion on proyecto.facturacion =
facturacion.id_facturacion
                join empleados on proyecto.numero_empleados =
empleados.id_empleados
                JOIN cliente ON proyecto.id_cliente = cliente.id_cliente
                join entidad on cliente.entidad = entidad.id_entidad
                JOIN estatus ON proyecto.estatus = estatus.id_estatus
                join partner on proyecto.partner = partner.id_partner
        """
        print('proyecto headings')
    elif tabla ==3 and busqueda == 0:
        sql = """
        SELECT
        derivacion.id_derivacion,
```

```
        derivacion.id_proyecto,
        derivacion.id_cliente,
        cliente.nombre_cliente,
        partner.partner,
        estatus.estatus,
        derivacion.fecha_derivación,
        derivacion.fecha_fin
        FROM tfm.derivacion
        JOIN cliente ON derivacion.id_cliente = cliente.id_cliente
        JOIN partner ON derivacion.partner = partner.id_partner
        JOIN estatus ON derivacion.estatus = estatus.id_estatus
        """
    elif tabla == 2 and busqueda == 2:
        #condiciones de busqueda
        contar = 0
        for i in filtros:
            if i != '':
                contar += 1

        print(contar)

        #sql código
        if contar == 0:
            sql = f"""
                SELECT
                    proyecto.id_cliente,
                    proyecto.id_proyecto,
                    tipo_oportunidad.tipo_oportunidad,
                    tipo_cliente.tipo_cliente,
                    entidad.nombre_entidad,
                    proyecto.inversión,
                    proyecto.descripción_empresa,
                    proyecto.interes,
                    proyecto.ayuda_aplicada,
                    sector.sector,
                    facturacion.facturacion,
                    empleados.empleados,
                    ccaa.ccaa,
                    estatus.estatus,
                    partner.partner
                    FROM proyecto
                    JOIN tipo_oportunidad ON proyecto.tipo_oportunidad =
tipo_oportunidad.id_oportunidad
                    JOIN tipo_cliente ON proyecto.tipo_cliente =
tipo_cliente.id_tipo
                    JOIN sector ON proyecto.sector = sector.id_sector
                    join ccaa on proyecto.ccaa = ccaa.id_ccaa
                    JOIN facturacion on proyecto.facturacion =
facturacion.id_facturacion
                    join empleados on proyecto.numero_empleados =
empleados.id_empleados
```

```
                    JOIN cliente ON proyecto.id_cliente =
cliente.id_cliente
                    join entidad on cliente.entidad = entidad.id_entidad
                    JOIN estatus ON proyecto.estatus = estatus.id_estatus
                    join partner on partner.id_partner = proyecto.partner
            """
        else:

            where = 'WHERE '

            j = 0
            for i in filtros:
                if j == 0 and i != '':
                    where += f"estatus.estatus LIKE '%{i}%' "
                elif j == 1 and i != '':
                    where += f"tipo_cliente.tipo_cliente LIKE '%{i}%' "
                elif j == 2 and i != '':
                    where += f"nombre_entidad LIKE '%{i}%' "
                elif j == 3 and i != '':
                    where += f"inversión LIKE '%{i}%' "
                elif j == 4 and i != '':
                    where += f"sector.sector LIKE '%{i}%'' "
                elif j == 5 and i != '':
                    where += f"facturacion.facturacion LIKE '%{i}%' "
                elif j == 6 and i != '':
                    where += f"tipo_oportunidad.tipo_oportunidad LIKE
'%{i}%' "
                elif j == 7 and i != '':
                    where += f"cliente.id_cliente LIKE '{i}' "
                j += 1

                if contar > 1 and i != '':
                    contar -= 1
                    where += ' AND '


            sql = f"""
                SELECT
                    proyecto.id_cliente,
                    proyecto.id_proyecto,
                    tipo_oportunidad.tipo_oportunidad,
                    tipo_cliente.tipo_cliente,
                    entidad.nombre_entidad,
                    proyecto.inversión,
                    proyecto.descripción_empresa,
                    proyecto.interes,
                    proyecto.ayuda_aplicada,
                    sector.sector,
                    facturacion.facturacion,
                    empleados.empleados,
                    ccaa.ccaa,
```

