



**UNIVERSIDAD  
DE LA RIOJA**

**Facultad de Ciencia y Tecnología**

## **TRABAJO FIN DE GRADO**

**Grado en Ingeniería Informática**

**BIOps: Business Intelligence como código**

Realizado por:

Miguel Orio Ruiz

Tutelado por:

Jesús María Aransay Azofra

**Logroño, Julio, 2023**



## Resumen

En este trabajo investigaremos acerca de herramientas de Business Intelligence poco populares y en desarrollo. Centrado en aquellas que trabajan con código, se elabora una comparativa.

Se explora en profundidad la más apropiada, explorando su funcionalidad e integrándose con herramientas externas, creando un producto de visualización de datos completo y adaptado.

## Abstract

This project focuses on researching some unpopular and unrecognized Business Intelligence (BI) tools that create visualizations out of code. A comparison is made choosing which one is more appropriate for a BI project.

A deeper investigation is made around the most suitable one, developing a product that enlarges the research and integrates it with external applications.



# Índice

<b>Resumen.....</b>	<b>3</b>
<b>Abstract.....</b>	<b>3</b>
<b>Índice.....</b>	<b>5</b>
<b>Capítulo 0: Introducción.....</b>	<b>7</b>
Planteamiento.....	7
Antecedentes.....	7
<b>Capítulo 1: Planificación.....</b>	<b>8</b>
1.1. Actores.....	8
1.2. Metodología.....	8
1.3. Alcance.....	9
1.3.1 Requisitos funcionales.....	9
1.3.2 Requisitos no funcionales.....	9
1.3.3 Exclusiones.....	10
1.3.4 Entregables.....	10
1.4. Estructura de descomposición del proyecto (EDP).....	11
1.5. Planificación temporal.....	11
1.5.1. Diagrama de Gantt.....	12
1.5.2. Diagrama de hitos.....	13
1.6. Riesgos.....	13
<b>Capítulo 2: Sprint 1. Comparativa entre herramientas de BI sobre código.....</b>	<b>14</b>
2.1. Requisitos del Sprint 1.....	14
2.2.1. Graphext.....	15
2.2.2. Evidence.....	16
2.2.3. Mprove.....	17
2.3. Definición de criterios.....	18
2.4. Comparativa de herramientas.....	19
<b>Capítulo 3: Sprint 2. Desarrollo de un programa de BI.....</b>	<b>22</b>
3.1. Preparación del entorno de Snowflake.....	22
3.2. Requisitos del proyecto de Evidence.....	23
3.3. Estructura del proyecto de Evidence.....	24

<b>Capítulo 4: Sprint 3. Adaptación de formato.....</b>	<b>28</b>
4.0. Introducción sobre el Sprint 3.....	28
4.1. Análisis del E-302.....	29
4.2. El transformador de Markdown en Python.....	29
4.2.1 Conocimientos de Python requeridos.....	29
4.2.2 Análisis del transformador de Markdown.....	31
4.2.3 Diseño del transformador de Markdown.....	31
4.2.4 Implementación del transformador de Markdown.....	32
4.3. El potencial de Jinja.....	34
4.4. El problema de las variables de entorno.....	36
<b>Capítulo 5: Sprint 3. Control de estado de la BD.....</b>	<b>37</b>
5.1. Traslado a una BD multitabla.....	37
5.2. DBT como herramienta.....	38
5.2.1. Uso y sintaxis de DBT.....	38
<b>Capítulo 6: Sprint 3. Orquestación.....</b>	<b>40</b>
6.1. Airflow, DAGs y operadores.....	40
6.2. Volúmenes y Docker Compose.....	41
<b>Capítulo 7: Estructura global del proyecto.....</b>	<b>42</b>
<b>Capítulo 8: Revisión de la planificación.....</b>	<b>43</b>
8.1. Revisión de requisitos.....	43
8.1. Revisión temporal.....	43
<b>Capítulo 9: Conclusiones del trabajo.....</b>	<b>46</b>
9.1. Conclusiones del proyecto.....	46
9.2. Conclusiones personales.....	46
<b>Bibliografía.....</b>	<b>47</b>
<b>Anexo I: Código del generador Python.....</b>	<b>48</b>
<b>Anexo II: Código del DAG de ejecución principal.....</b>	<b>49</b>

# Capítulo 0: Introducción

## Planteamiento

El Business Intelligence (BI) es un conjunto de técnicas que facilitan a las empresas tener conocimiento sobre su estado, su situación, su progresión o su toma de decisiones.

