Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Computación

Refinamiento del proceso ETL para la ingesta automatizada de datos financieros en un Data Lake

Autor:

Juan Rubio Gómez

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Computación

Refinamiento del proceso ETL para la ingesta automatizada de datos financieros en un Data Lake

Autor:

Juan Rubio Gómez

Tutores:

Dr. Quiliano Isaac Moro Sancho

Paula Pastor Gómez

Resumen

En el actual panorama empresarial, la capacidad de recoger, almacenar y analizar grandes volúmenes de datos financieros de manera eficiente es crucial para la toma de decisiones estratégicas y alcanzar los objetivos comerciales. Es por eso que cualquier empresa hoy en día, independientemente de su tamaño, recopila y almacena una inmensa cantidad de datos. Para ello, el uso de almacenes de datos como los *Data Lakes* se ha convertido en una parte fundamental de este proceso, permitiendo la consolidación de datos de diversas fuentes de origen para su posterior análisis. Sin embargo, la eficacia de un almacén depende en gran medida de la calidad y la integridad de los datos que se ingieren en él.

El proceso de Extracción, Transformación y Carga (ETL) desempeña un papel fundamental en la ingesta de datos. Este proceso, dividido en tres fases, implica la extracción de datos de múltiples fuentes, su adecuada transformación para alcanzar la estructura y formato necesaria y, finalmente, su carga en el almacén de datos destino.

El presente Trabajo de Fin de Grado se enfoca en el diseño y la implementación de una lógica de transformación para un proceso ETL en la ingestión automatizada de datos financieros en un Data Lake. En esta memoria se explorarán diversas estrategias y metodologías para perfeccionar la calidad y la eficiencia de este proceso, con el objetivo de obtener un contexto global de este tipo de procesos y conocer las tecnologías y metodologías más comunes. Además, se realizarán análisis detallados y pruebas para validar la efectividad y la corrección de la lógica de transformación propuesta.

Palabras clave: *Big Data*, *Data Lake*, ETL, lógica de transformación de datos, validación, automatización, entidad bancaria, *cloud*.

Abstract

In today's business landscape, the ability to collect, store and analyze large volumes of financial data efficiently is crucial to making strategic decisions and achieving business objectives. That is why any company today, regardless of its size, collects and stores an immense amount of data. To this end, the use of data warehouses such as Data Lakes has become a fundamental part of this process, enabling the consolidation of data from various sources of origin for further analysis. However, the effectiveness of a warehouse is highly dependent on the quality and integrity of the data being ingested into it.

The Extract, Transform and Load (ETL) process plays a key role in data ingestion. This process, divided into three phases, involves the extraction of data from multiple sources, its own transformation to achieve the required structure and format, and finally its loading into the target data warehouse.

This final degree project focuses on the design and implementation of a transformation logic for an ETL process in the automated ingestion of financial data in a Data Lake. In this document we will explore different strategies and methodologies to improve the quality and efficiency of this process, in order to obtain a global context of this type of processes and to know the most common technologies and methodologies. In addition, detailed analysis and testing will be performed to validate the effectiveness and correctness of the proposed transformation logic.

Keywords: Big Data, Data Lake, ETL, data transformation logic, validation, automation, bank, cloud.

Agradecimientos

Quiero agradecer de corazón a mi familia por su cariño y apoyo inquebrantable a lo largo de toda mi vida. También quiero agradecer a las personas más cercanas a mí por estar siempre conmigo.

Por último, quiero agracecer a Quiliano Isaac Moro Sancho por guiarme en este trabajo y a Diego García Álvarez por sus consejos, así como a Iraide Puente Bustinza y a Paula Pastor Gómez por su tiempo y ayuda constante a la hora de aprender, tanto en mi periodo de prácticas como en el periodo de desarrollo de este TFG.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En la actualidad, las empresas modernas tienen que lidiar con enormes volúmenes de datos que se generan tan masivamente que resultan difíciles de procesar y manejar con tecnologías convencionales. Este fenómeno se conoce como *Big Data* y juega un papel crucial en la toma de decisiones y el diseño de estrategias a seguir, así como en la optimización de procesos o incluso en la investigación científica. Pero no es suficiente con dar cabida y almacenar estos millones de datos, si no que su valor únicamente radica en la capacidad de explotarlos y analizarlos y, para ello, gestionar esta gran cantidad de datos adecuadamente es un proceso clave.

Los avances en las tecnologías de almacenamiento y el almacenamiento en la nube han permitido manejar y analizar grandes cantidades de datos de manera automatizada y en tiempo real, proporcionando una comprensión más profunda de estos datos, eliminando casi por completo errores humanos y liberando recursos. Dadas estas circustancias, parece crítico para una entidad automatizar el procesamiento de datos en sus proyectos, para así maximizar su utilidad y calidad.

1.1. Contexto

Este proyecto de fin de grado está realizado en convenio con la empresa *NTT DATA Spain, S.L.U.*, aplicándose a un ejemplo real de una empresa de banca y finanza multinacional. Por ello, los datos implicados en él son de carácter confidencial y no pueden ser revelados en ningún punto de esta memoria, omitiéndose o sustituyéndose toda la información sensible, desde los propios datos y desarrollos hasta el nombre de la entidad bancaria o información relativa a ella.

El proyecto se centrará en documentar el proceso del dato desde un estado crudo o en bruto (*raw data*) hasta un estado en el que este si puede ser analizado y explotado, dentro de nuestro marco real de datos financieros relacionados con proyectos internacionales de sostenibilidad. Esto lo logramos a través de la creación y elección de una serie de transformaciones (durante esta

memoria nombraremos este proceso como *lógica* o *lógica de transformación*) que permiten obtener los datos en el formato y estructura deseada, para realizar una posterior ingesta en un *Date Lake* de manera automatizada. La automatización de este tipo de procesos genera en las empresas una mayor productividad, fiabilidad y eficiencia, reduciendo el tiempo dedicado a tareas repetitivas y propensas a errores.

Dentro de la empresa, me ha guidado mi co-tutora, Paula Pastor Gómez, compañera de equipo durante mis prácticas de empresa curriculares y extracurriculares.

1.2. Motivación

Los motivos que me han guiado y animado a realizar este proyecto son, principalmente:

- Adentrarme en el mundo del *Big Data* con un caso práctico real.
- Comprender el funcionamiento de las tecnologías asociadas al *Big Data* y a los procesos ETL.
- Explorar y comprender los procesos ETL y la automatización en el procesamiento de datos durante el proceso de ingesta.
- Participar activamente en un proyecto real con un impacto significativo en la empresa, empleando sus metodologías y nutriéndome de su entorno.

1.3. Objetivos

Los objetivos fundamentales de este proyecto son los siguientes:

- Desarrollar y documentar el proceso ETL que describe cómo los datos financieros, desde un estado crudo donde la información no es utilizable, son transformados hasta alcanzar un estado en el cual pueden ser explotados y analizados para obtener información significativa para una entidad bancaria.
- Desarrollar la fase de transformación de un proceso ETL, enfocado en la realización del análisis, diseño y desarrollo de la lógica de transformación, con su posterior validación por la cual el dato crudo se transforma a un formato estandarizado y regularizado.
- Validar la ingesta de la fase de carga del proceso ETL de un *DataFrame*, generado por la lógica de la fase de transformación desarrollada previamente, en una tabla de una plataforma *Big Data*, con el fin de permitir su explotación para análisis, creación de *dashboards*, informes y otras actividades.

1.4. Estructura del Documento

La estructura de ese documento, que también se puede apreciar en el *Índice General*, se puede esquematizar según los siguientes capítulos:

- **Capítulo 1. Introducción:** Es el capítulo actual. En este capítulo se introduce el proyecto de fin de grado, su contexto, motivación y objetivos, así como la estructura de la presente memoria donde se documenta el proyecto.
- **Capítulo 2. Planificación, Costes y Riesgos:** En él se describe la metodología empleada en el desarrollo del proyecto y la planificación inical, se realiza un análisis de los costos y de los posibles riesgos asociados.
- **Capítulo 3. Marco Teórico:** Durante este capítulo se tratan los conceptos claves teóricos relacionados con el proyecto.
- **Capítulo 4. Marco Tecnológico:** Durante este capítulo se tratan las principales herramientas y tecnologías empleadas durante el trabajo, así como un estudio global a las opciones tecnológicas en el mercado para abordar este tipo de proyectos.
- **Capítulo 5. Planteamiento del Problema:** En este capítulo se explica detalladamente el contexto del problema y las necesidades y requisitos propios a cumplir en la solución.
- **Capítulo 6. Desarrollo de la Solución:** A lo largo de este capítulo se describe la solución del problema y los pasos detallados para llegar a ella. También se muestra la posterior validación de la solución obtenida.
- **Capítulo 7. Descripción de las Iteraciones:** Se presenta el progreso real del proyecto en comparación con la planificación inicial estipulada en la primera sección del capítulo 2.
- **Capítulo 8. Conclusiones y Trabajo Futuro:** En el trascurso de este capítulo se presentan las conclusiones obtenidas tanto con la solución obtenida como con el desarrollo del proyecto. Además, se destacan posibles áreas de mejora y se exploran puertas abiertas para futuras expansiones del proyecto.
- **Bibliografía:** Se referencian de las fuentes de información utilizadas durante el proyecto.
- **Anexos:** Se proporciona material complementario y detallado que respalda y enriquece la comprensión del trabajo realizado. Este material consiste en una sección de acrónimos con los principales términos usados en la memoria, una guía básica de uso de *PySpark*, la tecnología principal usada en el proyecto, y las intrucciones con el pseudocógido y la validación correspondiente a la lógica de transformación desarrollada.

Capítulo 2

Planificación, Costes y Riesgos

Elegir una metodología de trabajo y aplicarla es un proceso fundamental para lograr el éxito en cualquier proyecto. Por ello, en este capítulo se trata sobre la metodología de trabajo elegida, donde se detalla la planificación inicial de hitos y tareas a lo largo del tiempo, así como el análisis de costes y riesgos del proyecto.

2.1. Metodología y Planificación Inicial

La metodología de trabajo empleada está impuesta por la empresa y es de tipo ágil *Scrum*. Esta metodología permite cierta flexibilidad para adaptarse a los cambios y ajustar estimaciones de tiempos y tareas a medida que el proyecto avanza, por lo que favorece al desarrollo de este TFG, ya que al realizar el proyecto en común con la empresa, podría tener dificultades o retrasos fuera de mi alcance. Además, *Scrum* se centra en la entrega iterativa e incremental de valor, lo que permite mostrar avances tangibles en cada etapa del proyecto de forma temprana, facilitando así una rápida retroalimentación con cualquiera de mis tutores y detectando de posibles desviaciones o complicaciones en el objetivo.

2.1.1. Descomposición en Sprints

La característica principal de esta metodología consiste en que permite dar un enfoque iterativo al desarrollo del proyecto mediante entregas y cumplimiento de tareas. Teniendo en cuenta que la estimación de tiempo total es aproximadamente 300 horas de trabajo y el comienzo del proyecto es el 04 de marzo de 2024, calculando que el tiempo aproximado de dedicación es de 25 horas semanales (5 horas/día de lunes a viernes) obtenemos un total de un plazo doce semanas. Estas doce semanas se repartirán en seis *sprints* de dos semanas cada uno, estimando como fecha de fin proyecto el 26 de mayo y dejando cabida a un posible *sprint* más (el fin sería el 09 de junio)

como margen por posibles inconvenientes o retrasos. Cabe resaltar de nuevo que esto es una estimación inicial que puede verse alterada por diversos factores internos o externos a mí, estudiados posteriormente en el análisis de riesgos.

Cada *sprint*, de duración de dos semanas, comenzará con un *Sprint Planning* en el que mi tutora de empresa y yo definiremos los objetivos del *sprint*, seleccionando las tareas que se abordarán en él. Estas tareas se acumulan en un *Product Backlog*. Al final del *sprint* se llevará a cabo una breve reunión para realizar *Sprint Review* junto a un *Sprint Retrospective* donde yo, junto a mi tutora de empresa Paula, mostraré los avances en el proyecto y las complicaciones durante el *sprint*, con idea de obtener retroalimentación sobre puntos a mejorar y enfocarse en el siguiente *sprint*. Además, se irá informando con regularidad al tutor académico, Quiliano Isaac Moro Sancho, sobre el estado global del proyecto, avances en la memoria o, si procede, cuestiones de cualquier índole en la que pueda aportarme conocimiento e ideas.

Sprint	Fecha de Inicio	Fecha de Fin
Sprint 1	04/03/2024	17/03/2024
Sprint 2	18/03/2024	31/03/2024
Sprint 3	01/04/2024	14/04/2024
Sprint 4	15/04/2024	28/04/2024
Sprint 5	29/04/2024	12/05/2024
Sprint 6	13/05/2024	26/05/2024

Tabla 2.1: Planificación de los sprints

2.1.2. Descomposición del Trabajo y Diagrama de Gantt

Se ha elaborado un diagrama de descomposición del trabajo (WBS) para una mejor comprensión de la composición del proyecto y las tareas asociadas. En este esquema se incluyen las actividades principales que abarca el proyecto, como se puede consultar en la figura 2.1.

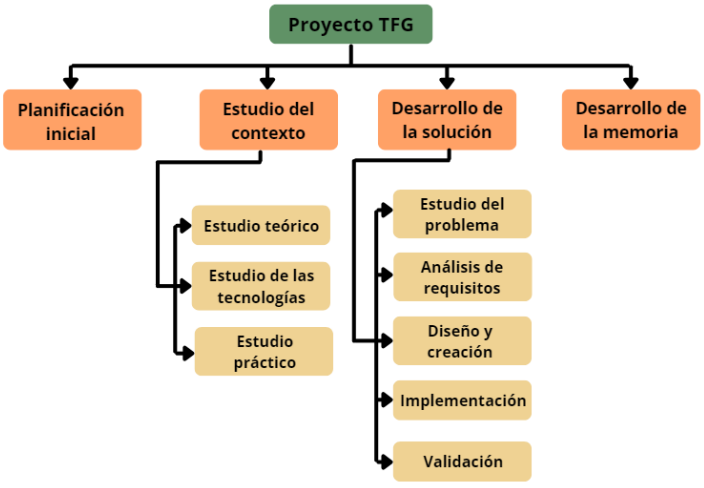

Figura 2.1: Diagrama de Descomposición del Trabajo (WBS)

También se ha realizado la identificación y planificación de las tareas de este proyecto estableciendo un plan de tareas a través de un **Diagrama de Gantt**, que puede observarse en la figura 2.2.

Como se menciona en la sección 2.1.1, el proyecto está estimado en 300 horas de duración, comenzando el día 04 de marzo de 2024 y con un final programado para el 26 de mayo de 2024, manteniendo una holgura respecto a la fecha de entrega final por si se diera algún posible inconveniente o problema que cause algún retraso. En caso de que esto ocurriera, no habría ningún incoveniente en añadir un *sprint* adicional y alargar la duración del proyecto dos semanas más, aunque esto no esté contemplado en la planificación incial.

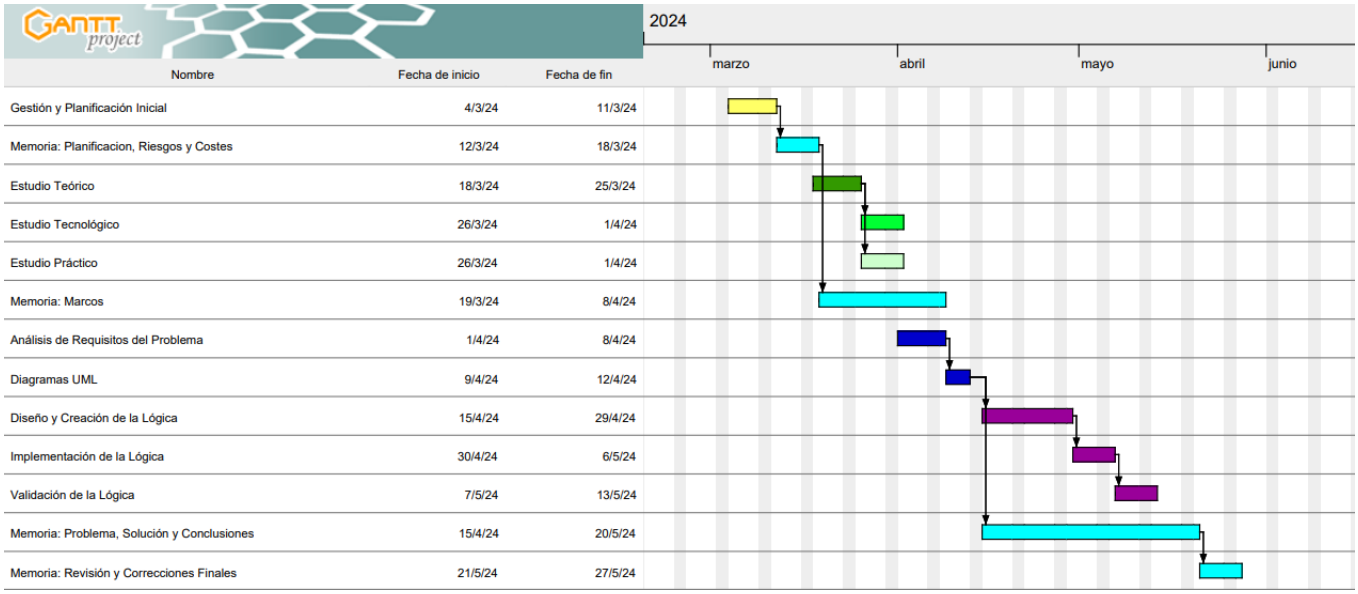

Figura 2.2: Diagrama de Gantt: planificación inicial

En base a este diagrama, se han fijado una serie de **hitos** del proyecto o fechas claves en las que deben estar completadas cierta serie de tareas, para controlar y regular el avance del proyecto en base a los objetivos. Estos hitos son:

- **04 de marzo de 2024**: fecha de inicio del proyecto.
- **01 de abril de 2024**: fecha límite para comprender el contexto teórico, tecnológico y práctico, incluyendo el manejo con soltura y destreza de *PySpark*, la tecnología clave del proyecto.
- **15 de abril de 2024**: fecha límite para haber recogido, analizado y estudiado los requisitos del problema, para poder comenzar el posterior diseño y desarrollo de la lógica.
- **06 de mayo de 2024**: fecha límite para haber completado la implementación de la lógica de transformación, para poder pasar a la parte de validación.
- **27 de mayo de 2024**: fecha límite de fin del proyecto.

2.2. Análisis de Costes

En cuanto al presupuesto económico estimado para el proyecto, se van a tener en cuenta diferentes factores. Primero, el personal, donde únicamente se necesita una persona que se supondrá como un recién graduado en Ingeniería Informática con salario medio en España [1] en torno a 12,18€/hora. Teniendo en cuenta la duración de 300 horas, supone un coste de alrededor de 3.652,71€.

Acerca del presupuesto en *hardware* se contabilizará como 0,00€, ya que todo lo necesario es proporcionado por la empresa. Aún así, para un cálculo aproximado del *hardware* utilizado, se cuenta con un ordenador portátil (Dell Latitude 5440 Portátil), con un precio de, aproximadamente, 1.554,96€ [2] y un monitor auxiliar (Monitor HP M24fw de 60,45 cm, 23,8") de 129,00€ [3]. Estos dispositivos *hardware* se emplearán en otros proyectos, por lo que carece de sentido contar con su coste al completo en esta estimación. Se puede estimar la vida útil de los dispositivos mencionados en aproximadamente 4 años, por lo que este proyecto de duración 12 semanas (3 meses) representa un 6,25 % del precio total de los dispositivos. Esto hace que el gasto en el ordenador portátil resulte en 97,19€ y el monitor en 8,06€, aproximadamente. No se empleará ningún otro dispositivo como podría ser un ratón o un teclado externo.

Los gastos en *software* se establecerán en 0,00€ ya que, al igual que el *hardware*, todo será proporcionado por la entidad bancaria interesada. Se tiene en cuenta también otros gastos extra, como transporte o conexión a Internet, fijándose en alrededor de 100€.

El desglose total de los costes estimados, con un resultado igual a 3.752,71€ como coste estimado total, puede verse reflejado en la tabla 2.2:

	Concepto	**Coste Total (€)**
Coste Personal	Ingeniero Informático *Junior*	3.652,71 €
Coste Hardware	Ordenador Portátil Dell Latitude 5440	0,00 €
	Monitor HP M24fw de 60,45 cm, 23,8"	0,00 €
Coste *Software*	Licencias y *Software*	0,00 €
Gastos Extra	Transporte, Internet, ...	100 €
	Total	**3.752,71 €**

Tabla 2.2: Análisis de costes final

2.3. Análisis de Riesgos

A continuación, se presentan los riesgos que pueden surgir durante la ejecución del proyecto y las estrategias sugeridas para prevenir, mitigar y si es posible eliminar sus efectos en caso de que se materialicen. El análisis se va a centrar en dos dimensiones: probabilidad y riesgo. La dimensión *probabilidad* aborda la posibilidad de que ocurra el evento o condición de riesgo y la dimensión *impacto* detalla el alcance de lo que sucedería y afectaría al proyecto si el riesgo se llegara a materializar [4]. Tanto *probabilidad* como *impacto* se medirán en una escala numérica de enteros comprendidios entre 1 y 5, siendo 1 el menor de los valores y 5 el mayor, indicando menor o mayor probabilidad o impacto, respectivamente, según la siguiente escala: "*Raro*", "*Improbable*", "*Moderado*", "*Probable*" y "*Casi Seguro*" para la probabilidad e "*Insignificante*", "*Menor*", "*Significativo*", "*Mayor*" y "*Grave/Severo*" para el impacto.