```
                    estatus.estatus,
                    partner.partner
                    FROM proyecto
                    JOIN tipo_oportunidad ON proyecto.tipo_oportunidad =
tipo_oportunidad.id_oportunidad
                    JOIN tipo_cliente ON proyecto.tipo_cliente =
tipo_cliente.id_tipo
                    JOIN sector ON proyecto.sector = sector.id_sector
                    join ccaa on proyecto.ccaa = ccaa.id_ccaa
                    JOIN facturacion on proyecto.facturacion =
facturacion.id_facturacion
                    join empleados on proyecto.numero_empleados =
empleados.id_empleados
                    JOIN cliente ON proyecto.id_cliente =
cliente.id_cliente
                    join entidad on cliente.entidad = entidad.id_entidad
                    JOIN estatus ON proyecto.estatus = estatus.id_estatus
                    join partner on partner.id_partner = proyecto.partner
                    {where}
            """
    elif tabla == 1 and busqueda == 1:
        #condiciones de busqueda
        contar = 0
        for i in filtros:
            if i != '':
                contar += 1

        print(contar)

        #sql código
        if contar == 0:
            sql = f"""
                SELECT
                id_cliente,
                nombre_empresa,
                cif_nif,
                nombre_entidad,
                nombre_cliente,
                apellido,
                correo_cliente,
                fecha_lead
                FROM tfm.cliente
                inner join entidad ON cliente.entidad =
entidad.id_entidad
            """
        else:

            where = 'WHERE '

            j = 0
            for i in filtros:
```

```
                if j == 2 and i != '':
                    where += f"nombre_empresa LIKE '%{i}%' "
                elif j == 3 and i != '':
                    where += f"cif_nif LIKE '%{i}%' "
                elif j == 4 and i != '':
                    where += f"correo_cliente LIKE '%{i}%' "
                elif j == 6 and i != '':
                    where += f"nombre_cliente LIKE '%{i}%' "

                j += 1

                if contar > 1 and i != '':
                    contar -= 1
                    where += ' AND '


            sql = f"""
                SELECT
                id_cliente,
                nombre_empresa,
                cif_nif,
                nombre_entidad,
                nombre_cliente,
                apellido,
                correo_cliente,
                fecha_lead
                FROM tfm.cliente
                inner join entidad ON cliente.entidad =
entidad.id_entidad
                {where}
            """

    print(sql)

    sql = str(sql)
    return sql

def derivacion(campos_derivacion):

    partner = buscar_partner(campos_derivacion[2])

    sql1 = f"""
    INSERT INTO tfm.derivacion (id_proyecto, id_cliente, partner,
estatus,fecha_derivación) VALUES
({campos_derivacion[0]},{campos_derivacion[1]},{partner},{campos_derivaci
on[3]},'{campos_derivacion[4]}');
    """
    sql2 = f"""
```

```
    UPDATE tfm.proyecto SET partner = '{partner}', estatus =
'{campos_derivacion[3]}', fecha_derivacion = '{campos_derivacion[4]}'
WHERE (id_proyecto = '{campos_derivacion[0]}');
    """

    print(sql1)
    conexion = Conexion_DB()

    try:
        conexion.cursor.execute(sql1)
        conexion.cursor.execute(sql2)
        conexion.cerrar()
        titulo = 'Derivcaión Correcta'
        mensaje = 'El lead se ha derivado correctamente'
        messagebox.showinfo(title=titulo,message=mensaje)
    except:
        titulo = 'Derivcaión Correcta'
        mensaje = 'se han introducidos campos erroneos'
        messagebox.showwarning(title=titulo,message=mensaje)
```

10.6 CÓDIGO FUNCIONES AUX PARA SELCCIONAR ID PARA LOS INSERT INTO