Aplicado a herramientas, hablaremos de aquellas que apoyen la extracción de información y conclusiones, como modificaciones y transformaciones de tablas en una BD o aplicaciones de visualización de datos.

En nuestro caso queremos desarrollar una aplicación que se enfoque en la visualización de datos y la generación de informes o *dashboards*.

Las principales aplicaciones usadas en la empresa para estos fines son de formato *drag and drop*, como PowerBI o Qlik. Sin embargo, la propuesta de este trabajo pasa por el desarrollo de una de estas aplicaciones cuyo formato esté basado en código.

Este trabajo consiste en comparar tecnologías y elegir una que permita realizar un trabajo de Business Intelligence de forma adecuada. Una vez elegida, desarrollar un programa de BI basado en código, que aproveche las virtudes de otros programas que también utilicen código, como la maleabilidad, la transparencia, la portabilidad o el control de versiones.

## Antecedentes

Durante mi estancia en la empresa durante prácticas externas, usé distintas herramientas para visualizaciones de datos y generación de dashboards. Las tres herramientas que utilicé son Evidence, Graphext y Mprove.

Evidence es una herramienta que genera informes en formato de página web.

Graphext es una herramienta para navegador que genera visualizaciones puramente gráficas.

Mprove es una herramienta web que genera visualizaciones en una interfaz web propia y se especializa en el control de versiones.

Ahondaremos más en estas herramientas cuando tengamos que elegir cual de ellas es más conveniente para nuestro trabajo

# Capítulo 1: Planificación

La planificación de un proyecto es la base de su solidez. Durante este capítulo forjaremos una base fuerte sobre la que apoyarnos a lo largo del trabajo. Para ello definiremos la metodología a utilizar, los límites sobre el alcance, sobre los tiempos y otros factores a tener en cuenta.

## 1.1. Actores

- **Miguel Orio**  
Responsable de la autoría del proyecto y del desarrollo del programa, así como de la memoria y la defensa del trabajo.
- **Xabier Gil**  
Cliente del proyecto. Encargado de proporcionar requisitos, así como ayuda para la implementación de funcionalidades adicionales.
- **María Martínez**  
Cliente del proyecto. Encargada de la formación en tecnologías y ayuda para el seguimiento y control del proyecto.
- **Jesus María Aransay**  
Tutor académico y de prácticas de la Universidad de La Rioja. Guía para el desarrollo de la memoria del proyecto.

## 1.2. Metodología

El trabajo que queremos realizar está dividido en iteraciones, por lo que adaptaremos nuestro trabajo a una metodología ágil que funcione mediante sprints de funcionalidad.

Para el seguimiento y control de cada uno de esos sprints utilizaremos un modelo *Kanban* de cuadro de tareas, coordinado mediante la aplicación Jira.<sup>1</sup>

En nuestro caso, el proyecto consta de 3 sprints:

**Sprint 1 (S1):** En este sprint se realiza una comparativa entre herramientas para elegir las tecnologías del proyecto.

**Sprint 2:(S2):** En este sprint se genera un artefacto de BI algo elaborado con la herramienta elegida.

**Sprint 3 (S3):** En este sprint se utilizan herramientas externas sobre el artefacto para darle funcionalidad adicional

---

<sup>1</sup>  **Jira**

Jira es una herramienta de Atlassian para la gestión de proyectos y el seguimiento de tareas, incidencias y errores.