La exposición total o daño asociado a un riesgo se calcula como el producto de ambas métricas [5]. Gracias a este análisis se puede clasificar los riesgos encontrados según su daño relativo al proyecto en una matriz de riesgos, tal y como indica la figura 2.3, permitiendo conocer los riesgos con mayor daño global al proyecto para poder asignar más recursos a mitigarlos o incluso a prevenirlos.

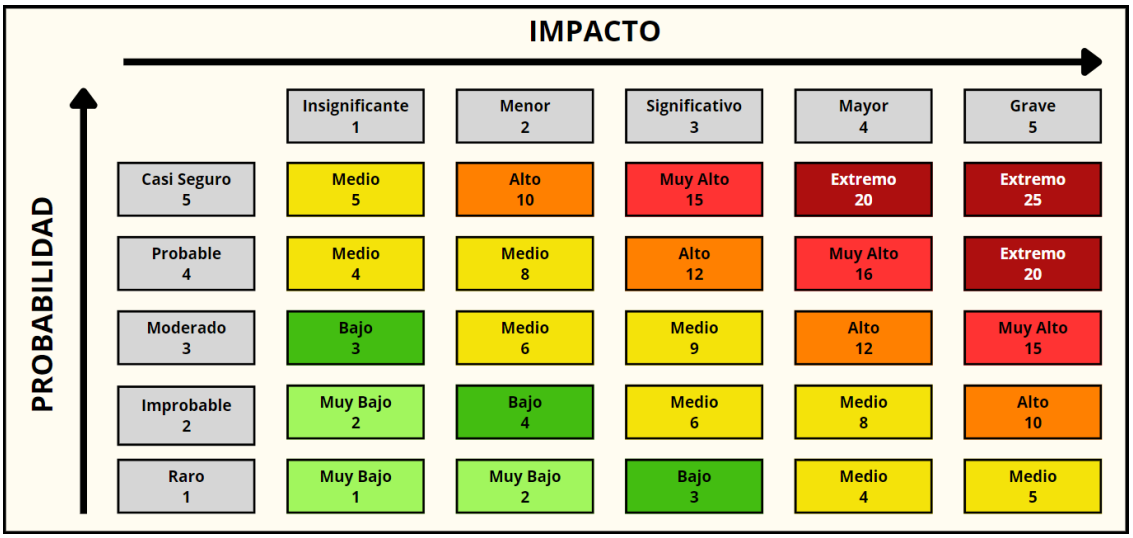

Figura 2.3: Matriz de clasificación riesgos

Para los riesgos analizados se detalla una breve descripción de este y sus medidas de prevención (para evitar que ocurra) y mitigación (una vez que ha ocurrido, minimizar su impacto lo máximo posible), además de los ya comentados impacto, probabilidad y daño o exposición. Dichos riesgos se muestran a continuación:

Riesgo 1. Estimaciones optimistas y/o mala organización

Riesgo 1	**Estimaciones optimistas y/o mala organización**
Descripción	La carga de trabajo de este proyecto debe de planificarse correctamente, ya que realizar una mala organización o estimación puede resultar en retrasos y errores.
Probabilidad	2 - Improbable
Impacto	4 - Mayor
Daño	8 - Medio
Prevención	Establecer un proceso de planificación detallado y fomentar la comunicacióncon los tutores para identificar y abordar posibles problemas de organización.
Mitigación	Revisar periódicamente el progreso del proyecto para ajustar las estimaciones según sea necesario.

Tabla 2.3: Riesgo 1

Riesgo 2. Recursos teóricos/conceptuales insuficientes

Riesgo 2	**Recursos teóricos/conceptuales insuficientes**
Descripción	Dado que el proyecto está enfocado en el cliente, parte del software y conocimientos empleados son específicos de la entidad bancaria y es posible que no pueda tener acceso a toda la información que desearía.
Probabilidad	3 - Moderado
Impacto	3 - Significativo
Daño	6 - Medio
Prevención	Realizar una búsqueda y evalución exhaustiva de los recursos teóricos disponibles antes de comenzar el proyecto, identificando posibles lagunas en el conocimiento para ver cómo abordarlas.
Mitigación	Fomentar una comunicación clara y continua con mis compañeros de empresa o incluso con la entidad bancaria para obtener acceso a los recursos y conocimientos necesarios.

Tabla 2.4: Riesgo 2

Riesgo 3. Uso de tecnologías desconocidas

Riesgo 3	**Uso de tecnologías desconocidas**
Descripción	Como se ha comentado, el software y las tecnologías empleadas pertenecen a la entidad bacaria, por lo que hay alguna de ellas fuera de mi conocimiento que requieren superar una curva de aprendizaje y adopción.
Probabilidad	5 - Casi Seguro
Impacto	3 - Significativo
Daño	15 - Muy Alto
Prevención	Recopilar toda la información y conocimientos posibles acerca de esas tecnologías para que la curva de aprendizje se produzca antes de comenzar el proyecto y no afecte a sus plazos.
Mitigación	Detectar ausencias de conocimiento en tecnologías y asignar tiempo durante los *sprints* para superarlas.

Tabla 2.5: Riesgo 3

Riesgo 4. Carga de trabajo adicional al proyecto

Riesgo 4	**Carga de trabajo adicional al proyecto**
Descripción	En paralelo a este proyecto, también se está realizando el Trabajo Fin de Grado de Estadística (12 *ECTS*) y se está cursando la asignatura Gramáticas y Lenguajes Formales (6 *ECTS*), además de las prácticas en el empresa colaboradora en este proyecto (12 *ECTS*) [6]. Esto supone una carga total (de 42 *ECTS*) bastante grande que puede influir negativamente con el desarrollo del trabajo.
Probabilidad	4 - Probable
Impacto	3 - Significativo
Daño	12 - Alto
Prevención	Evaluar cuidadosamente la carga de trabajo total y tratar de ajustar las responsabilidades académicas o laborales para evitar una sobrecarga.
Mitigación	Establecer una organización personal del tiempo y un cronograma realista que considere la carga de trabajo global.

Tabla 2.6: Riesgo 4

Riesgo 5. Imposibilidad de acceder al espacio de trabajo

Riesgo 5	**Imposibilidad de acceder al espacio de trabajo**
Descripción	Cabe la posibilidad de que, por causas externas a mí, tenga inhabilitado el acceso a mi entorno de desarrollo en algún periodo, ya bien sea por caídas del servidor o de la plataforma, actualizaciones o cualquier otro motivo.
Probabilidad	2 - Improbable
Impacto	4 - Mayor
Daño	8 - Medio
Prevención	Estar informado acerca de posibles problemas en la infraestructura tecnológica para comprender los protocolos de mantenimiento, actualización y resolución de problemas del servidor.
Mitigación	Establecer un plan de contingencia donde se especifique que otras tareas puedo realizar si esto ocurre, como por ejemplo, el desarrollo de la memoria.

Tabla 2.7: Riesgo 5

Riesgo 6. Enfermedad o indisponibilidad

Riesgo 6	**Enfermedad o indisponibilidad del trabajador**
Descripción	Enfermedades o motivos personales durante un periodo largo de tiempo sería crítico al ser el único empleado responsable del proyecto y podría demorar notablemente los plazos de entrega y desarrollo.
Probabilidad	1 - Raro
Impacto	4 - Mayor
Daño	4 - Bajo
Prevención	Priorizar el cuidado personal y cuidar la salud para reducir el riesgo de enfermedades prolongadas.
Mitigación	Documentar claramente el proyecto para que si estoy inactivo un periodo pueda retomarlo sin excesivas dificultades. También, tener un horario flexible y poder acceder a la plataforma de la entidad bancaria desde casa ayuda a la mitigación este riesgo.

Tabla 2.8: Riesgo 6

Riesgo 7. Fin del contrato con la empresa antes de finalizar el TFG

Riesgo 7	**Fin del contrato con la empresa antes de finalizar el TFG**
Descripción	La finalización de mi contrato de prácticas con la empresa con la que realizo el proyecto sería un claro impedimento a la hora de seguir con el trabajo.
Probabilidad	1 - Raro
Impacto	5 - Grave
Daño	5 - Medio
Prevención	Estar al tanto de los periodos que abarca mi contrato y mantaner comunicación con la empresa respecto a este tema.
Mitigación	Tratar de establecer un nuevo acuerdo o ponerme en contacto con mi tutora de empresa.

Tabla 2.9: Riesgo 7

Riesgo 8. Falta de apoyo al proyecto

Riesgo 8	**Falta de apoyo al proyecto**
Descripción	Al ser en convenio con la empresa y tratarse de datos confidenciales, el tutor académico apenas puede entrar en temas prácticos y la tutora de empresa puede estar más ocupada y/o centradas en sus trabajos. Esto supondría más problemas a la hora de realizar el proyecto y podría demorarlo.
Probabilidad	1 - Raro
Impacto	4 - Mayor
Daño	4 - Bajo
Prevención	Establecer y mantener una comunicación proactiva para garantizar que se aborde cualquier problema de manera oportuna y efectiva.
Mitigación	Programar encuentros y compartir progresos con anticipación para facilitar la organización de todos.

Tabla 2.10: Riesgo 8

Riesgo 9. El resultado del proyecto no cumple los objetivos

Riesgo 9	**El resultado del proyecto no cumple los objetivos**
Descripción	El proyecto podría no cumplir con los objetivos establecidos en 1.3, lo que podría deberse a diversos factores. Esto podría resultar en una entrega insatisfactoria o en incumplimiento de plazos, afectando por completo al proyecto.
Probabilidad	1 - Raro
Impacto	5 - Grave/Severo
Daño	5 - Medio
Prevención	Antes de comenzar el proyecto, llevar a cabo una evaluación detallada de los recursos disponibles, las capacidades sobre las tecnologías a usar y los riesgos potenciales que podrían afectar el logro de los objetivos, además de programar reuniones regularmente con los tutores abordando estos temas.
Mitigación	Identificar claramente los objetivos, de forma que sean alcanzables y estableciendo hitos de seguimiento para evaluar el progreso y corregir desviaciones.

Tabla 2.11: Riesgo 9

Riesgo 10. Demoras inesperadas por parte de la entidad bancaria

Riesgo 10	**Demoras inesperadas por parte de la entidad bancaria**
Descripción	Retrasos en los plazos acordados con la entidad bancaria podrían repercutirme directamente y provocar retrasos en mi proyecto.
Probabilidad	2 - Improbable
Impacto	4 - Mayor
Daño	8 - Medio
Prevención	Obtener plazos realistas y claros con la entidad bancaria, teniendo en cuenta posibles retrasos y especificar claramente el problema que estos supondrían si fueran a más.
Mitigación	Mantener una flexibilidad en la planificación del proyecto para poder adaptarse a posibles cambios en los tiempos de entrega por parte de la entidad bancaria para que afecten lo mínimo posible.

Tabla 2.12: Riesgo 10

Riesgo 11. Requisitos mal entendidos o cambio en los requisitos

Riesgo 11	Requisitos mal entendidos o cambio en los requisitos
Descripción	Existe la posibilidad de malentender algunos de los requisitos establecidos por la entidad bancaria, lo que podría comprometer la duración final del proyecto.
Probabilidad	2 - Improbable
Impacto	3 - Significativo
Daño	6 - Medio
Prevención	Revisión exhaustivamente los requisitos con la entidad bancaria para aclarar cualquier ambigüedad y confirmar la comprensión mutua. Documentar todo lo relacionado a la comprensión de estos requisitos.
Mitigación	Establecer reuniones con la entidad bancaria y compañeros de la empresa para resolver ambigüedades o malentendidos.

Tabla 2.13: Riesgo 11

Riesgo 12. Pérdida del desarrollo del proyecto

Riesgo 12	Pérdida del desarrollo del proyecto
Descripción	La pérdida de código o documentación en desarrollo provocaría la repetición de trabajo, afectando a la duración del proyecto. La mayoría del proyecto se desarrollará en la nube por lo que en principio no debería ocurrir.
Probabilidad	1 - Raro
Impacto	4 - Mayor
Daño	4 - Bajo
Prevención	Guardar todo progreso logrado en la nube además de en local y dentro de la plataforma ofrecida por la entidad bancaria.
Mitigación	Realizar una documentación clara y plasmar avances en un cuadreno de bitácora para que en caso de que suceda, se tenga claro como volver a obtener lo perdido.

Tabla 2.14: Riesgo 12

La mayoría de los riesgos identificados se sitúan en el nivel de exposición "*Medio*", por lo que no deben ser subestimados y deben tratarse con atención. Algunos están clasificados como "*Bajo*" y no deben ser ignorados aunque, sobre todo, es crucial priorizar la atención en los riesgos identificados como "*Muy Alto*" (riesgo 3 en la tabla 2.5) y como "*Alto*" (riesgo 4 en la tabla 2.6).

Capítulo 3

Marco Teórico

A lo largo de este capítulo se exploran los conceptos teóricos que sustentan el proyecto, proporcionando un contexto crucial para comprender el desarrollo del mismo.

3.1. Big Data

Vivimos en la "Era de los Datos" o, lo que es lo mismo, en la era del *Big Data* [7]. Este término, *Big Data*, utilizado para describir conjuntos de datos grandes y complejos procedentes de diversas fuentes, se ha convertido en una parte muy significativa de nuestro mundo físico y digital. Con la continua expansión de los datos móviles, la inteligencia artificial, la computación en la nube, el aprendizaje automático y el IoT, la industria del *Big Data* está experimentando un crecimiento exponencial que solo va a más. Esta industria abarca diversos campos y aspectos interrelacionados, como centros de datos, servicios en la nube, dispositivos IoT, herramientas de aprendizaje automático y profundo, inteligencia artificial y herramientas de análisis predictivo [8].

3.1.1. Características y Tipos

Entrando más en profundidad, podemos definir el *Big Data* como una combinación de datos estructurados, semiestructurados y no estructurados que las organizaciones recopilan, analizan y extraen para obtener información y conocimientos. Estas grandes cantidades de datos se caracterizan por lo que es llamado como las tres V's:

- El gran **volumen** de datos en muchos entornos.
- La amplia **variedad** de tipos de datos que se almacenan en los sistemas de *Big Data*.
- La alta **velocidad** a la que se generan, recopilan y procesan los datos.

Más recientemente, se han ido agregando más otras V's a diferentes características del *Big Data*, como se muestra en la figura 3.1. En cuanto a la variedad, la estructura de los datos que lo componen se diferencian entre tres tipos [9] [10]:

- **Datos estructurados**, los cuales tienen un formato y un tamaño fijo y definido, como transacciones y registros financieros.
- **Datos no estructurados**, en su formato original ya que no tienen un formato específico, es imposible almacenar este tipo de datos en una tabla, ya que su información no se puede desglosar en los tipos de datos básicos de manera sencilla. Ejemplos de esto son texto, documentos y archivos multimedia.
- **Datos semiestructurados**, los cuales no siguen un formato fijo, pero incluyen marcadores que facilitan la separación de sus diferentes campos. Estos datos suelen tener metadatos propios que describen sus elementos y relaciones. Algunos ejemplos comunes son registros del servidor web, transmisión de datos desde sensores y archivos en formatos CSV, JSON, HTML o XML.

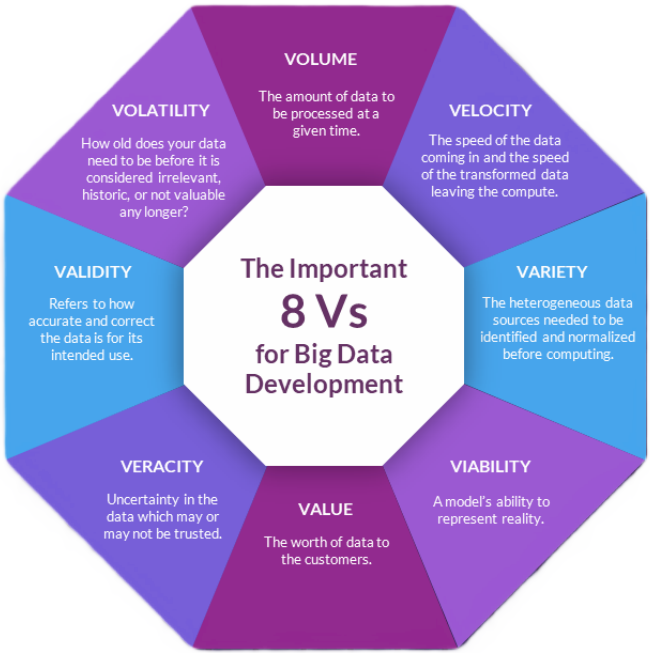

Figura 3.1: Características del Big Data [11]

Todo este crecimiento del *Big Data* viene desembocado por la gran cantidad de volumen de datos generados en todo el mundo cada segundo y es que más del 57 % de los datos del mundo son producidos por usuarios de Internet [8]. Para hacerse una idea, la tabla 3.1 muestra la evolución del volumen de datos generado a lo largo de los años, con una estimación para el presente 2024 y el año 2025. Cabe destacar que las unidad de la tabla es *Zettabyte (ZB)*, cuya equivalencia es [12]:

$$1\ ZB \approx 10^{12}\ GB \approx 10^{21}\ bytes \approx 2^{70}\ Bytes$$

Año	Volumen de Datos (ZB)
2025*	181 ZB
2024*	147 ZB
2023	120 ZB
2022	97 ZB
2021	79 ZB
2020	$64,2$ ZB
2019	41 ZB
2018	33 ZB
2017	26 ZB
2016	18 ZB
2015	$15,5$ ZB
2014	$12,5$ ZB
2013	9 ZB

Tabla 3.1: Evolución del volumen de datos creado [13]

3.1.2. Cloud Computing

Como se acaba de ver en la tabla 3.1, existe un claro crecimiento exponencial que ha llevado a la tecnología a buscar caminos y soluciones óptimas capaces de gestionar, almacenar y analizar estos datos. Y es aquí donde entra el término *Cloud Computing* o computación en la nube, que consiste en la prestación de servicios informáticos como servidores, almacenamiento, bases de datos, redes, software, análisis y muchos más a través de Internet, con el objetivo de proporcionar recursos de forma flexible, una innovación más rápida y economías de escala. La computación en la nube ha revolucionado la forma en la que se abstrae y se utiliza la infraestructura informática, empleándose para prácticamente cualquier cosa que pueda considerarse un servicio en la actualidad [14].

La nube aporta una gran variedad beneficios relevantes a empresas independientemente de su tamaño. Algunos de los beneficios más relevantes del *Big Data* en la nube son los siguientes [15]:

- **Escalabilidad**: la nube permite una expansión de recursos que, en principio, es ilimitada, superando las limitaciones físicas de los centros de datos tradicionales. Tiene capacidad para gestionar miles de servidores a través de múltiples ubicaciones globales, proporcionando un acceso rápido y flexible a una infraestructura de *Big Data* sin necesidad de grandes inversiones iniciales.

- **Agilidad**: la nube proporciona una gran flexibilidad que permite ajustar dinámicamente los recursos según las demandas específicas de cada proyecto. Esto permite que los recursos pueden ser seleccionados y liberados según sea necesario, optimizando la gestión de recursos y costos.

- **Presupuesto**: como se ha comentado al hablar de escalabilidad, la nube también es económica. Los métodos tradicionales, como la instalación de un centro de datos empresarial, implican enormes gastos de capital en *hardware*, instalaciones, energía y un mantenimiento continuo de todo. Sin embargo, en la nube todos esos gastos se combinan en un modelo flexible de pago por uso.

- **Resiliencia**: la propia nube por sí sola garantiza cierta seguridad y la total disponibilidad de los datos, replicándolos y distribuyéndolos eficientemente. Esto produce altos niveles de redundancia y disponibilidad, garantizando una alta disponibilidad en todos los recursos hasta en circunstancias adversas, como caídas o fallos del servicio.

Estos grandes beneficios se hacen mucho más fuertes que sus inconvenientes, como pueden ser la dependencia a una conexión veloz y sin interrupciones, el riesgo de una disminución en la seguridad, ya que el sistema se vuelve más expuesto y con ello más vulnerable a ataques, o el costo de almacenamiento a largo plazo para grandes proyectos. Cabe destacar también que la funcionalidad puede estar limitada cuando la nube esté gestionada por proveedores de servicios externos, implicando una pérdida de control sobre ciertos aspectos o incluso limitaciones en la funcionalidad.