```
def tip_oportunidad_bien(dato):
    if dato == 'Tramitación masiva':
        return 1
    if dato == 'Asesoramiento personalizado integral':
        return 2
    if dato == 'Asesoramiento personalizado a proyectos':
        return 3
    return 4

def tip_cliente_bien(dato):
    if dato == 'Emprendedor':
        return 1
    if dato == 'Autónomo':
        return 2
    if dato == 'PYME':
        return 3
    if dato == 'Gran Empresa':
        return 4
    if dato == 'Asociaciones empresariales':
        return 5
    if dato == 'ONG':
        return 6
    if dato == 'Centros de investigación':
```

```
        return 7
    return 8


def sector_bien(dato):
    if dato == 'A - Agricultura, ganadería, silvicultura y pesca':
        return 1
    if dato == 'B - Industrias extractivas':
        return 2
    if dato == 'C - Industria manufacturera':
        return 3
    if dato == 'D - Suministro de energía eléctrica, gas, vapor y aire
acondicionado':
        return 4
    if dato == 'E - Suministro de agua, actividades de saneamiento,
gestión de residuos y descontaminación':
        return 5
    if dato == 'F - Construcción':
        return 6
    if dato == 'G - Comercio al por mayor y al por menor; reparación de
vehículos de motor y motocicletas':
        return 7
    if dato == 'H - Transporte y almacenamiento':
        return 8
    if dato == 'I - Hostelería':
        return 9
    if dato == 'J - Información y comunicaciones':
        return 10
    if dato == 'K - Actividades financieras y de seguros':
        return 11
    if dato == 'L - Actividades inmobiliarias':
        return 12
    if dato == 'M - Actividades profesionales, científicas y técnicas':
        return 13
    if dato == 'N - Actividades administrativas y servicios auxiliares':
        return 14
    if dato == 'O - Administración Pública y defensa; Seguridad Social
obligatoria':
        return 15
    if dato == 'P - Educación':
        return 16
    if dato == 'Q - Actividades sanitarias y de servicios sociales':
        return 17
    if dato == 'R - Actividades artísticas, recreativas y de
entretenimiento':
        return 18
    if dato == 'S - Otros servicios':
        return 19
    if dato == 'T - Actividades de los hogares como empleadores de
personal doméstico; actividades de los hogares como productores de bienes
y servicios para uso propio':
```

```
        return 20
    if dato == 'U -Actividades de organizaciones y organismos
extraterritoriales':
        return 21
    return 22
def facturacion_bien(dato):
    if dato == 'Menos de 2 M€':
        return 1
    if dato == 'De 2 M€ a 5 M€':
        return 2
    if dato == 'De 5 M€ a 10 M€':
        return 3
    if dato == 'De 10 M€ a 50 M€':
        return 4
    if dato == 'Mas de 50 M€':
        return 5
    return 6
def numero_empleados_bien(dato):
    if dato == 'De 1 a 2':
        return 1
    if dato == 'De 3 a 9 ':
        return 2
    if dato == 'De 10 a 49':
        return 3
    if dato == 'Mas de 50':
        return 4
    return 5
def provincia(dato):
    if dato == 'Almería':
        return 1
    if dato == 'Cádiz':
        return 2
    if dato == 'Córdoba':
        return 3
    if dato == 'Granada':
        return 4
    if dato == 'Huelva':
        return 5
    if dato == 'Jaén':
        return 6
    if dato == 'Málaga':
        return 7
    if dato == 'Sevilla':
        return 8
    if dato == 'Huesca':
        return 9
    if dato == 'Teruel':
        return 10
    if dato == 'Zaragoza':
        return 11
    if dato == 'Oviedo':
```