## 1.3. Alcance

En este punto delimitaremos el proyecto en base a los requerimientos del cliente respecto a la totalidad del trabajo y al producto final. Los requisitos que sean necesarios dentro de cada sprint los trataremos en su capítulo correspondiente.

### 1.3.1 Requisitos funcionales

Tabla 1.1 - Requisitos funcionales del proyecto	
Código	Descripción
RF-001	El producto final será una herramienta de Business Intelligence que permita la visualización de datos.
RF-002	El producto final tendrá código visible, accesible y modificable.
RF-003	El producto final será capaz de sincronizarse de alguna forma con datos almacenados en el SGBD de Snowflake. <sup>2</sup>
RF-004	El producto final contará con un dashboard de visualización sobre un conjunto de datos existente.
RF-005	El producto final contará con opciones de portabilidad.
RF-006	El producto final soportará inserciones de metadatos.

### 1.3.2 Requisitos no funcionales

Tabla 1.2 - Requisitos no funcionales del proyecto	
Código	Descripción
RNF-001	Se elegirá una tecnología de BI con código que sea adecuada.
RNF-002	El producto final se realizará con la herramienta seleccionada.
RNF-003	La base de datos tendrá mantenimiento y escalabilidad
RNF-004	El código se controlará con un sistema de control de versiones
RNF-005	Documentación del código generado
RNF-006	Optimización del código generado

---

<sup>2</sup>  **Snowflake**

Snowflake es una aplicación en la nube de almacenamiento y procesamiento de datos. Tiene un servicio de red global, una interfaz web y todas las funciones de un SGBD, como ejecuciones de scripts, no solo en SQL.

### 1.3.3 Exclusiones

En la Tabla 1.3 se explican aquellas partes del proyecto que quedan fuera del alcance, ya sea por falta de tiempo o relevancia. Sin embargo no dejan de ser posibles funcionalidades adicionales que sería interesante tener abiertas a tratar.

Tabla 1.3 - Exclusiones del proyecto	
Código	Descripción
EX-001	No se migrará el producto realizado a las herramientas descartadas en S1
EX-002	No se realizarán modificaciones al front-end por defecto que tengan las aplicaciones utilizadas en S1
EX-003	No se tratará, a menos de que falle la funcionalidad, de adaptar el programa a otra base de datos aparte de Snowflake

### 1.3.4 Entregables

Los entregables reflejan los productos generados del proyecto. Su plazo de desarrollo está delimitado por el diagrama de hitos (Tabla 1.6)

- **E-101:** Comparativa en profundidad de las herramientas Evidence, Mprove y Graphext aplicado al contexto que buscamos (funcionalidad, conexiones con otras herramientas, adaptación al trabajo con código).
- **E-201:** Modelo funcional de la herramienta elegida entre las del E-101 que desarrolle en anchura la funcionalidad de la herramienta.
- **E-301:** Sistema de ejecución automática, dinámica y periódica de la herramienta elegida en E-101 mediante funcionalidad externa.

El entregable E-201 está formado por el conjunto de otros tres entregables con funcionalidad y entregas independientes que se desarrollan por separado: E-302, E-303 y E-304.