3.1.3. Ciclo de Vida del Dato

Esta sección aborda el ciclo mediante el cual los datos se utilizan para extraer valor, información y conocimiento en cualquier ámbito o contexto, incluido el empresarial. Este ciclo está esquematizado por fases en la figura 3.2 y seguidamente detallado para un proyecto cualquiera de *Big Data*, como podría ser el de este Trabajo Fin de Grado [16] [17]:

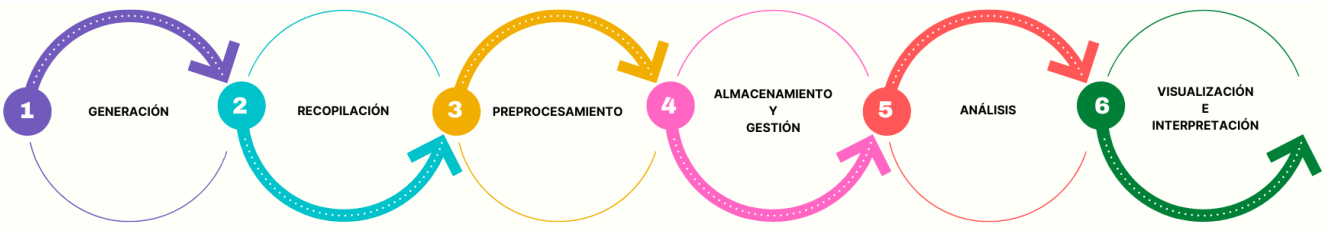

Figura 3.2: Ciclo de vida del dato

- Primero, lo que tiene que pasar para que comience el ciclo de vida del dato es la **generación** del propio dato. La mayoría de las veces, esto se produce de forma inconsciente: tanto los individuos como las empresas generan datos de forma constante con cada interacción en Internet, con cada compra, cada venta,... todo deja un rastro de datos. También pueden generarse datos de forma intencional, a través encuestas y formularios, entre otros métodos.

- Una vez generados, se debe hacer una selección o **recopilación** de los datos que generan interés. Para ello, es crítico que un equipo de especialistas realicen un **análisis interno** para identificar qué información se debe capturar y cuáles son los mejores medios para hacerlo. Este análisis debe basarse en la comprensión del negocio y una justificación de las necesidades, estableciendo una serie de KPI's para comprender los resultados del análisis y la capacidad para cumplir las metas y objetivos establecidos [18]. Se denomina KPI a una medida cuantitativa que se utiliza para evaluar el rendimiento y el progreso hacia los objetivos establecidos.

- Con los datos ya recopilados, se pasa al **preprocesamiento**, un proceso de **filtrado** de datos donde las claves son la limpieza (lidiar con datos faltantes, atípicos o inconsistentes), integración (unificar las diversas fuentes de datos), conversión (moldear en el formato o estructura deseada) y especificación (minimizar la cantidad de datos sin perder valor). Durante esta fase se puede conocer la calidad de los datos desde el punto de vista técnico (formatos, complejidad, disponibilidad, integridad de fuentes, etc.).

- La siguiente fase es la del **almacenamiento** de los datos previamente recopilados y procesados. Esta fase se refiera al uso de la memoria para almacenar los datos recopilados en bases de datos o de conjuntos de datos. Es importante establecer protocolos de seguridad y hacer copias de seguridad de todos los datos que se vayan a almacenar como medida preventiva en caso de que la fuente original se corrompa o se vea comprometida. Lógicamente, estos datos almacenados deben que tener una **gestión** continua, en la que se organizan, almacenan y recuperan datos según sea necesario durante todo el ciclo de un proyecto.

- Una vez en este punto, llega el momento de **analizar** los datos según los objetivos y metas fijados, con el objetivo final de proporcionar una visión unificada de la información que aporte algún tipo de valor. Esta fase, estrechamente relacionada con la ciencia de datos, emplea técnicas estadísticas, como modelado estadístico, algoritmos, inteligencia artificial, minería de datos y aprendizaje automático y profundo para predecir, detectar patrones o anomalías y optimizar millones de problemas que antes ni siquiera podían ser imaginados.

- Con los datos ya analizados, la mejor forma de trasmitir los resultados obtenidos de manera óptima, clara y valiosa es a través de la **visualización**. Esta fase se refiere al proceso de creación de representaciones gráficas de información procedente de los análisis, facilitando la comunicación rápida de su resultados a una audiencia más amplia.

- Por último, se debe **interpretar** la información obtenida tras todos estos pasos para realmente obtener valor de todo esto. Esta fase suele desembocar en decisiones, planes y estrategias empresariales que acerquen a una empresa a sus metas y objetivos.

3.2. Ingestas y Procesos ETL

Después de entrar en contexto con el *Big Data*, definir detalladamente su ciclo de vida y exponer su potencial, beneficios e importancia en la actualidad y en el futuro, esta sección se centra en el tema principal de este proyecto: las ingestas de datos y los procesos ETL.

3.2.1. Ingesta de Datos

Una **ingesta de datos** se produce en la fase de integración, dentro del preporcesamiento, según el ciclo de vida del dato de la figura 3.2. Además, la ingesta de datos es la primera capa de la arquitectura de *Big Data*, siendo responsable de recopilar datos de diversas fuentes de datos para depositarlos en un sistema de almacenamiento de datos de destino, como puede ser un *Data Warehouse* o, como en nuestro caso, en un *Data Lake* (se trata este tema en la sección 3.2.3). En esta etapa se puede comprender el tamaño y la complejidad de los datos, lo que afectará la arquitectura o a cada decisión tomada en el futuro. La etapa de ingestar datos tiene sus desafíos, entre los que destacan la complejidad debido a la velocidad y variedad de los datos, además de que el desarrollo puede ser costosos en tiempo y recursos. Otro desafío es la seguridad, ya que al transferir datos de un lugar a otro, existe un riesgo de vulnerabilidad para datos confidenciales. También destacar que durante el proceso, la confiabilidad de los datos puede verse comprometida haciendo que los datos no tengan valor o, en el peor de los casos, que propicien decisiones incorrectas basadas en datos falsos [19]. Estas ingestas presentan dos variantes en su realización:

- **Ingestas de datos en tiempo real** o ***streaming***: se basa en recopilar y procesar datos de diversas fuentes en tiempo real. También es conocido como transmisión por secuencias y se emplea este enfoque cuando los datos recopilados son urgentes, ya que la velocidad de los datos será alta y los datos se extraerán, procesarán y guardarán lo más rápido posible.

- **Ingestas de datos por lotes** o ***batch***: donde los datos se mueven desde las fuentes de datos al destino en intervalos programados, periódicamente. Este enfoque es útil cuando las empresas necesitan recopilar grandes cantidades de datos en periodos establecidos y regulares (una vez al mes, una vez a la semana, etc.).

3.2.2. Proceso ETL

El objetivo al realizar una ingesta es crear una única canalización de datos, conocida como *pipeline*, para mover datos desde las diversas fuentes de origen hasta el sistema destino elegido. Por tanto, entendemos como *pipeline* de datos un proceso que consume datos desde un punto de origen, los limpia y transforma según las necesidades y los escribe en un nuevo destino.

El **proceso ETL** (*Extract-Transform-Load*) es un tipo de *pipeline* de integración de datos que requiere tres pasos distintos pero interrelacionados (extraer, transformar y cargar), como se muestra en la figura 3.3. Estos tres pasos confrontan entonces tres diferenciadas fases del proceso ETL que frecuentemente se ejecutan en paralelo [20].

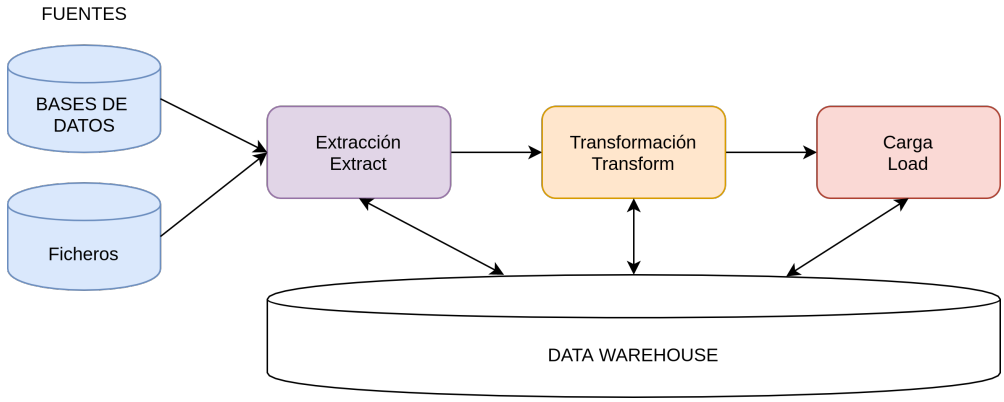

Figura 3.3: Proceso ETL [21]

Extracción

En el primer paso del proceso, los datos se extraen de diversas fuentes y formatos, recopilándose en el área de preparación antes de ser enviados al almacén de datos. Este área de preparación es una plataforma donde se clasifican los datos antes de enviarlos directamente al almacén de datos, verificando su integridad y formato. Es crucial evitar daños que puedan afectar la estructura del almacén y garantizar que la extracción de datos tenga un impacto mínimo en el sistema de origen. La extracción de datos puede realizarse de tres maneras distintas [22]:

- **Extracción completa**: en este método, los datos de origen se proporcionar tal como están y no se necesita información lógica adicional en el sistema de origen. No es necesario realizar un seguimiento del sistema origen para detectar cambios. Este método es muy útil cuando la cantidad de datos es manejable y no muy grande o cuando se está realizando la extracción inicial de datos al comienzo de un proyecto.

- **Modificación de actualización** o **basada en cambios**: este método, gracias a mecanismos y herramientas en bases de datos ETL, identifica y extrae únicamente los cambios que se

han producido en la fuente de datos desde la última extracción. Esto se logra mediante notificaciones del sistema sobre modificaciones en registros, lo que facilita la replicación de datos y ayuda a eliminar redundancias.

- **Extracción incremental**: solo se extraen los datos que han sido modificados o añadidos desde la última extracción, reduciendo el tiempo y los recursos necesarios, ya que solo se obtienen los datos nuevos o actualizados desde la última vez.

Transformación

La fase de transformación de un proceso ETL es la parte fundamental en la que se basa el trabajo de este proyecto. En esta fase se aplican una serie de reglas de negocio o funciones (en esta memoria denominado el conjunto de ellas como *lógica*) sobre los datos extraídos para convertirlos en datos que serán cargados, con el formato y contenido esperado. Esto incluye aplicar operaciones sobre los datos, tal como muestra el ejemplo de la figura 3.4, como pueden ser cambios de codificación, eliminación de datos duplicados, cruces de diferentes fuentes, agregaciones, transformaciones, cambios en la estructura o muchas otras operaciones que permitan acercarse al formato y contenido buscado. Estas operaciones deben transformar los datos para mejorarlos, incrementar su calidad, integrarlos con otros sistemas, normalizarlos, eliminar duplicidades o ambigüedades y no deben duplicar ni eliminar información relevante o errónea. A menudo, este proceso implica el uso de tablas de almacenamiento provisional para conservar los datos temporalmente a medida que estos se transforman.

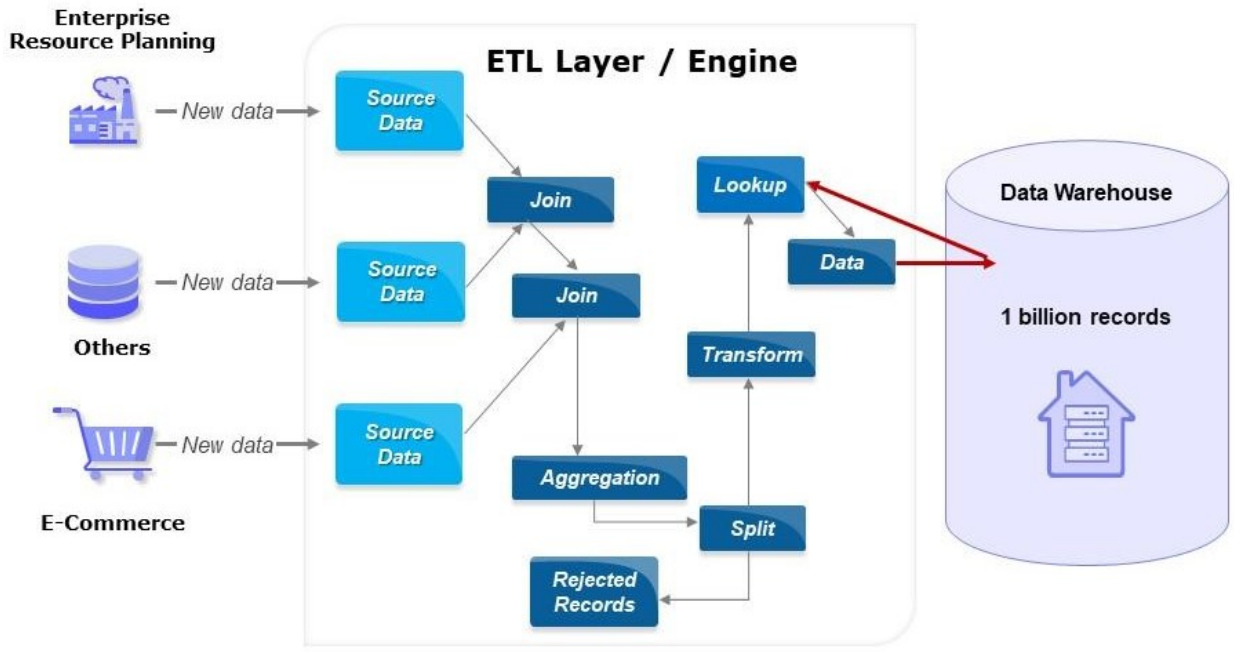

Figura 3.4: Ejemplo de la fase de transformación de un proceso ETL [23]

Carga

Una vez transformados, los datos ya estarán listos para su carga. Los datos procedentes de la fase anterior son cargados en el sistema de destino, normalmente un *Data Warehouse* o un *Data Lake*. Dependiendo de los requerimientos de la organización, este proceso puede abarcar una amplia variedad de acciones diferentes, pero normalmente los datos que se cargarán suelen ser en cantidades enormes y deben manejarse con una estrategia de carga previamente planificada. Por lo tanto, este proceso debe optimizarse completamente para llevarlo a cabo de manera óptima y eficiente. Se diferencian dos métodos básicos para efectuar una carga [22]:

- **Carga completa**: este tipo de carga implica cargar todos los datos desde la fuente de origen hasta el sistema de destino del proceso ETL. Es útil cuando se requiere una actualización completa de los datos en el almacén, como al inicio del proyecto o cuando se necesitan recargas periódicas para mantener la integridad de los datos. Este es el método más sencillo de carga y, ya que requiere bastante tiempo y recursos, pero no es el mejor método para implementarlo de forma predeterminada.

- **Carga incremental**: este enfoque solo carga los datos que han sido modificados o añadidos desde la última ejecución. Esto reduce el tiempo y los recursos necesarios para la carga de datos, ya que solo se actualizan las partes nuevas de la base de datos. Es útil cuando la cantidad de datos es demasiado grande para cargarlos de una sola vez o cuando hay mucha diferencia de tamaño entre los registros actualizados respecto al conjunto total de datos.

Resumiendo, el proceso ETL es una canalización de datos que se usa para recopilar datos de varios orígenes, transformándolos según una lógica de negocio y cargándolos en un almacén de datos de destino. Existe también otra alternativa, una ligera variante del proceso ETL que es el proceso ELT [22]. El proceso ELT difiere de ETL solo por la ubicación en la que se realiza la transformación. En un proceso ELT los datos se extraen y se guardan en el área de preparación desde donde se cargan directamente al sistema destino, donde una vez ahí se realiza la transformación de los datos. Por lo tanto, en la canalización de ELT, la transformación se produce en el almacén de datos de destino, simplificando la arquitectura del sistema. Otra ventaja de este enfoque es que al escalar el almacén de datos de destino también se escala el rendimiento de la canalización de ELT. No obstante, ELT solo funciona bien si el sistema de destino tiene la suficiente potencia para transformar los datos de forma eficaz [24].

Otra de las claves que aborda este proyecto es la **automatización**. La automatización es una de las claves en la mejora de rendimiento de este tipo de procesos ETL, puesto que implica minimizar el papel de los trabajadores humanos y depender únicamente de herramientas para limpiar datos, moverlos a través del proceso ETL y verificar los resultados de forma eficiente y rápida, ahorrando tiempo y recursos que pueden ir destinados a otras tareas más productivas [26].

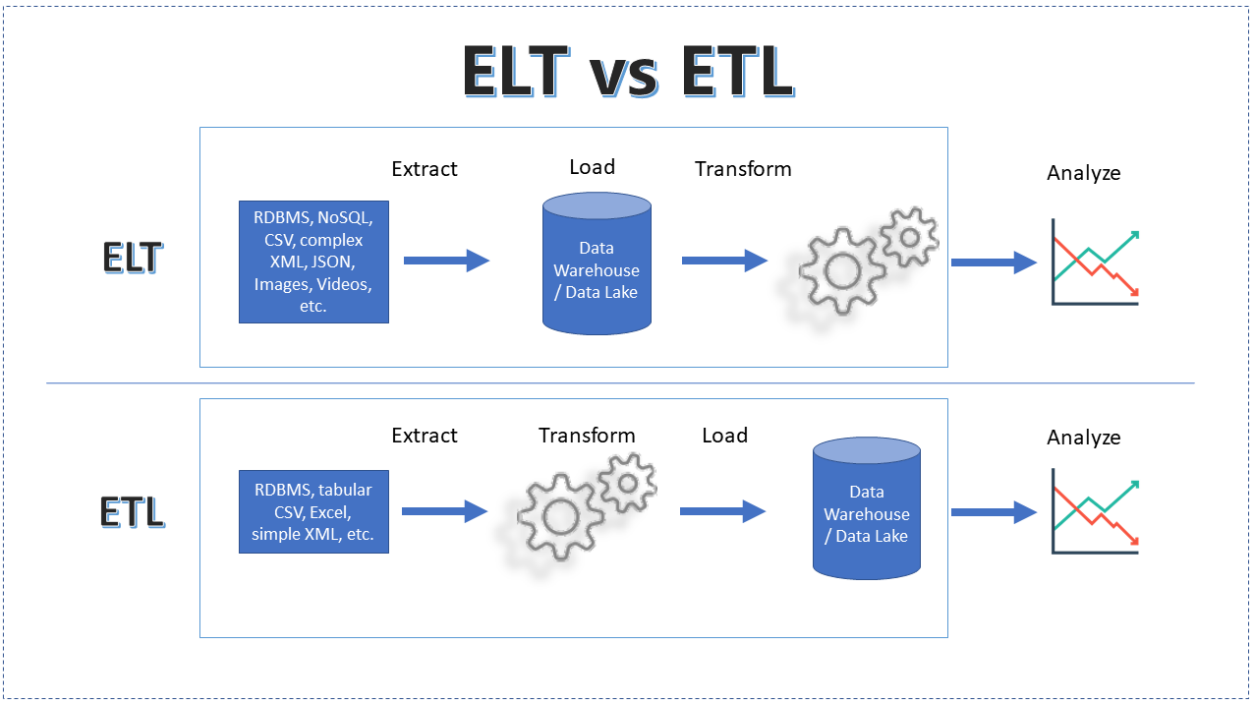

Figura 3.5: Proceso ELT vs Proceso ETL [25]

3.2.3. Data Warehouse vs Data Lake

Anteriormente se ha comentado cómo el fin del proceso ETL tiene lugar con la carga de los datos con el aspecto y las características deseadas en un sistema final de almacén de datos. Este sistema final habitualmente suele ser un *Data Warehouse* o un *Data Lake*, pero ¿qué diferencia hay entre ellos? Si bien tanto los unos como otros se utilizan para almacenar datos, tienen diferentes propósitos y tienen características únicas, como se muestra a continuación [27].