```
        return 12
    if dato == 'Islas Baleares':
        return 13
    if dato == 'Baleares (Illes)':
        return 14
    if dato == 'Santa Cruz de Tenerife':
        return 15
    if dato == 'Las Palmas':
        return 16
    if dato == 'Santander':
        return 17
    if dato == 'Albacete':
        return 18
    if dato == 'Ciudad Real':
        return 19
    if dato == 'Cuenca':
        return 20
    if dato == 'Guadalajara':
        return 21
    if dato == 'Toledo':
        return 22
    if dato == 'Ávila':
        return 23
    if dato == 'Burgos':
        return 24
    if dato == 'León':
        return 25
    if dato == 'Salamanca':
        return 26
    if dato == 'Segovia':
        return 27
    if dato == 'Soria':
        return 28
    if dato == 'Valladolid':
        return 29
    if dato == 'Palencia':
        return 30
    if dato == 'Zamora':
        return 31
    if dato == 'Barcelona':
        return 32
    if dato == 'Girona':
        return 33
    if dato == 'Lleida':
        return 34
    if dato == 'Tarragona':
        return 35
    if dato == 'Alicante':
        return 36
    if dato == 'Castellón':
        return 37
```

```
    if dato == 'Valencia':
        return 38
    if dato == 'Badajoz':
        return 39
    if dato == 'Cáceres':
        return 40
    if dato == 'A Coruña':
        return 41
    if dato == 'Lugo':
        return 42
    if dato == 'Ourense':
        return 43
    if dato == 'Pontevedra':
        return 44
    if dato == 'Madrid':
        return 45
    if dato == 'Murcia':
        return 46
    if dato == 'Pamplona':
        return 47
    if dato == 'Navarra':
        return 48
    if dato == 'Bilbao':
        return 49
    if dato == 'San Sebastián':
        return 50
    if dato == 'Vitoria':
        return 51
    if dato == 'Guipúzcoa':
        return 52
    if dato == 'Vizcaya':
        return 53
    if dato == 'Álava':
        return 54
    if dato == 'Logroño':
        return 55
    if dato == 'La Rioja':
        return 56
    return 57
def ccaa_bien(dato):
    if dato == 'Andalucía':
        return 1
    if dato == 'Aragón':
        return 2
    if dato == 'Asturias':
        return 3
    if dato == 'Baleares':
        return 4
    if dato == 'Canarias':
        return 5
    if dato == 'Cantabria':
```

```
        return 6
    if dato == 'Castilla-La Mancha':
        return 7
    if dato == 'Castilla y León':
        return 8
    if dato == 'Cataluña':
        return 9
    if dato == 'Comunidad Valenciana':
        return 10
    if dato == 'Extremadura':
        return 11
    if dato == 'Galicia':
        return 12
    if dato == 'Madrid':
        return 13
    if dato == 'Murcia':
        return 14
    if dato == 'Navarra':
        return 15
    if dato == 'País Vasco':
        return 16
    if dato == 'La Rioja':
        return 17
    return 18
def buscar_estatus(estatus):
    if estatus =='08_Finalizado':
        return 5
    elif estatus == '05_Pendiente reunión con partner' or estatus ==
'06_Pendiente identificar ayuda' or estatus == '07_Pendiente
contratación':
        return 4
    elif estatus == '04_Pendiente aceptación partner':
        return 2
    else:
        return 1

def buscar_partner(partner):

    if partner =='7EXPERTS':
        return 1
    if partner =='ATEINSA':
        return 2
    if partner =='AYMING':
        return 3
    if partner =='BI Consulting':
        return 4
    if partner =='DYRECTO':
        return 5
    if partner =='EVALUE':
        return 6
    if partner =='FACTORYDEA':
```

```
        return 7
    if partner =='FI Group':
        return 8
    if partner =='IDITEK':
        return 9
    if partner =='INCOTEC':
        return 10
    if partner =='INVEPAT':
        return 11
    if partner =='SULAYR':
        return 12
    if partner =='SF Consultores':
        return 13
    if partner =='ZABALA':
        return 14
    if partner =='LEYTON':
        return 15
    if partner =='SOOF':
        return 16
    if partner =='KVAR':
        return 17
    if partner =='QDQ':
        return 18
    else:
        return 19

def fechas_bien(lead,oport):

    lead = lead.split(' ')
    oport = oport.split(' ')
    return(lead[0],oport[0])
```

ANEXO II