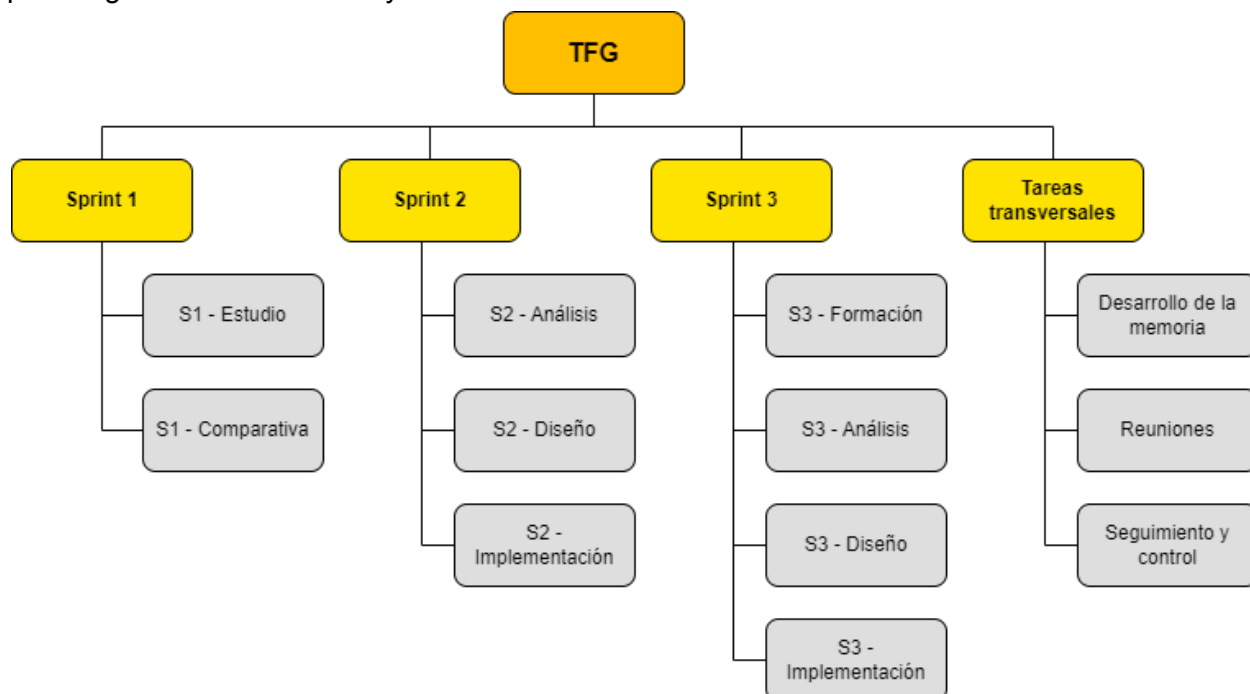
- **E-302:** Sistema de adaptación de archivos para un formato compatible para la herramienta utilizada en el E-201.
- **E-303:** Modelo de transformación de base de datos mediante DBT.
- **E-304:** Sistema de flujo de trabajo en un orquestador que permita de forma periódica sincronizar la herramienta con la funcionalidad que cambie en el resto del sistema y actualice la base de datos en el proceso.
- **E-401:** Memoria de prácticas
- **E-402:** Presentación de la defensa

## 1.4. Estructura de descomposición del proyecto (EDP)

La estructura de descomposición del proyecto (EDP) o estructura de descomposición de tareas (EDT) nos sirve para mostrar de forma esquemática cómo se dividen los entregables del trabajo.

Las tareas del sprint 1 corresponden al desarrollo del entregable E-101, las tareas del sprint 2 corresponden al desarrollo del E-201 y las tareas del sprint 3 corresponden al E-301.

Las tareas transversales son aquellas que se desarrollan a lo largo de todo el proyecto, y sirven para la generación de E-401 y E-402.



## 1.5. Planificación temporal

Para la realización de este proyecto contamos con algo más de 17 semanas. Sin embargo, dado que dedicaremos, en la mayoría de los casos, 25 horas a la semana, el proyecto concluirá algo antes de este plazo.

La semana 1 de trabajo corresponde a la cuarta semana de Enero, desde el 16 hasta el 22. Enero abarca las semanas 1 a 3, Febrero 3 a 7, Marzo 7 a 11, Abril 11 a 15 y Mayo 16 a 18.

Las semanas 25 (3 a 9 de Julio) y 27 (17 a 23 de Julio) corresponden a las fechas establecidas para el depósito y la defensa del Trabajo de Fin de Grado, respectivamente.

### 1.5.1. Diagrama de Gantt

El diagrama de Gantt refleja los tiempos que están planificados inicialmente para el desarrollo de las tareas de la EDP, así como el depósito de la memoria y la defensa del trabajo, cuyos plazos vienen delimitados por la Universidad de La Rioja.

A lo largo del trabajo se hará un seguimiento y control para establecer en qué medida se ha seguido este diagrama inicial.