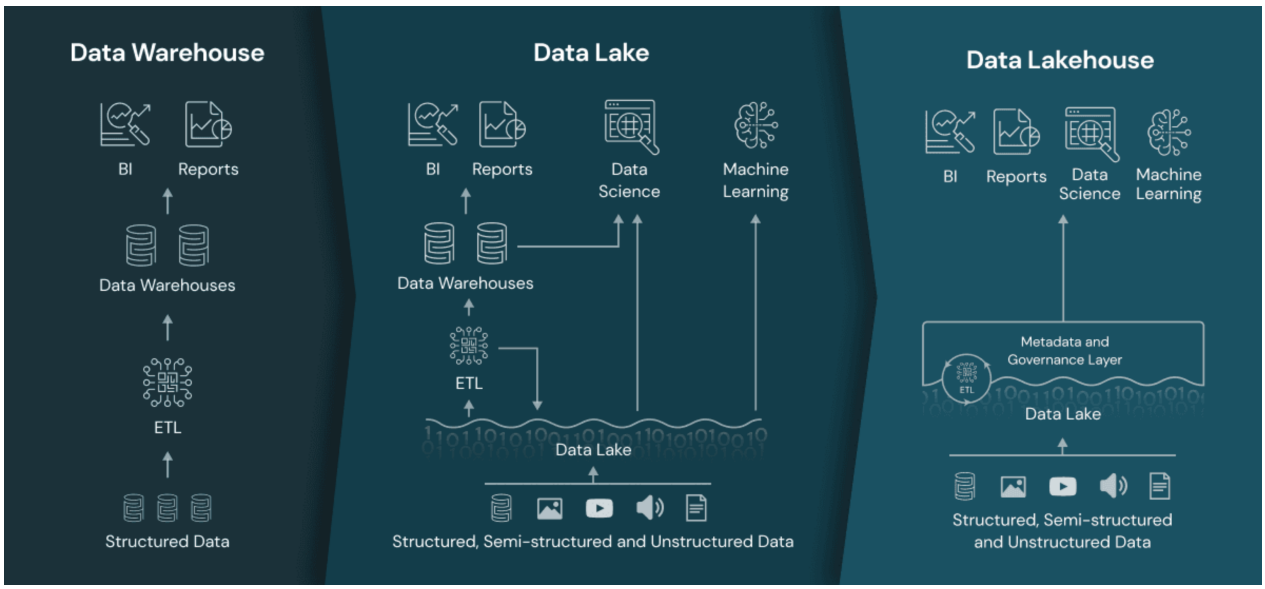

Figura 3.6: Data Warehouse vs Data Lake vs Data Lakehouse [28]

Data Warehouse

Un *Data Warehouse* o almacén de datos es un gran depósito que únicamente permite datos estructurados, es decir, datos filtrados que ya han sido procesados. Los almacenes de datos funcionan mejor cuando responden preguntas específicas y los datos de un *Data Warehouse* están ya procesados, limpiados y organizados, lo que los hace disponibles para crear informes y paneles para responder consultas de forma más directa y rápida.

Data Lake

Un *Data Lake* o lago de datos es un gran depósito de almacenamiento que contiene una cantidad significativa de datos sin procesar, en su formato original, hasta que se necesitan. Un lago de datos utiliza una arquitectura plana para almacenar datos, donde cada elemento recibe un identificador único y se etiqueta con metadatos extendidos. Cuando surge una pregunta de interés, se puede consultar el lago de datos para encontrar los datos relevantes y analizar ese conjunto más pequeño de datos para obtener respuestas. Debido a que los lagos de datos son útiles para el almacenamiento y análisis de datos estructurados, semiestructurados y no estructurados, un *Data Lake* puede almacenar todo tipo de datos, aportando gran flexibilidad, independientemente de su tipología, formato o procedencia, ya que no requiere normalizar su estructura.

Otras ventajas que presentan es su bajo coste y que facilitan análisis avanzados como el aprendizaje automático y el análisis predictivo y ofrecen versatilidad y escalabilidad debido a su capacidad para acomodar una amplia variedad de tipos de datos.

Resumiendo, los lagos de datos permiten que las organizaciones usen almacenamiento de bajo costo para almacenar grandes cantidades de datos sin procesar, proporcionando funciones de estructura y administración de datos.

Data Lakehouse

Un *Data Lakehouse* es un sistema que combina los beneficios de un *Data Lake* (repositorio de datos sin procesar en su forma original y de cualquier estructura) y un *Data Warehouse* (conjuntos organizados de datos estructurados). Este enfoque presenta múltiples beneficios, como una arquitectura simplificada y mejor administración, mejorando la calidad de los datos y evitando duplicaciones. También ofrece costes más bajos, mayor confiabilidad y una alta escalabilidad [29].

Concluyendo, un *Data Lakehouse* fusiona la flexibilidad y economía de un *Data Lake* con la capacidad de consulta contextual y alta velocidad de un *Data Warehouse*. Esto implica que las empresas puedan usar un solo sistema final para guardar todos sus datos sin perder la capacidad de analizarlos como deseen, permitiendo una gran eficiencia en trabajos de análisis y de aprendizaje

automático.

En resumen, la elección entre *Data Lake*, *Data Warehouse* y *Data Lakehouse* depende del caso de uso específico, la naturaleza y el volumen de los datos y los objetivos de una organización. En el caso del proyecto asociado a este TFG, el proceso ETL finaliza con la carga de datos en un *Data Lake*.

Capítulo 4

Marco Tecnológico

Este capítulo aborda una visión general de las distintas opciones tecnológicas presentes en el mercado para tratar los conceptos comentados en el capítulo 3.

4.1. Tecnologías Big Data

Cuando se menciona la tecnología *Big Data* se habla de una tecnología capaz de analizar un gran volumen de datos o conjuntos de ellos a una velocidad muy elevada, obteniendo los resultados buscados [30]. Para ello, existen múltiples herramientas *software* que han ido surgiendo a lo largo de los últimos años [31], entre las que destaca Apache Hadoop.

Apache Hadoop es un marco de código abierto que permite el procesamiento distribuido de grandes conjuntos de datos en grupos de servidores básicos. Es una de las tecnologías más populares para el almacenamiento de los datos estructurados, semiestructurados y no estructurados que forman *Big Data*. Entre sus características principales, destacan [32]:

- **Confiabilidad**: se crean múltiples copias de los datos de manera automática para que en caso de fallo se vuelva a desplegar la lógica de procesamiento.
- **Tolerancia a fallos**: tras detectar un fallo se aplica una recuperación automática. Cuando un componente se recupera, vuelve a formar parte del *cluster*. Los fallos se tratan como una regla, no como una excepción.
- **Escalabilidad**: los datos y su procesamiento se distribuyen sobre un *cluster* de ordenadores, cada uno de ellos ofreciendo computación y almacenamiento local.
- **Portabilidad**: se puede instalar en todo tipos de *hardware* y sistemas operativos.

Apache Hadoop cuenta con un conjunto de utilidades comunes (*Hadoop Common*) y proporciona un sistema de archivos distribuidos (*HDFS*) diseñado para brindar escalabilidad y confiabilidad. Este sistema de archivos se corresponde con la capa de almacenamiento, y como tal es un sistema de ficheros distribuido y tolerante a fallos que puede almacenar gran cantidad de datos, escalar de forma incremental y sobrevivir a fallos de *hardware* sin tener pérdidas de datos.

Hadoop también implementa un paradigma llamado *MapReduce* empleado para paralelizar las tarea, donde las ejecuciones se dividen en pequeños fragmentos de trabajo de forma que cada uno de ellos se pueda ejecutar en cualquier nodo del *cluster*.

Además, Hadoop proporciona un administrador de recursos (*YARN*) que permite una programación eficiente de la ejecución de trabajos. Este adminisitrador es un distribuidor de datos y gestor de recursos distribuidos que abstrae la gestión de recursos de los procesos *MapReduce*, resultando en una asignación de recursos más efectiva.

Todos estos elementos hacen que Hadoop simule un sistema de fichero locales en nuestro ordenador personal, pero realmente los datos están repartidos entre miles de servidores. Existen múltiples tecnologías para utilizar estas herramientas, facilitando el acceso, gestión y extensión del propio Hadoop. Entre ellas destaca **Apache Spark**, que es un *framework* de computación distribuida que utiliza un conjunto de operaciones (transformaciones y acciones) para procesar grandes volúmenes de datos. Por tanto, Apache Spark es un motor muy eficiente de procesamiento de datos a gran escala que implementa procesamiento en tiempo real. Destaca también su capacidad de trabajar en *clusters* completos con paralelismo de datos, asegurando a la vez la tolerancia a fallos. Para ello, en vez de almacenar los datos en disco, trabaja de forma masiva en memoria, provocando que los procesos requieran un menor tiempo. Spark puede trabajar de forma autónoma, sin necesidad de Hadoop, aunque Spark no cuenta con un sistema de archivos propio y a menudo se utilizan juntos para mejorar el rendimiento y emplear el sistema de archivos distribuido *HDFS*. En este proyecto se empleará una interfaz para Apache Spark en Python, llamada *PySpark*, para realizar análisis de datos exploratorios a escala, construir algoritmos de machine learning y crear ETL's para una plataforma de datos, además de Hadoop para, a través de *HDFS*, interactuar con el almacenamiento de datos desde una terminal.

Para finalizar, hoy en día Hadoop Hadoop ofrece ventajas significativas sobre otras opciones, como las bases de datos SQL. Estas últimas están diseñadas para datos estructurados en tablas relacionales con una estructura definida, mientras que Hadoop está especialmente diseñado para manejar todo tipo de información, convirtiéndolo en una herramienta poderosa para la gestión y análisis de datos a gran escala.

4.2. Tecnologías Cloud

En la actualidad, múltiples proveedores ofrecen plataformas de servicios en la nube para la gestión, almacenamiento y cómputo del *Big Data*, tanto de forma pública como de forma privada. La **nube pública** se caracteriza en que los recursos se comparten de forma pública y entre varios clientes a la vez, permitiendo el acceso via internet. Estas nubes son administradas por proveedores que, a través de una red pública, ofrecen acceso rápido a recursos informáticos asequibles. En la nube pública destacan los siguientes provedores [33]:

- **Amazon Web Services (AWS)**: la nube de Amazon fue el primer proveedor *cloud* y es líder en el mercado de servicios en la nube, ofreciendo una amplia gama de servicios de cómputo, almacenamiento, bases de datos, IoT, seguridad e inteligencia artifical entre otros. Cuenta con la mayor presencia global y la mayor cantidad de regiones y zonas de disponibilidad, destacando por su amplitud y variedad en herramientas y soportes para una gran variedad de plataformas y tecnologías. Ofrece opciones de precios variadas, desde pago por uso hasta compromisos a largo plazo.
- **Azure**: la plataforma *cloud* de Microsoft se caracteriza por su estrecha integración con herramientas como *Azure Active Directory* y *Azure DevOps*, así como su enfoque en soluciones híbridas con *Azure Stack*. Cuenta con una amplia gama de servicios, ofreciendo modelos de pago por uso o compromisos a largo plazo. Cuenta con una sólida presencia global, con numerosas regiones y centros de datos disponibles por todo el mundo.
- **Google Cloud**: la nube proporcionada por Goolge destaca por su fortaleza en análisis de datos, inteligencia artificial y *machine learning*, con herramientas como *BigQuery* y *TensorFlow*. Ofrece una presencia global similar a Azure, con múltiples regiones y zonas de disponibilidad. También proporciona opciones de precios flexibles, con modelos de pago por uso y compromisos a largo plazo.

Por otro lado, en la **nube privada** los recursos ofrecidos están dedicados exclusivamente a una sola organización y se accede a través de un *cluster* dedicado, típicamente mediante una conexión privada como una red de fibra propia o una VPN. Este tipo de nube privada es utilizada exclusivamente por la organización, ya sea administrada internamente o por terceros. Hablando de nube privada, destaca **OpenStack**, una plataforma de computación en la nube de código abierto que permite crear y administrar tanto nubes públicas como privadas, con una gran variedad de servicios como cómputo, almacenamiento y redes. Es utilizado por empresas y proveedores de servicios para construir infraestructuras en la nube flexibles y escalables, siendo uno de los proyectos *open source* más activos del mundo.

En el caso particular de este proyecto se emplea Amazon Web Services, ya que es la tecnología usada por la entidad bancaria.

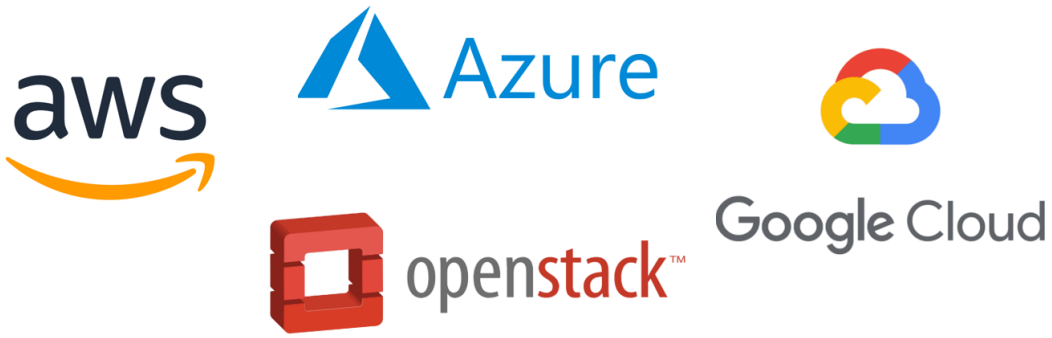

Figura 4.1: Principales tecnologías cloud

4.3. Tecologías ETL

Las herramientas empleadas para realizar procesos ETL deben cumplir una serie de requerimientos, como permitir conectividad con diferentes sistemas y tipos de datos, tener capacidad para transformar los datos (desde tranformaciones simples hasta tranformaciones complejas), permitir tener información del funcionamiento de todo el proceso y tener el control de errores para establecer acciones a realizar cuando estos se producen. Las herramientas más empleadas al realizar procesos ETL son [32]:

- **Pentaho Data Integration (PDI)**: también conocido como **Kettle**, es una de las herramientas de código abierto proporcionadas por *Pentaho Suite* y permite emplear técnicas ETL para poder implementar procesos de extracción, transformación y carga de datos. Esta herramienta cuenta con una interfaz gráfica de usuario llamada *Spoon*, la cual permite diseñar flujos de trabajo ETL y capacidades avanzadas para integración con *Big Data*. Esto permite diseñar el proceso ETL de una forma muy clara y sencilla, realizando la propia documentación al mismo tiempo que la implementación. En el ejemplo de la figura 4.1 se muestra un simple ejemplo en el que, a partir de un CSV y de un JSON, se selecciona una serie de campos o variables (columnas), se ordenan sus observaciones (filas) y se realiza una unión o *join* de ambos conjuntos.

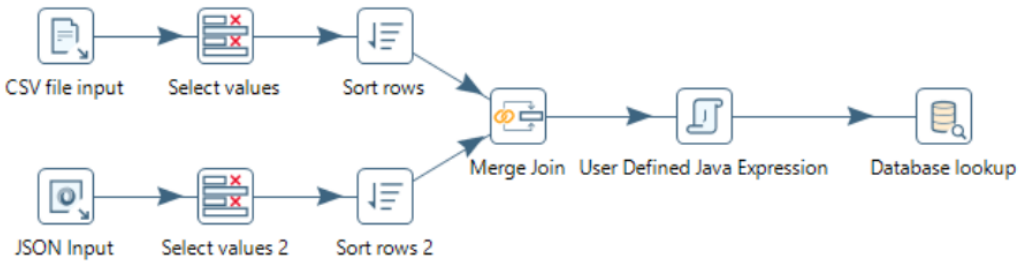

Figura 4.2: Ejemplo simple de flujo ETL en Spoon, Pentaho [34]

- **Oracle Data Integrator (ODI)**: ofrece otros métodos variados de integración además de ETL, siempre con alto rendimiento y capacidades en tiempo real. Garantiza la calidad de los

datos y su gobierno y es compatible con múltiples tecnologías. Además, también proporciona una interfaz para el diseño del proceso, igual que hacía PDI.

- **Talend Open Studio (TOS)**: es una *suite* de código libre que aporta un conjunto muy complejo, variado y completo de herramientas para llevar a cabo la integración de datos [35].
- **SQL Server Integration Services (SSIS)**: esta plataforma de *Microsoft* permite para la creación de soluciones de transformaciones e integración de datos. Permite extraer y transformar datos de diversos orígenes y después cargar los datos en uno o varios destinos, empleando herramientas gráficas para crear soluciones sin escribir código, aunque también permite crear paquetes mediante programación y codificar tareas personalizadas [36].
- **MuleSoft**: es conocida principalmente como una plataforma de integración más que como una herramienta ETL específica. Sin embargo, se puede utilizar para tareas de procesos ETL dentro del contexto de integración, ya que permite construir integraciones ETL con su propio lenguaje de transformación de datos denominado *DataWeave*. Se trata de un lenguaje funcional que permite transformar mensajes (JSON, XML, CSV) ejecutando operaciones sobre *arrays*, sobre *strings*, fechas y demás [37] [38].
- **Informatica Data Integration**: es una plataforma que incluye la herramienta de integración de datos *PowerCenter*, la cual permite diseñar, desarrollar y ejecutar procesos de integración de datos de forma visual. Cuenta con una interfaz gráfica intuitiva de *drag & drop* (arrastrar y soltar) y ofrece una amplia conectividad y compatibilidad con muchas fuentes de datos. Destaca por su escalabilidad y rendimiento, siendo capaz de gestionar grandes volúmenes de datos y procesos complejos [39].
- **IBM InfoSphere DataStage**: es una herramienta de integración de datos que permite mover y transformar datos de forma precisa y eficiente entre diferentes sistemas. Está diseñada para crear flujos de datos confiables, proporcionando flexibilidad y soportando tanto procesos ETL como ELT. También cuenta con una interfaz visual.

Figura 4.3: Principales herramientas ETL [32]

4.4. Herramientas del Proyecto

Una vez comentada la visión general de las opciones y tecnologías empleadas en este ámbito, en esta sección se van a describir las herramientas utilizadas durante el proyecto. La mayoría de estas tecnologías vienen dictaminadas por la entidad bancaria, ya que son las que ofrecen y deben usarse para el desarrollo de la solución.

Hadoop

Se emplea **Hadoop**, explicada detalladamente en la sección 4.1, usando su sistema de archivos distribuidos HDFS para interactuar con el almacenamiento de datos desde una terminal, con los comandos de la forma *hdfs dfs -comandoLinux*. Estos comandos admiten otros parámetros con diferentes opciones, como se puede ver en 4.1.

```
hdfs dfs -ls
hdfs dfs -get archivo ruta_destino_local
hdfs dfs -put archivo_local ruta_destino
hdfs dfs -rm archivo
```

Listing 4.1: Ejemplo de uso de comandos con HDFS

Figura 4.4: Logo de Hadoop

PySpark

PySpark es la API de Python para Apache Spark, como se comenta en la sección 4.1. **PySpark** permite realizar el procesamiento de datos a gran escala en tiempo real en un entorno distribuido utilizando Python, ofreciendo un *shell* PySpark para analizar sus datos de forma interactiva. Esto permite combinar la capacidad de aprendizaje y la facilidad de uso de Python con el poder de Apache Spark para permitir el procesamiento y análisis de datos de cualquier tamaño para todos los que estén familiarizados con Python. Esta tecnología permite diseñar y desarrollar a lógica a llevar a cabo para la fase de transformación.

Para usar PySpark se emplea Jupyter Notebook, como se detalla a continuación, con un *kernel* de spark-python en un servidor en la nube proporcionado por la entidad bancaria.

Por lo tanto, la herramienta fundamental en la que se basa el desarrollo principal del proyecto

Figura 4.5: Logo de PySpark

es **PySpark**, puesto que será la herramienta con la cual se realizará el análisis y diseño de la lógica relativa a la fase de transformación. Para una mayor familiarización con esta herramienta, se ha facilitado una guía básica introductoria al uso de esta tecnología en los anexos, en el apéndice B.

Jupyter Notebook

El Proyecto Jupyter es una organización sin fines de lucro dedicada al desarrollo de software de código abierto y estándares abiertos para la computación interactiva en múltiples lenguajes de programación, ofreciendo soporte para entornos de ejecución en diversos lenguajes. Su nombre proviene de los lenguajes principales que respalda: Julia, Python y R. Jupyter ha creado productos como **Jupyter Notebook**, JupyterHub y JupyterLab.

Durante este proyecto se emplea Jupyter Notebook, un entorno de desarrollo interactivo estructurado en celdas, de modo que permite ejecutar secciones de código de manera independiente, ahorrando tiempo y evitando errores [40].

Figura 4.6: Logo de Jupyter

Sandbox

Una herramienta **Sandbox** es un *software* que crea un entorno seguro, aislado y controlado en un sistema informático, a través de un espacio de almacenamiento con capacidad de procesamiento al que únicamente tienen acceso las personas del ámbito que se determine. En este entorno se pueden ejecutar aplicaciones o procesos de forma segura sin afectar el resto del sistema.

Por tanto, la idea se basa en proporcionar un entorno donde los desarrolladores puedan ex-

perimentar, probar y desarrollar código sin afectar al entorno de producción, mediante barreras que impiden que los cambios realizados dentro del *sandbox* se propaguen fuera de él. Para este proyecto, se realiza toda la implementación del código necesario para obtener la lógica de la transformación en un *sandbox*. Por temas de confidencialidad, no se puede detallar más el caso específico de la tecnología *Sandbox* empleada.

Microsoft To Do

Para seguir la **planificación** y organización del proyecto, se ha optado por una opción muy sencilla y simple como es **Microsoft To Do**. Esta aplicación basada en la nube se centra en organizar tareas, permitiendo a los usuarios gestionar sus tareas desde cualquier lugar. La decisión de emplear este *software* se basa en que el proyecto no cuenta con una alta complejidad, debido a la duración de este está estimada en 300 horas repartidas en tres meses, y tiene tareas muy diferenciadas y específicas. Además, es un proyecto al que solo me voy a dedicar yo, por lo que no requiere ninguna coordinación con ningún equipo en cuestión de tiempos, tareas o recursos, si no que tiene una gran flexibilidad en este aspecto.

Figura 4.7: Logo de Microsoft To Do

Overleaf

Para la realización de la **memoria** se ha empleado **Overleaf**, un editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos.

Figura 4.8: Logo de Overleaf

Capítulo 5

Planteamiento del Problema

En este capítulo se plantea el problema específico al que se enfrenta este proyecto, dando todo el contexto y especificaciones posibles, siempre evitando las cuestiones confidenciales y sustituyendo palabras críticas en este tema por otras más generales. También se realiza la recogida y análisis de requisitos.

5.1. Contexto Global del Problema

Para hablar del contexto del problema se debe recordar que este TFG se realiza en convenio con la empresa consultora *NTT DATA Spain, S.L.U.*, enfocado a un proyecto con un cliente, una entidad bancaria potente a nivel internacional. El proyecto tiene relación con la sostenibilidad en CIB (siglas en inglés de Banca Corporativa y de Inversión) y, por ello, se tratan datos de contratos y acuerdos con diferentes países de todo el mundo relacionados con inversiones y proyectos sostenibles. El objetivo del proyecto que involucra este trabajo, para ambas empresas, es realizar ingestas de datos y, a través de procesos ETL, obtener datos de millones de operaciones en un formato y en unas tablas específicas para poder analizar y obtener información valiosa a partir de ellas.

En concreto, el problema al que se enfrenta este TFG es la tranformación de un proceso ETL con periocidad mensual, por lo que, a partir de datos de miles de tablas de origen se debe obtener una tabla de destino que cumpla con los requisitos expuestos por el cliente. La periocidad mensual implica que la ingesta de datos provocada por el proceso ETL se realiza una vez al mes, una vez que los datos del final del mes anterior al mes en curso están cargados en la plataforma del banco.

Para realizar todo esto, se debe realizar un análisis de requisitos, un diseño, análisis, desarrollo e implementación de la lógica de transformación y una validación final, asegurando que todo funciona tal y como se desea.

5.2. Contexto Tecnológico del Problema

Como ya se ha comentado anteriormente, el proyecto tiene lugar en una plataforma proporcionada por la entidad bancaria, a través de la cual se proporciona las distintas tecnologías y herramientas comentadas en 4.4. Esta plataforma cuenta con tres entornos diferenciados:

- **Local**: entorno usado para el desarrollo del *software*, ficheros de configuración, pruebas, ficheros de entrada de prueba e ingestas sobre el sistema de ficheros local. En este entorno se desarrolla todo y se prueba antes de pasar al siguiente entorno, simulando sobre el sistema de ficheros local del ordenador las ingestas que posteriormente se llevarán al *HDFS*.
- **Work**: este entorno permite probar el *software* desarrollado sobre un entorno que simula al de producción, permitiendo entre otras cosas realizar las pruebas de ingesta y validación sobre una simulación de datos reales. Emplea *HDFS* como sistema de almacenamiento distribuido.
- **Live**: este entorno, también llamado **entorno de producción**, permite el uso en producción, sobre datos reales, con procesos automatizados y ejecuciones a través de peticiones. Cuando se ha validado y probado el *software* desarrollado en *local* en el entorno *work*, este se promociona a este último entorno. También emplea *HDFS* como sistema de almacenamiento distribuido.

Además, esta plataforma está estructurada en cuatro capas independientes:

- **Staging Inicial**: contien ficheros que pueden contener información almacenada en CSV, JSON, XML, TXT, etc.
- **Capa Raw**: en esta capa se almacenan los datos de la capa de *staging incial* en bruto. Estos son datos que, en teoría, no se procesan para su uso, salvo pequeños cambios de formato requeridos por la plataforma. Se guardan en HDFS en formato AVRO, aceptando únicamente los tipos de datos *String* o *Integer*.
- **Capa Master**: en esta capa se almacenan los datos procesados de la capa *raw*. Se guardan en *HDFS* en formato PARQUET, basado en columnas y altamente eficiente para datos estructurados. Los tipos de datos en esta capa son más flexibles (*Integer*, *Decimal*, *Date*, *String*, etc.).
- **Staging Final**: esta capa almacena información preparada para envío final a los consumidores de la información.

Como se comenta en la sección 4.4, la plataforma en la que se desarrolla el proyecto en un *sandbox* que permite un entorno de pruebas aislado y seguro en el que desarrollar y ejecutar código. Todo el proyecto se desarrolla en la capa *Master* y dependiendo de la fase, diseño, implementación o validación, en el entorno *work* o *live*.

5.3. Análisis de Requisitos

Una vez explicado el contexto del problema, se van a describir los requisitos, tanto funcionales como no funcionales. Los requisitos son las especificaciones y condiciones que deben ser cumplidas por el desarrollo final del proyecto. La recogida, análisis y especificación de requisitos en una fase clave a la hora de desarrollar cualquier *software*, ya que gracias a ello se pueden aclarar objetivos y retricciones que deben cumplirse para un desarrollo satisfactorio. Estos requisitos se dividen en dos categorías [41]:

- **Requisitos Funcionales**: estos requisitos definen los servicios o funciones que el desarrollo del proyecto debe proporcionar. Podrían expresarse como objetivos del desarrollo. Entre los requisitos funcionales, también se describen qué información y qué datos se deben almacenar para poder ofrecer los servicios necesarios.

- **Requisitos No Funcionales**: estos requisitos definen las restricciones o limitaciones que afectan a los servicios o funciones del proyecto, es decir, a los requisitos funcionales. Estos requisitos pueden ser restricciones de tiempo, recursos, determinadas tecnologías a emplear o incluso necesidades técnicas o legales que se deben cumplir.

La recogida inicial de requisitos en este proyecto viene de una serie de documentos proporcionados por la entidad bancaria, en hojas de cálculo de *Microsoft Excel* y en documentos de *Microsoft Word*. Además, también se aclara y facilita más información en reuniones *online* con el cliente. Cabe resaltar que en estos requisitos se menciona el término *DataFrame*, el cual se refiere a una estructura de datos tabular bidimensional. Esto permite organizar los datos en filas y columnas, facilitando su manipulación, y es ampliamente utilizado en el ámbito de análisis de datos.

Requisitos Funcionales (RF)

Código	Descripción
RF-01	El sistema, a partir de las distintas tablas de origen, devolverá una tabla con la estructura/esquema esperado tras aplicar la lógica de transformación desarrollada.
RF-02	El sistema, a partir de las distintas tablas de origen, devolverá una tabla con los campos/columnas esperados tras aplicar la lógica de transformación desarrollada.
RF-03	El sistema, a partir de las distintas tablas de origen, devolverá una tabla con los contratos/filas esperadas tras aplicar la lógica de transformación desarrollada.
RF-04	El sistema, a partir de las distintas tablas de origen, devolverá una tabla con el formato esperado tras aplicar la lógica de transformación desarrollada.
RF-05	El sistema permitirá procesar la misma lógica de transformación con una periocidad mensual.
RF-06	El sistema permitirá reprocesar la misma lógica de transformación en la misma fecha, por si ocurriera algún error y fuera necesario repetir el proceso.
RF-07	La tabla final resultante de aplicar la lógica de transformación deberá permitir ser particionada por los campos *fecha_ corte* y *sistema*.
RF-08	El *DataFrame* final contará con la información general asociada a los contratos globales (código local, sistema, expediente, país, entidad, tramo, oficina, cliente), incluyéndo sus marcas y categorías de sostenibilidad y sus fechas e importes asociados, haciendo un total de 23 campos. Estos campos y el tipo de datos correspondiente a cada uno de ellos deben ser los especificados en la tabla 5.1, en ese mismo orden.

Tabla 5.1: Requisitos funcionales (RF)

CAMPO	contrato	contrato_local	sistema	expediente	pais
TIPO	String	String	String	String	String

CAMPO	entidad	marca_sost	cat_sost	fecha_sost	oficina
TIPO	String	String	String	Date	String

CAMPO	cliente	tramo	porc_total	fecha_operacion	tipo_novacion
TIPO	String	String	Decimal (12, 9)	Date	String

CAMPO	imp_dispu	imp_dispo	imp_ext	importe_total	grupo_id
TIPO	Decimal (26, 6)	Decimal (26, 6)	Decimal (26, 6)	Decimal (26, 6)	String

CAMPO	proy_id	fecha_proy	fecha_corte
TIPO	String	TimeStamp	Date

Figura 5.1: Campos y tipos de datos de los campos del DataFrame final

Requisitos No Funcionales (RNF)

Código	Descripción
RNF-01	La lógica de transformaión debe aplicarse en la capa *Master* por estándares fijados por la entidad bancaria.
RNF-02	La lógica de transformaión debe cumplir unos estándares de calidad fijados por la entidad bancaria.
RNF-03	Las tablas con los datos en origen se encuentran en formato Apache PARQUET en la capa *Master*.
RNF-04	La tabla final resultante de aplicar la lógica de transformación será un *DataFrame* que será volcado en una tabla en formato PARQUET en la capa *Master*.
RNF-05	En la tabla final resultante, con los datos ingestados, la información confidencial debe estar cifrada según los estándares de la entidad bancaria.
RNF-06	La estructura e información acerca de los datos de origen y la ruta en la que están almacenados viene descrita en un documento proporcionado por la entidad bancaria.
RNF-07	La estructura que que tiene el *DataFrame* final y la ruta en la que se debe almacenar viene descrita en un documento proporcionado por la entidad bancaria.
RNF-08	Todo el desarrollo del proyecto se realizará empleando las herramientas del entorno de trabajo proporcionado por la entidad bancaria. El lenguaje utilizado será *Python*, empleando la librería *PySpark* con un *kernel spark-python* en el entorno de *Jupyter Notebook*, todo ello facilitado en un *sandbox* propio dle banco.
RNF-09	La lógica de transformación detallada debe almacenarse paso por paso y documentarse en un documento *Microsoft Excel*.
RNF-10	Las tablas de origen deben pertenecer a la entidad bancaria y estar almacenadas a su plataforma. Estas tablas provienen de otras ingestas previas o de información añadida manualmente.

Tabla 5.2: Requisitos no funcionales (RNF)

No se ha considerado necesario, una vez inmerso en el problema, la realización de diagramas de casos de uso, aunque inicialmente estaba planificado. Estos diagramas son representaciones visuales de las interacciones entre un sistema y sus actores en diversos escenarios de un sistema de *software*, siempre basadándose en los requisitos de este. La razón por la que no se ha considerado necesario el desarrollo de dichos diagramas se basa en que el desarrollo de la solución consiste en un proceso automatizado de procesamiento en *batch* y sin interacción con usuarios, ya que las distintas ejecuciones se planifican según mallas (como se comenta en la sección 6.1.3) organizadas por Control-M, una herramienta diseñada para automatizar flujos de trabajo.

Capítulo 6

Desarrollo de la Solución

Una vez analizados y comprendidos los requisitos y requerimientos planteados por la entidad bancaria cliente, en este capítulo se muestra la solución obtenida y los pasos hasta llegar a ella, mostrando todo el análisis, diseño y desarrollo de la lógica creada. Cabe recordar que, por temas de confidencialidad, se emplean *pseudónimos* en el nombre de tablas y campos y restringe completamente la mención de cualquier contenido sensible.

6.1. Desarrollo de la Solución

El desarrollo de la solución se estructura en dos fases principales relativas a la lógica a desarrollar: el diseño, donde se establece una guía de como realizar la implementación, y la propia implemtentación de la lógica diseñada. También, se comenta el patrón arquitectónico que permite planificar la implementación del sistema *software* que otorgue la solución al problema propuesto.

6.1.1. Aquitectura Lógica: Patrón Filtro-Tubería

Un patrón aborda un problema de diseño recurrente en ingeniería de *software*, dando una solución a dicho problema basándose en la experiencia práctica. El uso de patrones facilita el mantenimiento, la adaptabilidad y la documentación del propio sistema, aportando la estructura básica del modelo de la solución. Por su parte, los patrones arquitectónicos expresan un esquema de organización estructural para sistemas *software*, proporcionando un conjunto de subsistemas predefinidos, especificando sus responsabilidades y reglas para organizar las relaciones entre dichos subsistemas.

Es habitual que en la fase de diseño se empleen soluciones con patrones arquitectónicos que se ajusten al sistema de *software* a desarrollar y, por esta razón, se ha seleccionado el patrón

arquitectónico Filtro-Tubería (*Pipes and Filters*). Este patrón proporciona una estructura para sistemas que procesan flujos de datos, donde cada paso del proceso es encapsulado como un componente de tipo filtro y el flujo de datos entre un filtro y otro se modela mediante tuberías. Se suele aplicar en sistemas que requieren la transformación de unos datos de entrada en unos datos de salida o un resultado, identificando claramente una secuencia de pasos para resolver un problema y sin apenas interacitividad o eventos, como en el caso de este TFG. Concluyendo, el patrón Filtro-Tubería hace que cada paso de la lógica de transformación a desarrollar se realice por un componente conocido como filtro, donde todos los filtros están conectados por tuberías, que transmiten los datos de un filtro al siguiente [42]. Este enfoque, muy usado en compiladores, permite obtener modularidad, ya que cada filtro es un módulo independiente, reusabilidad de los filtros o pasos de la lógica y escalabilidad. La figura 6.1 muestra el patrón arquitectónico aplicado al caso real de este TFG, donde cada filtro es uno de los pasos especificados en la siguiente sección 6.1.2.

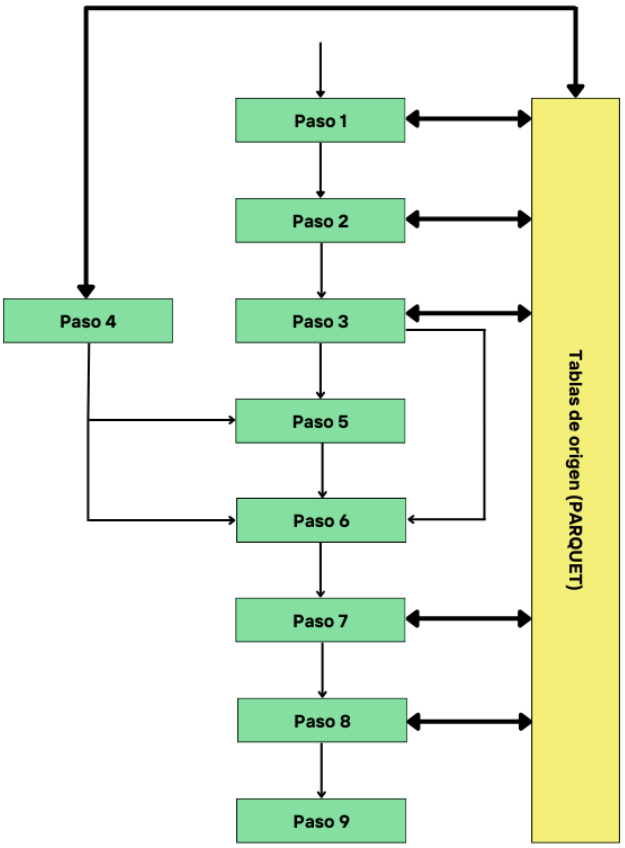

Figura 6.1: Patrón Filtro-Tubería aplicado al problema

En la anterior figura (6.1) se puede apreciar como cada filtro, que corresponde a cada paso de la lógica, está conectado a otros por tuberías. Todos los pasos, a excepción del 5, 6 y 9, están conectados con el lago de datos que almacena las tablas origen, que sirve como fuente de información global a la que varios filtros pueden acceder.

6.1.2. Diseño

Esta fase de análisis y diseño implica definir cómo los datos extraídos de diversas tablas en formato PARQUET serán manipulados y preparados para lograr los requisitos necesarios y poder cargarlos el destino. El objetivo principal de esta lógica consiste en, a partir de varias tablas de origen, estructurar una nueva tabla con los campos y formatos requeridos para que el *DataFrame* resultante sea volcado en otra tabla de destino.

Se ha definido como tabla de origen principal la tabla *marcas*, ya que contiene todas las observaciones (contratos en este caso) requeridos para la tabla final. Esta tabla *marcas* se leerá para el sistema aplicativo *CLAN*. Existen otros 3 aplicativos según el sistema de origen de los datos, *MIDAS*, *FCI* y *DCM*, y todos estos aplicativos tienen las mismas tablas, pero cada uno requiere unos tratamientos y una lectura distinta.

La tabla de origen principal *marcas* contiene la información de la marca y el porcentaje de sostenibilidad asociado a una operación. Esta tabla cuenta con 19 campos, la mayoría de ellos procedentes de otras tablas según otras lógicas. Los nombres de estos campos tienen que seguir unos estándares y reglas declarados por la entidad bancaria. Su estructura se muestra en la figura 6.1, aunque por confidencialidad, se sustituirán los nombres de los campos por un pseudónimo general que lo identifique:

Campo	Tipo	Descripción
pais	STRING	Identificador único del país
entidad_glob	STRING	Identificador único de la entidad financiera global
entidad	STRING	Identificador único de la entidad financiera
fecha_corte	DATE	Fecha de cierre
contrato	STRING	Identificador único de contrato (local o global)
expediente	STRING	Identificador único del expediente
marca_sost	STRING	Identificador del tipo de marca sostenible
cat_sost	STRING	Identificador del tipo categoría sostenible
subcat_sost	STRING	Identificador del tipo categoría sostenible
porc_subcat	DECIMAL	Porcentaje del contrato asociado a la subcategoría sostenible
fecha_sost	DATE	Fecha en la que la operación se marca como sostenible
estado	STRING	Tipo de estado sostenible
sistema	STRING	Tipo del sistema origen de sostenibilidad
certif	STRING	Indicador de certificación de sostenibilidad
emisiones	STRING	Identificador de emisiones sostenibles
fecha_verif	DATE	Fecha de verificación de sostenibilidad
proy_id	STRING	Identificador del proyecto de auditoría
proy_nombre	STRING	Nombre del proyecto de auditoría
fecha_proy	TIMESTAMP	Fecha de auditoría del proyecto

Tabla 6.1: Campos de la tabla de origen principal marcas

A continuación, se muestran detalladamente los pasos a seguir para la lógica desarrollada. En el apéndice C de los anexos, la tabla C.1 muestra el pseudocódigo perteneciente a cada paso de la lógica, con todas las indicaciones y aclaraciones necesarias para desarrollarla.

Paso 1

Leemos la tabla *marcas* particionada por *sistema CLAN*, quedándonos sólo con las observaciones de ese sistema, y filtramos para obtener únicamente las observaciones con la *fecha_corte* menor o igual a la fecha del último día del mes anterior, ya que son las observaciones que nos interesan de nuestra tabla de origen. En el caso de una primera ejecución, la fecha de corte que buscamos es "2024-03-31".

Existen particiones de la tabla por *fecha_corte* y por *sistema* que permiten cargar menos datos en la lectura. Al leer la partición por *sistema*, nuestra tabla contiene todos los campos iniciales a excepción del campo *sistema* y, como nos interesa tenerlo en nuestro *DataFrame*, lo agregamos a todas las observaciones. Como no queremos todos los campos, seleccionamos los once que necesitamos (*pais*, *entidad*, *fecha_corte*, *contrato*, *expediente*, *marca_sost*, *cat_sost*, *fecha_sost*, *sistema*, *proy_id* y *fecha_proy*).

Paso 2

Necesitamos únicamente los contratos con código global. La tabla *diccionario*, entre otros campos, contiene la traducción para cada contrato entre su código local y su código global, ambos identificadores únicos del contrato. Si el código global de un contrato no se encuentra en la tabla *diccionario* significa que ese contrato no posee código global, únicamente tendría código local. Para esta tabla *diccionario* se deben hacer dos lecturas distintas. El sistema de origen de nuestro aplicativo *CLAN* tiene dos variantes, *CLAN* y *CLAN-RX* (operaciones de *CLAN* que llegan desde otro aplicativo) y cada variante necesita una lectura distinta de la tabla *diccionario*, empleando ciertos filtros relativos a la naturaleza de los datos. Por ello, realizamos esta operación para las dos lecturas de nuestro *diccionario*.

Además, para estos contratos globales queremos conocer también su contrato local. Por tanto, nos interesa quedarnos con la relación *código local - código global* más actual, porque esta podría haber cambiado con el paso del tiempo. Debemos entonces crear una ventana o *window* que particione la tabla *diccionario* por *contrato_global* (también vale por *contrato_local*, ya que la relación es uno a uno), lo ordenamos por *fecha_corte* descendentemente y nos quedamos con la primera observación de cada contrato (que contendrá la fecha más alta, es decir, la mas reciente). Una vez hecho esto para cada lectura del diccionario (para *CLAN* y para *CLAN-RX*), unimos los dos conjuntos de datos en uno al que llamaremos *diccionario_total*.