Tarea	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
S0 - Estudio y comparativa										
S1 - Análisis y diseño										
S1 - Implementación										
S2 - Formación										
S2 - Análisis y diseño										
S2 - Implementación										
Desarrollo de la memoria										
Seguimiento y control										

### 1.5.2. Diagrama de hitos

Tabla 1.6 - Diagrama de hitos																
Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	25	27
E-101	◆															
E-201			◆													
E-301														◆		
E-302								◆								
E-303												◆				
E-304													◆			
E-401															◆	
E-402																◆

### 1.6. Riesgos

Tabla 1.7 - Plan de riesgos	
Riesgo	Solución
Cambios en el alcance del proyecto	Dedicar tiempo suficiente a realizar un análisis que defina de forma precisa el alcance para evitar la necesidad de hacer cambios mayores una vez avanzado el proyecto.
Pérdida de datos o problema con la base de datos.	Todos los scripts SQL de creación de BD, tablas y vistas, así como los de poblado de tablas e inserción de filas, quedarán almacenados como copia de seguridad.
Pérdida de código desarrollado	Se almacenará el código en el sistema de control de versiones de Bitbucket <sup>3</sup> .
Baja por enfermedad	Se distribuirán las horas de trabajo perdidas a lo largo de las semanas 14-17.

<sup>3</sup>  **Bitbucket**

Bitbucket es una herramienta de Atlassian para el control de versiones. Es un servicio de alojamiento web compatible con sistemas Git y Mercurial.

## Capítulo 2: Sprint 1. Comparativa entre herramientas de BI sobre código

Durante este capítulo abordaremos todo el contenido relacionado con el Sprint 0, el cual se divide en dos partes.

En primera instancia, elaboraremos un estudio de las tres herramientas vistas en prácticas, de cara a sacar conclusiones sobre en qué casos de uso es efectivo usar cada una de ellas. Para ello haremos una demo de funcionalidad mínima y estudiaremos su funcionamiento.

Para la segunda parte, primero estableceremos los criterios bajo los que evaluaremos cada herramienta, para luego establecer una tabla comparativa de las tres herramientas en base a esos criterios. Esta tabla corresponde al entregable e hito E-101. Lo último será decidir qué herramienta elegiremos para desarrollarla en profundidad durante el Sprint 1.

### 2.1. Requisitos del Sprint 1

Tabla 2.1 - Requisitos del producto E-101	
Requisitos funcionales	
Código	Descripción
RF-101	La tabla comparativa incluirá los criterios establecidos
RF-102	La tabla comparativa será utilizada para elegir una herramienta para su estudio en profundidad.
Requisitos no funcionales	
Código	Descripción
RNF-101	La tabla comparativa tendrá un código de colores que establezca si sus características son buenas, malas o neutras.
RNF-102	Las demos no necesitan tener ningún tipo de persistencia ni continuidad. Son independientes a lo que se realizará en S1.
RNF-103	Las demos tratarán de realizar diferentes tipos de gráficos de visualización de datos.
RNF-104	Durante las demos se intentarán conexiones con Snowflake y Bitbucket, tratando de seguir RF-003 y RNF-004

## 2.2. Prueba de herramientas

He realizado una demo simple de las herramientas Graphext, Evidence y Mprove para poder identificar las características, fortalezas y flaquezas de cada una de ellas.

### 2.2.1. Graphext

Graphext <sup>4</sup> es un programa con un funcionamiento simple: tras una ingesta de una tabla, el programa desarrolla en diferentes pestañas diferentes opciones de visualización.

La pestaña principal muestra la tabla y un gráfico de distribución de cada columna.

Las pestañas *Plot*, *Compare* y *Correlations* nos permite generar referencias visuales de una columna sobre otra, de una columna sobre todas las demás o de dos columnas entre sí, respectivamente, y guardar los resultados que queramos como *Insights*.

Entre sus ventajas se encuentran la simplicidad y portabilidad de funcionar en línea y sus múltiples integraciones, tanto BD (Postgre,MySQL,Azure,SQL server, Google Drive..) como API keys (como Google NLP o Twitter) o ingestas directas (Google sheets o csv).