Tenemos entonces nuestra tabla *marcas* con códigos de contratos locales y globales, y nuestra tabla *diccionario_ total* que relaciona ambos. Ahora, debemos crear dos *DataFrames* independientes que luego se unirán. Podemos asociar esto a dos subpasos:

- Debemos quedarnos los contratos con código global en *marcas* y añadir su código local, ubicado en el *diccionario*. Para ello, todo contrato de nuestra tabla *marcas* que cruce con la tabla *diccionario_ total* por el campo *marcas.contrato* igual a *diccionario_ total.contrato_ global* tendrá código global. Mediante un *inner join* completamos este sub-paso. Llamaremos a esta tabla *cruce1*.

- Debemos averiguar si los contratos con código local que tenemos en *marcas* tienen código global y, si lo tienen, quedárnoslos. Los contratos que no tengan código global descartarlos. Para ello, hacemos un *inner join* de *marcas* con *diccionario_ total* por el campo *marcas.contrato* igual a *diccionario_ total.contrato_ local* y reordenamos los campos del *DataFrame* resultante para que tenga la misma estructura que *cruce_ 1*. Llamaremos a esta tabla *cruce_ 2*.

Una vez hecho esto, hacemos una unión de las tablas *cruce_ 1* y *cruce_ 2* para juntarlo en una misma tabla. Esta tabla contendrá la misma estructura que la selección hecha en el paso 1 en *marcas*, junto al campo *contrato_ local* y conteniendo todas las observaciones únicamente contratos con códigos globales con su correspondiente código local.

Paso 3

Ahora, cruzaremos nuestra tabla resultante del paso 2 con la tabla *facility*. Esta tabla *facility*, que contiene información relativa al expediente asociado un contrato, debe leerse con unos filtros específicos que no aportan valor a este proyecto, ya que vienen impuestos por la propia naturaleza de los datos.

El cruce con *facility* será por los campos *expediente* y *contrato_ local* y de tipo *left join*, ya que no nos interesa perder ningún contrato. De la tabla *facility* nos interesa quedarnos con los campos *oficina*, *cliente*, *tramo*, *porc_ total* y *fecha_ operacion*. El campo *porc_ total* indica el porcentaje del importe del contrato del que es dueño la entidad bancaria.

Paso 4

Sobre la tabla obtenida del paso 3, queremos sacar los distintos importes según la tabla de *balance*. Para ello, primero leemos esta tabla *balance*, que contiene información acerca de los importes asociados a los contratos y expedientes, y seleccionamos los campos *entidad*, *oficina*,

cliente, *tramo*, *expediente*, *imp_ dispu* (corresponde con el importe del saldo vivo dispuesto de la operación), *imp_ dispo* (corresponde con el importe de la parte del saldo vivo disponible), *imp_ ext* (correspondiente con un importe extra asociado) y *fecha_ corte_ blc* (es una fecha distinta a la fecha de corte de la tabla *marcas*).

Antes de realizar el cruce con *balance*, necesitamos calcular el importe total del saldo vivo. Para ello, se necesita sumar los importes dispuestos de los contratos del mismo tramo y oficina para todos sus clientes y a esa suma, agregarle el importe disponible. Esto lo hacemos creando otro *DataFrame* en el que se encuentren los contratos agrupados por los campos *expediente*, *entidad*, *oficina*, *tramo* y *fecha_ corte_ blc*, realizando una agregación de su saldo dispuesto en un nuevo campo *suma*. Ahora, cruzamos con un *inner join* para agregar este nuevo campo *importe_ total* con el resto de la tabla resultante hasta el momento.

Después, creamos una nueva columna sobre con la suma del agregado anterior (campo *suma*) y el saldo disponible de cada contrato, con el nombre de *importe_ total*.

Paso 5

Realizamos un cruce de tipo *inner join* de nuestra tabla del paso 3 con la tabla *balance*, por los campos *expediente*, *entidad*, *oficina*, *cliente* y *tramo*, ya que son los coincidentes entre ambas tablas que identifican a un contrato, al no disponer del identificador local o global en la tabla *balance*.

Como nos interesa quedarnos con la observación del contrato más reciente, hacemos una *window*, agrupando por los identificadores de contrato mencionados anteriormente.

Paso 6

Probablemente, en el cruce con *balance* no todos nuestros contratos hayan cruzado. Nos interesa no perder ningún contrato, por lo que añadiremos estos que no han cruzado con valores *null* en sus campos de importes. Esto lo hacemos con un *left anti join* con los mismos campos de cruce que el *inner join* anterior, agregando los campos *suma*, *importe_ total*, *imp_ dispu*, *imp_ dispo*, *imp_ ext* y *fecha_ corte_ blc* con valor *null*. Estos contratos que no cruzaron inicialmente los unimos ahora con los que sí cruzaron.

Paso 7

Leemos ahora la tabla *cliente*, que incluye las relaciones entre los miembros de un grupo con composición normativa. Cruzaremos con esta tabla con un *left join* para no perder registros, mediante el campo de cruce *cliente*. Además, debemos filtrar por los que el campo *relacion_ principal*

sea igual a *"Y"*. Con esto, obtendremos el campo *grupo_id* asociado a la relación del contrato con el grupo asociado.

Paso 8

También queremos saber que contratos son novaciones. Para ello, debemos leer la tabla *novaciones*, que contiene información acerca de todas las novaciones existentes, agrupando por sus campos *archivo* y *delta_archivo*, debido a la naturaleza de los datos y cómo están almacenados.

Luego, cruzamos la tabla *novaciones* agrupada con la tabla del paso 7 por *expediente* igual a *archivo* y por *tramo* igual a *delta_archivo*. Con esto, creamos el campo *tipo_novacion* y, si el cruce anterior ocurre, *tipo_novacion* será igual a *"Y"* y si no ocurre, igual a *"N"*. Esto lo hacemos mediante un *inner join* (para los cruzan), un *left anti join* (para los que no cruzan) y una unión final de ambas tablas. Además, los contratos con *año_prod_nov* distinto al año actual, que en este caso sería 2024, serán también de tipo *"N"*.

Paso 9

Por último, seleccionamos y reordenamos los campos para que queden tal y como se requiere (*contrato*, *contrato_local*, *sistema*, *expediente*, *pais*, *entidad*, *marca_sost*, *cat_sost*, *fecha_sost*, *oficina*, *cliente*, *tramo*, *porc_total*, *fecha_operacion*, *tipo_novacion*, *imp_dispu*, *imp_dispo*, *imp_ext*, *importe_total*, *grupo_id*, *proy_id*, *fecha_proy* y *fecha_corte*).

Con esto se completa la lógica buscada, obteniendo finalmente el *DataFrame*, con los campos, contratos, formato y estructura buscada.

6.1.3. Implementación

Con el diseño de la lógica finalizado, pasamos a la fase de implementación. Para ello, se emplea *PySpark* en el entorno *work* para comprobar que la lógica ideada funciona correctamente antes de enviarse a producción.

Una vez que se ha completado la implementación de todos los pasos de la lógica, se realiza una primera validación inicial tal y como se indica en el apartado 6.2.1. Cuando el resultado de esta primera validación es favorable, se envía la lógica a un equipo de desarrollo propio de la entidad bancaria, para que este lo desarrolle y ponga en producción este nuevo flujo de datos.

Este equipo de desarrollo seguirá los pasos indicados, empleando tecnologías específicas de la entidad bancaria o emplenado *Scala Spark* dependiento de la complejidad de la lógica. Este equipo de desarrollo se centra en, a través de los pasos de la lógica enviada, realizar las transformaciones

indicadas tanto en la capa *Raw* (transformaciones muy sencillas o cambios de formato) como en la capa *Master* (transformaciones más complejas). Esto se logra a través de mallas o cadenas de *jobs* que se ejecutan en un orden estricto, intercalando *jobs* de ingesta con *jobs* de calidad, verificando que se cumplen los estándares de calidad fijados. Un *job* consiste en un fragmento de código o programa que está listo para ser ejecutado de forma automática. Además, el equipo de desarrollo también realiza validaciones y pruebas en los entornos *local* (para pruebas unitarias como el funcionamiento de funciones o métodos independientes) y *work* (para pruebas integradas como comprobar el funcionamiento del sistema en conjunto).

Una vez realizado y verificado todo esto, se realiza la puesta en producción y se ejecuta de forma automática la ingesta, que después se verificará a través de la segunda validación, comentada en el apartado 6.2.2.

6.2. Validación de la Solución

Es muy importante comprobar que la solución propuesta cumple los requisitos y objetivos establecidos. Para ello, se valida esta solución hasta en dos niveles: pruebas sobre la propia lógica en el entorno *work* y pruebas sobre la ingesta final en producción o *live*.

6.2.1. Pruebas en la Lógica

La primera parte de esta fase consiste en una validación en el entorno *work*, previa a la puesta en producción. En este proceso se prueba que la lógica diseñada funcione correctamente, probando la propia lógica en la misma tabla de origen particionada en distintas fechas y comparando estos resultados con los reales esperados. Estas comprobaciones se basan principalmente en comprobaciones de duplicados con conteos de observaciones totales, conteos de observaciones distintas, agrupamientos, cruces *left anti-join* para obtener los contratos distintos entre dos *DataFrames*, visualizaciones del *DataFrame* por pantalla para verificar sus campos y estructura. Todas estas comprobaciones se detallan en el siguiente apartado (6.2.2).

Una vez que estas comprobaciones se han realizado exitosamente, la validación de la lógica se da por favorable y se envía al equipo de desarrollo, como ya se comentó anteriormente.

6.2.2. Pruebas en la Ingesta

Esta validación se realizará una vez que la lógica de transformación haya sido puesta en producción por el equipo de desarrollo. Entonces se realiza la ingesta de datos sobre una tabla en una ubicación especial para comprobaciones, en el entorno *live*. Es entonces cuando se volverán

a leer estas tablas, de nuevo usando PySpark, para realizar comprobaciones que verifiquen que la ingesta se ha realizado correctamente. Para ello, fundamentalmente se realizan el mismo tipo de comprobaciones que en la validación anterior, que principalmente consisten en emplear los siguientes métodos (para más información acerca de estos métodos consultar el anexo B):

- **count()**: se realizan recuentos de contratos, comparando el número de observaciones de la tabla final con los esperados. También se comparan este número de observaciones con los obtenidos en tu entorno *local* al aplicar la lógica por tu cuenta. En caso de que haya diferencias, se realiza un estudio sobre los contratos que sobran o faltan, usando métodos como *join()* y *show()* para verificar que campos o contratos están provocando la inconsistencia. A menudo, *count()* se emplea también con ***distinct()*** para cerciorar que no solo el número de contratos totales coincide, si no también el número de contratos distintos, asegurando así que no se ha perdido ningún contrato por el camino.

- **join()**: es de gran utilidad realizar un cruce *left anti-join* para obtener las observaciones que sobran o faltan en una tabla. Este tipo de cruce devuelve el conjunto de datos que tiene todas las filas del conjunto de datos A que no coinciden con el conjunto de datos de la B, según la orden *A.join(B, on = campos , how = 'left_join')*, siendo *campos* todas las columnas del *DataFrame*. Esto permite obtener los contratos que tu lógica obtiene pero no se han ingestado (*A join B*) y los contratos que se han ingestado pero tu lógica no los tenía en cuenta (*B join A*). Para asegurar que no se pierde ni se añade ningún contrato no contemplado, estos cruces deben devolver cero contratos.

- **printSchema()**: para comprobar el esquema o estructura del *DataFrame*. Permite ver los campos y su tipo de dato asociado.

Durante la fase de validación de este proyecto, no se ha detectado ningún fallo o error inesperado, por lo que todo se ha efectuado de la manera programada y deseada. Si se ha tenido que emplear algún cambio de tipo de dato con el método *cast()* y se han arreglado pequeños incovenientes como duplicidades en algún contrato o pérdidas de contratos durante algún *join* de la lógica. El resumen de estas validaciones se pueden apreciar en la tabla C.1 del apendice C de los anexos. En este resumen se muestran las pruebas y validaciones principales realizadas, indicando el tipo de prueba en cada paso, el fallo encontrado si este existe y la corrección realizada para corregirlo. Finalmente, la nueva lógica ha sido puesta en producción e ingestada de manera correcta.

6.3. Resumen y Pasos Siguientes

Para aclarar todo el proceso anterior, podemos dividirlo en los siguientes apartados:

- Primero, se analiza, diseña e implementa la lógica de transformación y se realiza la primera validación en el entorno *work*.
- Segundo, se envía esta lógica al equipo de desarrollo, que la pone en producción realizando la ingesta, después de validar sus propias pruebas, tanto unitarias como integradas.
- Tercero, se accede a los nuevos datos de la tabla ingestada en el entorno *live* y se realiza la segunda validación, asegurando que la ingesta se ha efectuado correctamente y sin perder ningún dato.
- Finalmente, cuando la validación del punto anterior se ha completado favorablemente, los datos ingestados se extraen a un archivo CSV que ya está listo para ser explotado y analizado. Este fichero está ya preparado para realizar informes, *dashboards* y análisis descriptivos, entre otros. Para ello, otro equipo especializado en este ámbito emplea *MicroStrategy*, que proporciona herramientas de visualización para poder sacar el máximo valor y beneficio posible a los datos presentes.

Capítulo 7

Descripción de las Iteraciones

En este capítulo se muestra el progreso del proyecto dividido en *sprints* y en semanas, según la planificación inicial realizada en la sección 2.1. Se muestra la duración, trabajo realizado y dificultades de cada *sprint* y semana, así como los comentarios que se consideren relevantes para el seguimiento del proyecto.

7.1. Sprint 1 (04/03/2024 - 17/03/2024)

Durante este primer *sprint* el objetivo principal se ha centrado en convocar reuniones con los tutores, tanto académico como de empresa, para establecer claramente los objetivos y metas del proyecto. Una vez con esto, se ha realizado la planificación inicial del proyecto y la estimación de costes y riesgos.

7.1.1. Semana 1 (04/03/2024 - 10/03/2024)

Durante esta primera mitad del *sprint* se han realizado las reuniones comentadas y se ha realizado y se ha incluido en la memoria la planificación inicial, fijando la metodología a seguir y realizando los diagramas de Gantt y WBS.

7.1.2. Semana 2 (11/03/2024 - 17/03/2024)

En la segunda semana se ha realizado la estimación de costes y el análisis de riesgos, incluyéndo ambos en la presente memoria.

Conclusión del Sprint 1

Este primer *sprint* ha sido introductorio al trabajo y se han cumplido todos los objetivos marcados en él sin ninguna dificultad destacable.

7.2. Sprint 2 (18/03/2024 - 31/03/2024)

Este segundo *sprint* se ha enfocado al estudio e investigación de los conceptos claves de este proyecto, tanto teóricos como tecnológicos y prácticos. Al mismo tiempo, se ha a recolectado la bibliografía de utilidad para más tarde agregar la información que se considere relevante de estas referencias a la memoria.

7.2.1. Semana 3 (18/03/2024 - 24/03/2024)

La semana 3 del proyecto se ha enfocado en el estudio teórico del proyecto, con la recolección y posterior lectura de artículos, páginas web y publicaciones relacionadas con el tema. Esta información se ha comenzado a plasmar en la memoria en el capítulo 3.

7.2.2. Semana 4 (25/03/2024 - 31/03/2024)

En esta semana se ha realizado el estudio tecnológico y se ha recopilado información sobre las tecnologías a usar para que el desarrollo de la solución proyecto sea más fluído y eficiente. Se ha completado también el capítulo 3 de la memoria, comenzado en la semana 3.

Conclusión del Sprint 2

Este segundo *sprint* se ha enfocado en la investigación y estudio de los marcos teóricos, tecnológicos y prácticos del proyecto y se ha completado el capítulo 3 de la memoria. No ha habido ninguna dificultad más allá de buscar fuentes de información de valor y contrastarlas, aunque me hubiera gustado haber comenzado a redactar el capítulo del marco tecnológico de la memoria. El siguiente *sprint* comienza el 01 de abril, fecha marcada como hito en 2.1.2, y todas las tareas asociadas a esta fecha están completadas satisfactoriamente.

7.3. Sprint 3 (01/04/2024 - 14/04/2024)

Durante este *sprint* se ha realizado el capítulo 4 de la memoria y la recogida de requisitos, a través de varios documentos prporcionados por el cliente.

7.3.1. Semana 5 (01/04/2024 - 07/04/2024)

Esta semana se ha realizado por completo el capítulo 4 y se ha comenzado con la recogida de requisitos, a través de varios documentos, donde se ha anotado toda la información que la entidad bancaria precisa necesarias.

7.3.2. Semana 6 (08/04/2024 - 14/04/2024)

La sexta semana se ha dedicado a filtrar y especificar los requisitos a través de la información recogida durante la anterior semana, introduciéndolos directamente en la memoria en forma de tabla (sección 5.3) y aclarando las dudas acerca de ellos con mi tutora de empresa. Con esto, ya se puede comenzar el análisis, diseño y desarrollo de la lógica en el siguiente *sprint*.

Conclusión del Sprint 3

En este *sprint* se ha concluido la documentación del contexto del proyecto (marco teórico y marco tecnológico) y se ha realizado la recogida y análisis de requisitos, parte fundamental para poder realizar la lógica en el siguiente *sprint*. En este *sprint* estaba previsto realizar los diagramas UML necesarios acerca de los posibles casos de uso, pero como se comenta en el capítulo 5, no se ha considerado necesario. Esto ha provocado que se haya podido avanzar más en el desarrollo de la memoria.

El próximo *sprint* comienza el 15 de abril, marcado como hito del proyecto, y las tareas marcadas para esta fecha clave están completadas, por lo que podemos dar por muy buena la planificación del proyecto por ahora.

7.4. Sprint 4 (15/04/2024 - 28/04/2024)

Durante este cuarto *sprint* se ha avanzado en el desarrollo de la memoria y se ha realizado el diseño, análisis y creación de la lógica de transformación.

7.4.1. Semana 7 (15/04/2024 - 21/04/2024)

Durante esta semana se ha completado el capítulo introductorio de la memoria (1), así como el capítulo 5, recogiendo información centrada en el problema específico y en la tecnología de la entidad bancaria. Se ha comenzado también el análisis y diseño de la lógica, realizando los primeros paso de ella.

7.4.2. Semana 8 (22/04/2024 - 28/04/2024)

La semana 8 se ha empleado para finalizar el diseño de la lógica, diseñando y desarrollando todos los pasos necesarios para llegar hasta la tabla final deseada. No se ha finalizado por completo la lógica, debido a algunas dificultades en su diseño, pero se ha avanzado en gran medida.

Conclusión del Sprint 4

En este *sprint* se ha realizado casi por completo el diseño de la lógica, lo cual es una parte fundamental del proyecto. El diseño y desarrollo han presentado algunas dificultades, que gracias a la ayuda de mi tutora de empresa han sido solventadas, pero han supuesto un breve retraso respecto la planificación.

7.5. Sprint 5 (29/04/2024 - 12/05/2024)

En este quinto *sprint* se ha completado el diseño de la lógica y se ha implementado, desarrollando también la primera validación, correspondiente a las pruebas en la propia lógica antes de enviarla al equipo de desarrollo.

7.5.1. Semana 9 (29/04/2024 - 05/05/2024)

Esta semana se ha finalizado por completo el diseño de la lógica y se ha comenzado con su implementación. Según la planificación incial debería haberse finalizado la implementación esta misma semana, pero debido al breve retraso provocado en el *sprint* anterior no ha sido posible.

7.5.2. Semana 10 (06/05/2024 - 12/05/2024)

En la semana 10 se ha logrado finalizar la implementación y realizar las primeras pruebas de validación sobre la lógica. Esta validación ha provocado algún cambio en la implementación de la

lógica, pero apenas ha modificado el diseño de esta.

Conclusión del Sprint 5

En este *sprint* se ha recuperado el pequeño retraso provocado en el *sprint* 4. Esto se ha realizado aumentando la carga de trabajo de la semana 10, cumpliendo con todas las tareas previstas hasta esta fecha. Además, se ha enviado el diseño de la lógica al equipo de desarrollo en cuanto se ha validado.

7.6. Sprint 6 (13/05/2024 - 26/05/2024)

En principio, este es el último *sprint* del proyecto. Durante este *sprint* se ha realizado la validación de la lógica sobre la ingesta final, realizada por el equipo de desarrollo, y se ha redactado el capítulo 6.

7.6.1. Semana 11 (13/05/2024 - 19/05/2024)

Durante esta semana se ha ido preparando el código correspondiente a la validación sobre la ingesta, comprobando que no se ha perdido ningún contrato y que todo esta según los requisitos. No se ha podido ejecutar ese código y validar la ingesta debido a que el equipo de desarrollo no ha completado la puesta en producción de la lógica. Este retraso provocado por factores externos estaba contemplado en los riesgos estudiados en el capítulo 2 y no ha influido tanto en el global del proyecto ya que se ha aprovechado para ir redactando el capítulo de 6 en la memoria.

7.6.2. Semana 12 (20/05/2024 - 26/05/2024)

Durante esta semana se ha completado la redacción del capítulo 6 y el equipo de desarrollo ha completado la ingesta, evitando un retraso mayor que habría supuesto una demora en la entrega final del proyecto. Gracias a ello, se ha completado la validación final de dicha ingesta, con un resultado exitoso y sin ningún problema a mayores.

Conclusión del Sprint 6

Este *sprint* ha sido complicado por la carga del trabajo externo, ya que aparte de lo propio de este proyecto y el retraso asociado al equipo de desarrollo, se ha realizado también el examen final de una asignatura. Por todo ello, aunque la productividad ha sido alta, no se ha finalizado

por completo el Trabajo Fin de Grado. Falta por redactar algún aspecto más de la memoria, así como revisar y corregir el trabajo realizado. Por ello, se empleará un *sprint* extra, no programado en la planificación incial pero sí contemplado en ella.

7.7. Sprint Extra (27/05/2024 - 09/06/2024)

Este *sprint* extra se ha empleado debido a pequeños retrasos durante las semanas en las que se ha realizado el proyecto. Se ha enfocado en realizar el capítulo 8 y revisar todo el desarrollo de la memoria, en busca de errores y de mejorar pequeños aspectos, todo ello con la ayuda del tutor académico y de la tutora de empresa. Este *sprint*, por tanto, ha servido para finalizar el Trabajo Fin de Grado, dando por concluida esta memoria.

Capítulo 8

Conclusiones y Trabajo Futuro

Durante todo el proyecto se ha podido constatar la enorme relevancia que tiene el *Big Data* en la actualidad, así como sus características, la forma en que se gestiona y las principales tecnologías que tratan con él. También se ha profundizado sobre las ingestas de datos y los procesos ETL, así como sus variantes, formas de uso y tecnologías relacionadas con ello.

De igual manera, tras finalizar este Trabajo Fin de Grado, se puede afirmar que se han cumplido los objetivos. Se ha logrado documentar por completo el proceso ETL por el cual ciertos datos reales de caracter financiero pasan de un estado crudo a un estado en el que son explotables y analizables, ya preparados para poder obtener información y valor real acerca de ellos. Además, se ha logrado diseñar, desarrollar, implementar y validar la lógica de transformación que permite el resultado esperado del proceso ETL, así como la posterior validación de la ingesta final resultante de dicho proceso.

En un plano más personal, he adquirido conocimientos y aprendizajes de gran valor sobre el tema del que este proyecto trata. Desde el inicio, este proyecto ha incrementado mi interés ya existente acerca del *Big Data* y de la multitud de diferentes tecnologías que lo rodean. Me ha permitido tener una visión global de muchas de las tecnologías punteras en procesos ETL y en ingestas de datos, así como entender cómo funcionan estos procesos con flujos de trabajos automatizados. Además, me ha brindado la oportunidad de trabajar en un caso real, viendo el desempeño real de todos los conceptos aprendidos y permitiéndome aplicarlos, aportando un valor tangible para la empresa *NTT DATA Spain S.L.U.* y para la entidad bancaria cliente.

Pese a todo esto, siempre quedan líneas de trabajo abiertas y aspectos de mejora. Sería de gran interés poder realizar la puesta en producción de la lógica de tranformación diseñada e implementada, realizando con ello la ingesta final de los datos. Esto permitiría adquirir nuevos conocimientos y habilidades en otro tipo de tecnologías. Además, el *Big Data* es un mundo gigantesco y completamente en expansión y crecimiento, por lo que queda abierto el marco de estudio y conocimiento de muchísimos más conceptos y tecnologías relacionadas con el tema en el que está enfocado este

proyecto. También sería de gran interés realizar visualizaciones y análisis con los datos ingestados, una vez transcurrido todo el proceso ETL y su ingesta, para poder extraer información y valor real a los datos, obteniendo además habilidades en otro tipo de tecnologías.

Bibliografía

[1] *Salario medio para Ingeniero Informático en España, 2024.* Talent, 2024. [Online]. Available: https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico

[2] Latitude 5440 portátil. [Online]. Available: https://www.dell.com/es-es/shop/port%C3%A1tiles-de-dell/port%C3%A1til-latitude-5440/spd/latitude-14-5440-laptop

[3] Monitor hp m24fw de 60,45 cm (23,8"), fhd, amd freesync™. [Online]. Available: https://www.hp.com/es-es/shop/product.aspx?id=2D9K1AA&opt=ABB&sel=MTO

[4] D. Hillson and D. T. Hulett, *Assessing risk probability: alternative approaches.* Project Managment Institute, 2004, https://www.pmi.org/learning/library/assessing-risk-probability-impact-alternative-approaches-8444.

[5] *Guía para entender la matriz de riesgo 5x5.* SafetyCulture, 2024, https://safetyculture.com/es/temas/evaluacion-de-riesgos/matriz-de-riesgo/.

[6] *Programa de doble titulación oficial: Grado en Ingeniería Informática y Grado en Estadística.* Universidad de Valladolid, 2023. [Online]. Available: https://www.uva.es/resources/docencia/_ficheros/2023/551/asignaturas.pdf

[7] *¿Qué es la era de los datos y cuándo comenzará.* Anáhuac México, 2023, https://www.anahuac.mx/mexico/noticias/Que-es-la-era-de-los-datos-y-cuando-comenzara.

[8] *50+ Incredible Big Data Statistics for 2024: Facts, Market Size and Industry Growth.* BDAN, 2024, https://bigdataanalyticsnews.com/big-data-statistics/.

[9] D. d. C. Fernández-Paniagua, *Ingesta de datos automatizada para un proyecto de Big Data.* Universidad de Valladolid, 2023, https://uvadoc.uva.es/handle/10324/62909.

[10] C. Hashemi-Pour, B. Botelho, and S. J. Bigelow, *Definition: Big Data.* TechTarget, 2024, https://www.techtarget.com/searchdatamanagement/definition/big-data.

[11] *The Important 8 Vs for Big Data Development.* SphinxWorldbiz, 2020, https://www.sphinxworldbiz.net/the-important-8-vs-for-big-data-development-infographics.

[12] *Zettabyte.* Wikipedia, 2024, https://es.wikipedia.org/wiki/Zettabyte.

[13] *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2025.* Statista, 2024, https://www.statista.com/statistics/871513/worldwide-data-created/.

[14] B. Berisha, E. Mëziu, and I. Shabani, *Big data analytics in Cloud computing: an overview.* SpringerOpen, 2022, https://publisherofcloudcomputing.springeropen.com/articles/10.1186/s13677-022-00301-w.

[15] S. Yadav, *Cloud for Big Data – What are the Key Benefits and Drawbacks.* Baseline, 2022, https://www.baselinemag.com/analytics-big-data/cloud-for-big-data-what-are-the-key-benefits-and-drawbacks/.

[16] *Big Data life cycle: understand all Big Data phases.* TokioSchool, 2022, https://www.tokioschool.com/en/news/big-data-life-cycle/.

[17] S. Hussain, *The 4 steps of the big data life cycle.* Benelux Intelligence Community, 2024, https://www.bi-kring.nl/29-business-intelligence/1353-the-4-steps-of-the-big-data-life-cycle.

[18] *El Ciclo de vida del Big Data en 4 fases.* KeepCoding, 2023, https://keepcoding.io/blog/el-ciclo-de-vida-del-big-data-4-fases/.

[19] E. Baraban, *Big Data Ingestion.* Towards Data Science, 2021, https://towardsdatascience.com/what-is-data-ingestion-5220edf50677.

[20] *¿Qué son los procesos ETL?* PowerData, 2017, https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/qu-son-los-procesos-etl.

[21] O. Fernandez, *Aprende qué es una Pipeline de Datos.* AprenderBigData, 2024, https://aprenderbigdata.com/pipeline-de-datos/.

[22] A. Sudan, *What Is ETL And How the ETL process works?* DataChannel, https://www.datachannel.co/blogs/what-is-etl-and-how-the-etl-process-works.

[23] *Etl model demonstrating data transformation.* SlideTeam, https://www.slideteam.net/etl-model-demonstrating-data-transformation.html.

[24] *Extracción, transformación y carga de datos (ETL).* Microsoft Learn, https://learn.microsoft.com/es-es/azure/architecture/data-guide/relational-data/etl.

[25] E. Sanchez, *ELT vs ETL: Main Differences Between ETL and ELT (Full Comparison).* Skyvia, 2022, https://blog.skyvia.com/elt-vs-etl/.

[26] *5 Benefits of Automated Data Ingestion.* Saras, 2022, https://sarasanalytics.com/blog/self-service-data-ingestion.

[27] M. Buenning, *What Is a Data Lake?* NinjaOne, 2024, https://www.ninjaone.com/it-hub/it-service-management/what-is-a-data-lake/.

[28] C. Bolba, *Data Warehouse vs. Data Lake vs. Data Lakehouse: Know the differences.* Medium, 2023, https://cassio-bolba.medium.com/data-warehouse-vs-data-lake-vs-data-lakehouse-know-the-differences-51bb3f82e137.

[29] *What is a Data Lakehouse?* GoogleCloud. [Online]. Available: https://cloud.google.com/discover/what-is-a-data-lakehouse?hl=es-419

[30] Isaac, *Tecnología Big Data: qué es y para que sirve.* Profesional Review, 2022, https://www.profesionalreview.com/2022/07/10/tecnologia-big-data/.

[31] M. Narang, *Big Data Technologies that Everyone Should Know in 2024.* UpGrad, 2023, https://www.knowledgehut.com/blog/big-data/big-data-technologies.

[32] *Ingesta de datos.* Inteligencia Artificial y Big Data, 2024, https://aitor-medrano.github.io/iabd/de/etl.html.

[33] *Cloud: La nube.* Inteligencia Artificial y Big Data, 2024, https://aitor-medrano.github.io/iabd/cloud/cloud.html.

[34] P. Martín, *¿Qué es Pentaho Data Integration (PDI) y para qué sirve?* Itop Academy, 2019, https://itop.academy/blog/item/que-es-pentaho-data-integration-pdi-y-para-que-sirve.html.

[35] *¿Qué es Talend Open Studio?* KeepCoding, 2023, https://keepcoding.io/blog/talend-open-studio/.

[36] *SQL Server Integration Services.* Microsoft, 2022, https://learn.microsoft.com/es-es/sql/integration-services/.

[37] J. R. de la Huerga, *Qué es Mulesoft Anypoint y principales características.* NTTData, 2021, https://ifgeekthen.nttdata.com/es/que-es-mulesoft-anypoint-y-principales-caracteristicas.

[38] *Cómo usar MuleSoft como una herramienta ETL.* RootStack, 2024, https://rootstack.com/es/blog/como-usar-mulesoft-como-una-herramienta-etl.

[39] E. Henriquez, *¿Qué es Informatica PowerCenter?* Linkedin, 2023. [Online]. Available: https://es.linkedin.com/pulse/qu%C3%A9-es-informatica-powercenter-eduardo-henriquez

[40] *Jupyter Notebook: qué es y cómo se usa.* EBAC, 2023, https://ebac.mx/blog/jupyter-notebook.

[41] *Especificación de requerimientos.* Universidad de Granda, https://elvex.ugr.es/idbis/db/docs/design/2-requirements.pdf.

[42] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns (Volume 1).* Wiley, 1996, https://github.com/techyosemite/Software-Architectures/blob/master.

Todos los enlaces fueron comprobados correctamente el 31 de mayo de 2024.

Anexos

Apéndice A

Acrónimos

- **API**: *Application Programming Interface*
- **AWS**: *Amazon Web Services*
- **CIB**: *Corporate & Investment Banking*
- **CSV**: *Comma-Separated Values*
- **ECTS**: *European Credit Transfer System*
- **ELT**: *Extract, Load and Transform*
- **ETL**: *Extract, Transform and Load*
- **GB**: *Gigabytes*
- **GUI**: *Graphical User Interface*
- **HDFS**: *Hadoop Distributed File System*
- **HTML**: *HyperText Markup Language*
- **IoT**: *Internet of Things*
- **JSON**: *JavaScript Object Notation*
- **KPI**: *Key Performance Indicator*
- **NTT**: *Nippon Telegraph and Telephone*
- **ODI**: *Oracle Data Integration*
- **PDI**: *Pentaho Data Integration*

- **RDD**: *Resilient Distributed Dataset*
- **RF**: Requisitos Funcionales
- **RNF**: Requisitos No Funcionales
- **SLU**: Sociedad Limitada Unipersonal
- **SO**: Sistema Operativo
- **SQL**: *Structured Query Language*
- **SSIS**: *SQL Server Integration Services*
- **TFG**: Trabajo Fin de Grado
- **TOS**: *Talend Open Studio*
- **UML**: *Unified Modeling Language*
- **VPN**: *Virtual Private Network*
- **WBS**: *Work Breakdown Structure*
- **XML**: *Extensible Markup Language*
- **YARN**: *Yet Another Resource Negotiator*
- **ZB**: *Zettabytes*

Apéndice B

Uso Fundamental de PySpark: Tutorial Práctico

Se muestra a continuación un breve tutorial básico e introductivo a los principales métodos que ofrece PySpark y cómo emplearlos, con ejemplos sencillos no relacionados con el tema del proyecto, por temas de confidencialidad:

B.1. Métodos Principales

Los métodos son funciones que actúan sobre los *DataFrame*s, RDD's u otros objetos en el entorno de Spark. Los métodos principales empleados durante el proyecto son:

- El método **create*DataFrame*()** sirve para crear un *DataFrame* de diversas formas, desde un RDD o introduciendo en una lista los datos. Se puede especificar la estructura del *DataFrame* creado si se considera adecuado, como se muestra en B.1.

```
datos = [("Juan", 25), ("Pedro", 18), ("Ana", 32)]

# Definimos el esquema para el \textit{DataFrame}, no es obligatorio
esquema = StructType([
    StructField("Nombre", StringType(), True),
    StructField("Edad", IntegerType(), True)])

df = spark.create\textit{DataFrame}(data = datos, schema = esquema)
```

Listing B.1: Ejemplo de uso de create*DataFrame*()

- El método **read** sirve para cargar los datos de un fichero a un *DataFrame*. Este método admite varias operaciones según la naturaleza de nuestros datos de origen. El parámetro

header se utiliza para especificar si el archivo de origen contiene un encabezado con nombres de columnas, mientras que *inferSchema* se utiliza para permitir que PySpark infiera automáticamente el esquema de los datos y *sep* indica cómo delimitan los campos del archivo.

```
df = spark.read.csv("/data/tus_datos.csv", header = True, inferSchema = True, sep = ",") # para .csv
df = spark.read.text("/data/tus_datos.txt", header = True, inferSchema = False, sep = ";") # para .txt
df = spark.read.parquet("/data/tus_datos") # para formato parquet

# Otra forma es:
spark.read.load(format = "csv", resto_de_parametros) # sustituir csv por el formato deseado
```

Listing B.2: Ejemplo de uso de read

- El método **show()** sirve para visualizar el *DataFrame*. Este método muestra por defecto las 20 primeras filas del *DataFrame*. Con la opción *truncate* se indica el número de carácteres a mostrar por celda, pero si se pone en *False* se mostará todo el contenido de la celda.

```
df.show() # por defecto muestra 20 filas y 20 caracteres por celda
df.show(n = 5, truncate = 100) # muestra 5 filas y 100 caracteres por celda
df.show(15, False) # muestra 15 filas y todos los caracteres de cada celda
```

Listing B.3: Ejemplo de uso de show()

- El método **printSchema()** sirve para visualizar el schema del *DataFrame* en forma de árbol. Este método no acepta ningún argumento. El método **toPandas()** se utiliza para convertir un *DataFrame* de PySpark en un *DataFrame* de pandas.

```
df.printSchema()
df_pandas = df.toPandas()
```

Listing B.4: Ejemplo de uso de printSchema() y de toPandas()

- El método **count()** sirve para saber cuántas filas o registros contiene el *DataFrame* sobre el que se aplica. Este método se utiliza sobretodo para validar si se han cargado bien los datos desde el fichero, si se ha aplicado bien alguna operación como puede ser un filtro o un *join* o si se han añadido o borrado algún registro en un *DataFrame*. Para ver el número de columnas o campos podemos aplicar *.columns*.

```
df.count() # numero de registros
df.columns, len(df.columns) # nombre y numero de campos
```

Listing B.5: Ejemplo de uso de count()

- El método **withColumn()** sirve para devolver un nuevo *DataFrame* con las columnas originales más la columna adicional especificada. También puede modificar una columna existente sin agregar ninguna nueva, si el nombre de la columna especificado ya se encuentra en el *DataFrame*. Como segundo argumento, se pasa el contenido de esta nueva columna, que puede formarse a partir de las columnas existentes. Con **withColumnRenamed()** se renombra una columna existente del *DataFrame*, pasando como primer argumento el nombre de la columna a renombrar y como segundo argumento el nuevo nombre.

```
df = df.withColumn("nombre_nueva_columna", df.col1 + df.cl2 + df.col3)
df = df.withColumnRenamed("nombre_viejo","nombre_nuevo")
```

Listing B.6: Ejemplo de uso de withColumn() y de withColumnRenamed()

- El método **filter()** nos permite seleccionar las filas de nuestro *DataFrame* que cumplen una determinada condición. El método **where()** es equivalente a filter(). Por otro lado, **select()** nos permite crear un nuevo *DataFrame* que contenga sólo una parte de las columnas del original. Usando **alias()** podemos renombrar una columna y **selectExpr()** permite realizar un *select* empleando sintaxis SQL. A diferencia de *filter()*, usando *select()* tendríamos las mismas filas pero sólo las columnas elegidas. Por último, el método *drop()* permite eliminar las columnas especificadas de un *DataFrame*.

```
df.filter(df.col1 > 0)
df.where(df.col1 == 0)
df.filter((df.col1 > 0) & (df.col2 == "Si"))

df.select(df.col1, df.col2, df.col5)
df.select("col1", "col2", "col5") # otra forma equivalente
df.select((df.col1 + df.col2).alias("col1_mas_col2"))

df.selectExpr("col1 + col2 as col1_mas_col2")

df.drop("col4") # excluye el campo col4
```

Listing B.7: Ejemplo de uso de filter()/where() y de select() y drop()

- El método **groupBy()** agrupa un *DataFrame* por los valores que son iguales en las columnas especificadas. Una vez agrupados nos permitirá realizar cálculos sobre las filas del *DataFrame*. Por otro lado, el método **orderBy()** ordena un *DataFrame* según una o más columnas especificadas. Con las funciones **asc()** y **desc()** se puede elegir si el ordenamiento se realiza de forma ascendente o descendente, respectivamente. Si no se especifica el orden es ascendente.

```
df.groupBy("col5") # una columna
df.groupBy(["col1", "col2", "col3"]) # varias columnas

```

```
4    df.orderBy(F.col("col1").asc()) # ascendentemente
5    df.orderBy(F.col("col1").desc()) # descendentemente
6    df.orderBy("col1", "col2") # varias columnas, en caso de empate en col1, desempata
         el orden en col2
```

Listing B.8: Ejemplo de uso de groupBy() y de orderBy()

- El método **distinct()** devuelve un nuevo *DataFrame* que contiene únicamente las filas que son distintas del *DataFrame*. Se suele emplear junto a *count()* para obetener el numero de filas distintas, es decir, eliminando los duplicados. Para eliminar los duplicados en un nuevo *DataFrame* se emplea el método **dropDuplicates()**.

```
1    df.distinct().count()
2    df = df.dropDuplicates()
```

Listing B.9: Ejemplo de uso de distinct() y de dropDuplicates()

- El método **isin()** permite saber que valores están presentes en una columna del *DataFrame*. Se puedecombinar con *filter()*, o su equivalente *where()*, para filtrar los datos de un *DataFrame* en función de si una columna está o no contenida en una lista de valores.

```
1    lista_nombres = ["Juan", "Luis", "Jorge"]
2    df.filter(F.col("nombre").isin(lista_nombres)) # solo se queda con los nombres
         iguales a Juan, Luis o Jorge
```

Listing B.10: Ejemplo de uso de isin()

- El método **join()** permite la unión de tablas por campos de cruce con valores iguales en ambas tablas. Principalmente, se emplean tres argumentos que indican el *DataFrame* a cruzar (*other*), los campos de cruce (*on*) y el tipo de cruce (*how*). Hay varios tipos de *join*:
  - **Inner Join**: devuelve el *DataFrame* cuyos valores hayan coincidido en ambos *DataFrames*.
  - **Cross Join**: devuelve el conjunto de datos, que es el número de filas del primer conjunto de datos multiplicado por el número de filas del segundo conjunto de datos. Esta multiplicación es la denominada producto cartesiano.
  - **Left Outer Join**: devuelve el conjunto de datos que tiene todas las filas del conjunto de datos izquierdo y las filas coincidentes del conjunto de datos derecho, rellenando con valores *null*.
  - **Right Outer Join**: devuelve el conjunto de datos que tiene todas las filas del conjunto de datos derecho y las filas coincidentes del conjunto de datos derecho, rellenando con valores *null*.

- **Full Outer Join**: devuelve el conjunto de datos que tiene todas las filas cuando hay una coincidencia en el conjunto de datos izquierdo o derecho, rellenando con valores *null*.
- **Left Semi-Join**: devuelve el conjunto de datos que tiene todas las filas del conjunto de datos izquierdo con su correspondencia en el conjunto de datos derecho. A diferencia de *left outer join*, el conjunto de datos devuelto en ahora contiene solo las columnas del conjunto de datos izquierdo.
- **Left Anti-Join**: devuelve el conjunto de datos que tiene todas las filas del conjunto de datos de la izquierda que no coinciden con el conjunto de datos de la derecha. También contiene solo las columnas del conjunto de datos de la izquierda. Es especialmente útil para saber que registros de una tabla no se encuentran en la otra.

```
# Inner Join
df1.join(other = df2, on = df1.col1 == df2.col1, how = "inner")

# Left Outer Join
df1.join(df2, df1.col1 == df2.col5, how = "left")

# Right Outer Join
df1.join(df2, on = "col3", how = "right") # col3 es campo de cruce presente en las
    dos tablas

# Full Outer Join
df1.join(df2, on = df1.col1 == df2.col5, how = "full")

# Left Semi-Join
df1.join(df2, on = ["col1", "col2"], how = "left_semi") # col1 y col2 son dos campos
    de cruce presentes en las dos tablas

# Left Anti-Join
df1.join(df2, on = ["col1", "col2", "col3"], how = "left_anti")
```

Listing B.11: Ejemplo de uso de los distintos tipos de join()

- El método **union()** permite combinar dos *DataFrames* verticalmente, es decir, apilar uno encima del otro devolviendo un nuevo *DataFrame* que contiene todas las filas de los otros dos.

```
union = df1.union(df2)
```

Listing B.12: Ejemplo de uso de union())

- El método **cast()** se utiliza para cambiar o convertir el tipo de datos de una columna en un *DataFrame*.

```
df.withColumn("col2", F.col("col2").cast("int")) # cambia el tipo de la columna dos
    a Integer
```

Listing B.13: Ejemplo de uso de cast())

- Para exportar a un archivo CSV el contenido de un *DataFrame*, se emplea el método **write**. Este método suele ir acompañado de **format()** indicando el formato, en este caso CSV, además de **repartition()**, que indica el número de particiones que deseas obtener y **option()**, que permite añadir más opciones como leer o no el encabezado. Con **mode()** puedes espcificar si el archivo de sobreescribirá si existe otro con el mismo nombre en el archivo especificado en **save()**.

```
df1.repartition(1).write.format("csv").option("header","true")\
.mode('overwrite').save("nombre_archivo.csv")
```

Listing B.14: Ejemplo de uso para guardar en CSV un DataFrame

B.2. Más Funciones

También se pueden emplear distintas funciones que facilitan el manejo de datos y la realización de operaciones. Las funciones más principales y básicas son:

- Función **col()**: accede a una columna por su nombre.
- Función **avg()**: calcula el promedio de los valores en una columna.
- Función **min()**: encuentra el valor mínimo en una columna.
- Función **max()**: encuentra el valor máximo en una columna.
- Función **sum()**: calcula la suma de los valores en una columna.
- Función **lit()**: crea una columna con un valor literal de cualquier tipo.
- Función **length()**: devuelve la longitud de una cadena.
- Función **concat()**: concatena una o más cadenas.
- Función **substring()**: extrae una parte de una cadena.
- Función **like()**: obtiene los valores coincidentes con lo especificado. Se suele usar con el carácter "%" para representar cero o más caracteres cualesquiera.

- Función **upper()** y **lower()** : convierte una cadena a mayúsculas o minúsculas, respectivamente.
- Función **current_date()**: devuelve la fecha actual.
- Función **current_timestamp()**: devuelve la marca de tiempo actual.
- Función **date_format()**: formatea una fecha según el formato especificado.
- Función **year()**, **month()**, **dayofmonth()**: extrae componentes de fecha de una columna, el año, el mes y el día, respectivamente.
- Función **when()**: realiza una evaluación condicional. Se utiliza con **otherwise()** para el caso en el que la evaluación no se cumpla.
- Función **isNull()** o **isNotNull()**: comprueba si una columna es nula o no nula, respectivamente.

```
# Importamos las funciones como F
import pyspark.sql.functions as F

# col
df.select(F.col("nombre")).show()

# avg
promedio_edad = df.select(F.avg("edad")).show()

# min
edad_minima = df.select(F.min("edad")).show()

# max
edad_maxima = df.select(F.max("edad")).show()

# sum
suma_edades = df.select(F.sum("edad")).show()

# lit
df.withColumn("nueva", F.lit(1)).show() # llena la columna nueva de 1's
df.withColumn("nueva", F.lit(None)).show() # llena la columna nueva de valores NULL

# length
df.select(F.length("nombre")).show()

# concat
df.select(F.concat("nombre", "apellido")).show()

# substring
```

```
30    df.select(F.substring("nombre", 1, 3)).show() # subcadena desde el primer caracter con
          longitud 3
31
32    #like
33    df.where(F.col("nombre").like("%an%")) # filtra por todos los nombres que contengan la
          subcadena "ana" (Juan, Joana, Mariana, Candela, ...)
34
35    # upper y lower
36    df.select(F.upper("nombre"), F.lower("apellido")).show()
37
38    # current_date
39    df.select(F.current_date()).show()
40
41    # current_timestamp
42    df.select(F.current_timestamp()).show()
43
44    # date_format
45    df.select(F.date_format("fecha", "yyyy-MM-dd")).show() # pone la variable fecha en el
          formato indicado
46
47    # year, month y dayofmonth
48    df.select(F.year("fecha"), F.month("fecha"), F.dayofmonth("fecha")).show()
49
50    # when y otherwise
51    df.withColumn("situacion", F.when(df.edad >= 18, "mayor").otherwise("menor")).show() # la
          variable situacion sera mayor de edad si la variable edad es 18 o mas, y sera menor de
          edad si es menor
52
53    # isNull
54    df.filter(F.col("nombre").isNull()).show()
55
56    # isNotNull
57    df.filter(F.col("nombre").isNotNull()).show()
```

Listing B.15: Ejemplo de uso de funciones

B.3. Window

Cuando se habla de *window* o **ventana** se hace referencia a una partición o marco de un *DataFrame*, o lo que es lo mismo, a un conjunto de filas determinado. **Window** es entonces una herramienta que permite realizar cálculos basados en ventanas de un *DataFrame*, proporcionando funciones que operan sobre este grupo de filas y devuelven un valor único para cada fila de entrada. Destacan tres tipos de funciones *Window*: *ranking functions*, *analytic functions* y *aggregate functions*.

Ranking Functions

Se emplean para asignar un rango o posición a cada fila dentro de un conjunto de datos basado en un criterio específico, como una ordenación por una columna determinada. Algunas funciones típicas incluyen:

- **row_number()**: asigna un número único a cada fila en el resultado de la consulta, sin considerar empates.
- **rank()**: asigna un rango a cada fila en el resultado de la consulta, con posibles empates y sin saltos entre los rangos.
- **dense_rank()**: similar a *rank()*, pero no hay saltos entre los rangos, incluso en caso de empates.
- **percent_rank()**: calcula el percentil de la fila en el resultado de la consulta.
- **ntile()**: calcula la identificación del grupo *ntile* (de 1 a n inclusive) en una partición de ventana ordenada, es decir, devuelve el identificador de ntile en una partición.

```
# Importamos las funciones y ventana
from pyspark.sql.window import Window
from pyspark.sql import functions as F

# Definicion de la vetana
windowSpec = Window.partitionBy("asignatura").orderBy("nota")

# Aplicacion de rank()
df.withColumn("rank_col",F.rank().over(windowSpec))

# Aplicacion de row_number()
df.withColumn("row_number_col", F.row_number().over(windowSpec))
```

Listing B.16: Ejemplo de uso de ventanas con funciones ranking

Analytic Functions

Se utilizan sobre un conjunto de filas de entrada y devuelven un único valor para cada fila, basado en un rango de filas especificado por una ventana. Estas funciones permiten realizar cálculos avanzados y análisis de datos en ventanas definidas, como cálculos de agregación, cálculos de valor acumulado y comparaciones entre filas adyacentes. Las funciones típicas son:

- **lead()**: se utiliza para acceder a los valores de columnas en filas posteriores dentro de una ventana. Puedes especificar el número de filas adelante desde la fila actual para obtener el valor de la columna. Si no hay filas posteriores, la función devuelve *null*.
- **lag()**: similar a la función *lead()*, se utiliza para acceder a los valores de columnas en filas anteriores dentro de una ventana. Se puede especificar el número de filas hacia atrás desde la fila actual para obtener el valor de la columna. Si no hay filas anteriores, la función devuelve *null*.
- **cume_dist()**: devuelve la distribución acumulativa de valores dentro de una partición calculando la función de distribución acumulada de los valores en una ventana ordenada. Para cada fila, devuelve el porcentaje de filas en la ventana que tienen un valor menor o igual al valor de la fila actual.

```
# Importamos las funciones y ventana
from pyspark.sql.window import Window
from pyspark.sql import functions as F

# Definicion de la vetana
windowSpec = Window.partitionBy("asignatura").orderBy("nota")

# Aplicacion de lag()
df.withColumn("lag_col", F.lag("nota", 2).over(windowSpec))
```

Listing B.17: Ejemplo de uso de ventanas con funciones analíticas

Aggregate Functions

Realizan cálculos sobre un conjunto de valores y devuelven un solo valor de salida, también se consideradan *analytic functions*. Son las ya comentadas *sum()*, *avg()*, *min()*, *max()* y *count()*.

```
# Importamos las funciones y ventana
from pyspark.sql.window import Window
from pyspark.sql import functions as F

# Definicion de la vetana
windowSpecAgg = Window.partitionBy("asigntura") # no hace falta orderBy()

# Aplicamos funciones de agregacion
df.withColumn("media", F.avg(F.col("nota")).over(windowSpecAgg))\
.withColumn("minima", F.min(F.col("nota")).over(windowSpecAgg))\
.withColumn("maxima", F.max(F.col("nota")).over(windowSpecAgg))
```

Listing B.18: Ejemplo de uso de ventanas con funciones de agregación

Apéndice C

Pseudocódigo y Validación de la Lógica

Se presenta en la siguiente página, en la figura C.1, el pseudocódigo con todos los pasos e indicaciones necesarias para llevar a cabo la lógica de transformación que resuelve el problema planteado en este Trabajo Fin de Grado. Seguidamente, la tabla C.1 muestra el resumen del proceso de validación realizado:

C.1. Pseudocódigo de la Lógica

Tablas a utilizar		
marcas	*ruta_tabla_marcas*	*ORIGEN*
diccionario	*ruta_tabla_diccionario*	*ORIGEN*
facility	*ruta_tabla_facility*	*ORIGEN*
balance	*ruta_tabla_balance*	*ORIGEN*
cliente	*ruta_tabla_cliente*	*ORIGEN*
novaciones	*ruta_tabla_novaciones*	*ORIGEN*
destino	*ruta_tabla_destino*	*DESTINO*

Variables constantes	
ODATE	*Último día del mes anterior*

Pasos a seguir	
Explicación General	Partimos de las tablas de *marcas*, *diciconario*, *facility*, *balance*, *cliente* y *novaciones* para cargar 23 campos en la tabla de *destino*, con información relativa a cada contratoy su importe. **Periodicidad:** MENSUAL con **datos de fin de mes anterior.**
Paso 1	Leemos todas las operaciones de CLAN de la tabla de marcas. **SELECT** 'pais', 'entidad', 'fecha_corte', 'contrato', 'expediente', 'marca_sost', 'cat_sost', 'fecha_sost', 'sistema', 'proy_id', 'fecha_proy' **FROM** marcas **WHERE** ('sistema' == 'CLAN' **&&** fecha_corte **<=** **ODATE**)
Paso 2	Nos quedamos todos los contratos con código global y añadimos su código local. **1 - Leemos el diccionario para CLAN en diccionario_clan.** **SELECT** 'contrato_global', 'contrato_local', 'fecha_corte' **FROM** diccionario **WHERE** ('sistema' == "CLAN" **&** ('pais' **!=** "RX" **||** 'pais' **ISNULL**)) ** **2 - Leemos el diccionario para CLAN-RX en diccionario_clanRX.** **SELECT** 'contrato_global', 'contrato_local', 'fecha_corte' **FROM** diccionario **WHERE** ('sistema' == "CLAN" **&&** 'pais' == "RX") ** ** Para ambas lecturas de CLAN (CLAN y CLAN-RX) debemos hacer una ventana y quedarnos con los registros MÁS RECIENTES para cada contrato. **3 - Unimos los dos diccionarios en diccionario_total.** diccionario_clan **UNION** diccionario_clanRX **4 - Cruzamos marcas con la unión de los diccionarios.** **A)** Añadimos el código local de los contratos con código global en cruce_1. marcas **INNER JOIN** diccionario_total **ON** [marcas.contrato == diccionario_total.contrato_global] **B)** Comprobar si los contratos con código local tienen código global en cruce_2. marcas **INNER JOIN** diccionario_total **ON** [marcas.contrato == diccionario_total.contrato_local] **C)** Unimos cruce_1 y cruce_2 en tabla_paso_2. cruce_1 **UNION** cruce_2
Paso 3	Cruzamos la tabla resultante del *Paso 2* con la tabla facility. La lectura de la tabla facility se realizará con los filtros propios del sistema de CLAN (estos filtros carecen de sentido lógico y son propios de la naturaleza del sistema CLAN, por lo que los omito en esta demostración). **1 - Leemos la tabla de facility.** **SELECT** 'oficina', 'cliente', 'tramo', 'porc_total', 'fecha_operacion' **FROM** facility **WHERE** (*filtros_facility***) ** Lectura de facility con sus filtros adecuados. **2 - Cruzamos la tabla del paso 2 con la facility, resultando en la tabla tabla_paso_3.** tabla_paso_2 **LEFT JOIN** facility **ON** [tabla_paso_2 .contrato_local == facility.contrato_local **&&** tabla_paso_2.expediente == facility.expediente]
Paso 4	Calculamos, con la tabla de balance, los importes de los contratos. **1 - Leemos la tabla de balance y calculamos el agregado del saldo dispuesto, en la tabla tabla_paso_4A.** **A)** Calculamos el agregado. **SELECT** 'entidad', 'oficina', 'cliente', 'tramo', 'expediente', 'fecha_corte_blc', **SUM**('imp_dispu') **AS** 'suma' **FROM** balance **GROUPBY** ['expediente', 'entidad', 'oficina', 'tramo', 'fecha_corte_blc'] **B)** Seleccionamos solo los campos necesarios en tabla_paso_4B **SELECT** 'entidad', 'oficina', 'cliente', 'tramo', 'expediente', 'suma' **FROM** tabla_paso_4A **2 - Cruzamos la tabla balance, con los campos que nos interesan, con la tabla tabla_paso_4B, para poder calcular el importe total, en la tabla tabla_paso_4_2 .** [**SELECT** 'entidad', 'oficina', 'cliente', 'tramo', 'expediente', 'imp_dispu', 'imp_dispo', 'imp_ext', 'fecha_corte_blc', **FROM** balance] **INNER JOIN** tabla_paso_4B **ON** 'entidad', 'oficina', 'cliente', 'tramo', 'expediente' **3 - Calculamos el importe total, agregando a la tabla tabla_paso_4_2 el campo importe_total. Creamos así la tabla tabla_paso_4.** **SELECT** 'entidad', 'oficina', 'cliente', 'tramo', 'expediente', 'imp_dispu', 'imp_dispo', 'imp_ext', ('suma' **+** 'imp_dispo') **AS** 'importe_total' **FROM** tabla_paso_4_2

Paso 5	Cruzamos nuestra tabla resultante hasta ahora (tabla_paso_3) con la de balance con todos los importes necesarios (tabla_paso_4), creando la tabla tabla_paso_5. tabla_paso_3 **INNER JOIN** tabla_paso_4 **ON** ['expediente', 'entidad', 'oficina', 'cliente', 'tramo'] ** ** Debemos hacer una ventana y quedarnos con los importes MÁS RECIENTES para cada contrato.
Paso 6	Nos aseguramos de no perder contratos en el cruce anterior. **1 - Almacenamos lo contratos perdidos en el anterior cruce en la tabla tabla_paso_6_1.** tabla_paso_3 **LEFT ANTI JOIN** tabla_paso_4 **ON** ['expediente', 'entidad', 'oficina', 'cliente', 'tramo'] ** ** Agregamos los campos restantes (los que NO son campos de cruce) de la tabla tabla_paso_3 con valor NULL. **2 - Unimos los contratos que cruzaron en el paso 5 con los que no cruzaron, creando la tabla tabla_paso_6.** tabla_paso_5 **UNION** tabla_paso_6_1
Paso 7	Obtenemos, con la tabla de cliente, la relación del contrato con el grupo asociado a este, creando la tabla tabla_paso_7. tabla_paso_6 **LEFT JOIN** [**SELECT** 'cliente', 'relacion_principal', 'grupo_id' **FROM** cliente] **ON** ['cliente'] **WHERE** 'relacion_principal' **==** "Y"
Paso 8	Leemos y cruzamos con la tabla de novaciones para obtener los contratos que son novaciones. **1 - Leemos la tabla novaciones, agrupando por archivo y delta_archivo, y lo guardamos en la tabla tabla_paso_8_1.** **SELECT * FROM** novaciones **GROUPBY** ['archivo', 'delta_archivo'] **2 - Cruzamos tabla_paso_7 con tabla_paso_8_1 teniendo en cuenta todos los casos.** **A)** Los contratos que crucen y su año sea el de ODATE serán novaciones de tipo "Y". Esto queda en la tabla tabla_paso_8A. [**SELECT ***, "Y" **AS** 'tipo_novacion' **FROM** tabla_paso_7] **INNER JOIN** tabla_paso_8_1 **ON** [tabla_paso_7.expediente == tabla_paso_8_1.archivo **&&** tabla_paso_7.expediente == tabla_paso_8_1.delta_archivo] **WHERE** 'año_prod_nov' **== YEAR**(**ODATE**) **B)** Los contratos que crucen y su año NO sea el de ODATE serán novaciones de tipo "N". Esto queda en la tabla tabla_paso_8B. [**SELECT ***, "N" **AS** 'tipo_novacion' **FROM** tabla_paso_7] **INNER JOIN** tabla_paso_8_1 **ON** [tabla_paso_7.expediente == tabla_paso_8_1.archivo **&&** tabla_paso_7.expediente == tabla_paso_8_1.delta_archivo] **WHERE** 'año_prod_nov' **!= YEAR**(**ODATE**) **C)** Los contratos que NO crucen serán novaciones de tipo "N". Esto queda en la tabla tabla_paso_8C. [**SELECT ***, "N" **AS** 'tipo_novacion' **FROM** tabla_paso_7] **LEFT ANTI JOIN** tabla_paso_8_1 **ON** [tabla_paso_7.expediente == tabla_paso_8_1.archivo **&&** tabla_paso_7.expediente == tabla_paso_8_1.delta_archivo] **3 - Unimos los tres cruces anteriores en una sola tabla llamada tabla_paso_8.** (tabla_paso_8A **UNION** tabla_paso_8B) **UNION** tabla_paso_8C
Paso 9	Finalmente, seleccionamos los campos de interés y los reordenamos en la tabla tabla_paso_9. **SELECT** 'contrato', 'contrato_local', 'sistema', 'expediente', 'pais', 'entidad', 'marca_sost', 'cat_sost', 'fecha_sost', 'oficina', 'cliente', 'tramo', 'porc_total', 'fecha_operacion', 'tipo_novacion', 'imp_dispu', 'imp_dispo', 'imp_ext', 'importe_total', 'grupo_id', 'proy_id', 'fecha_proy', 'fecha_corte' **FROM** tabla_paso_8
Paso 10	La información de la tabla tabla_paso_9 se manda a una ruta específica para que el equipo de desarrollo lo ingeste en la tabla destino según su ruta (ruta_tabla_destino).

C.2. Validación de la Lógica

Pasos	Prueba	Fallo	Corrección/Explicación
Paso 2	Conteo y left anti join	Presencia de duplicados	Hacer una ventana para seleccionar el contrato más reciente en la relación código local - código global.
Paso 3	Conteo	Ninguno	El uso de left-join evita pérdida de contratos y no hay duplicados.
Paso 4	Conteo	Ninguno	No hay posibilidad de pérder contratos o duplicarlos en este paso.
Paso 5	Conteo y left anti join	Ninguno	Al aplicar la ventana se evita la presencia de duplicados.
Paso 6	Conteo y left anti join	Pérdida de contratos	Unir los contratos que no cruzaron en el paso 5.
Paso 7	Conteo	Ninguno	El uso de left join evita pérdida de contratos y no hay duplicados.
Paso 8	Conteo y left anti join	Pérdida de contratos	Unir los contratos que cruzan con los que no lo hacen (obtenidos con un left anti join).
Paso 9	PrintSchema	Mala ordenación	Reordenar las columnas según lo especificado.

Tabla C.1: Resumen de la validación sobre la lógica