A pesar de ello tiene enormes desventajas, como que sus gráficos no tienen ningún tipo de personalización, no permite generar Dashboards, no tiene acceso a control de versiones (rompiendo el RNF-004), no se conecta con Snowflake (rompiendo el RF-003) y al contactar con soporte no consiguió solucionar problemas de configuración al importar desde Postgre.

Pero aun así el principal problema es que es una herramienta no-code cuyo código no es accesible,siendo lo opuesto al propósito de este trabajo e imposible de utilizar para ello.

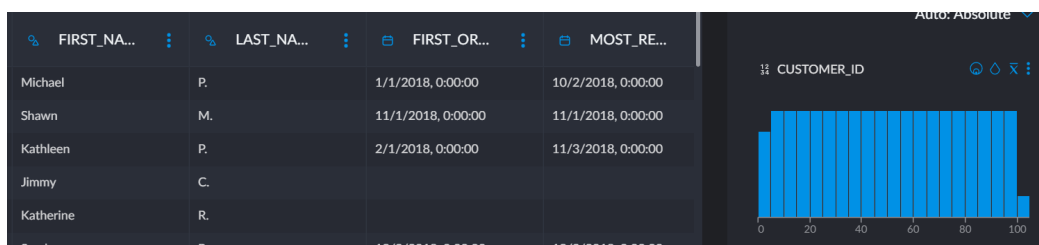


Figura 2.1: Pestaña principal de una tabla en Graphext.



Figura 2.2: Pestaña Correlations realizando comparativas entre dos columnas

---

<sup>4</sup>  **Graphext**

Graphext es una aplicación web no-code para Business Intelligence y Machine Learning. Permite la visualización rápida de gráficos para análisis de datos no estructurado.

## 2.2.2. Evidence

Evidence <sup>5</sup> es un programa de Business Intelligence con una arquitectura en dos capas: La primera está abusada en documentos Markdown <sup>6</sup> que leen datos con SQL contra una Data Warehouse y la segunda es una aplicación web que muestra los resultados.

La instalación de Evidence en local es simple, tan solo requiere de una versión de node.js (v14 o superior) y una versión de npm (v7 o superior). Para instalarlo se ejecutan 2 comandos de creación de proyecto y se hace funcionar con `npm run dev`.

La principal ventaja de Evidence es su total capacidad de customización. Aquello que escribes en el código markdown es aquello que se plasma en la aplicación web. Asimismo tienes bloques SQL en el código que funcionan exactamente igual que en Snowflake.

Entre las desventajas que tiene, la instalación con Docker no funciona por la imagen de Evidence, el estilo de la aplicación web se genera en un front llamado Svelte que es difícil de modificar y al ser un programa pequeño apenas hay soporte y solo se puede hacer mediante contacto en Slack o issues de Github.

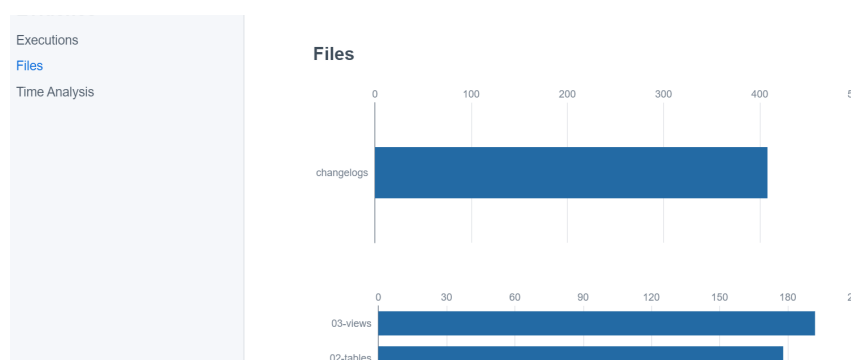


Figura 2.3: Visualización de la aplicación web de Evidence.

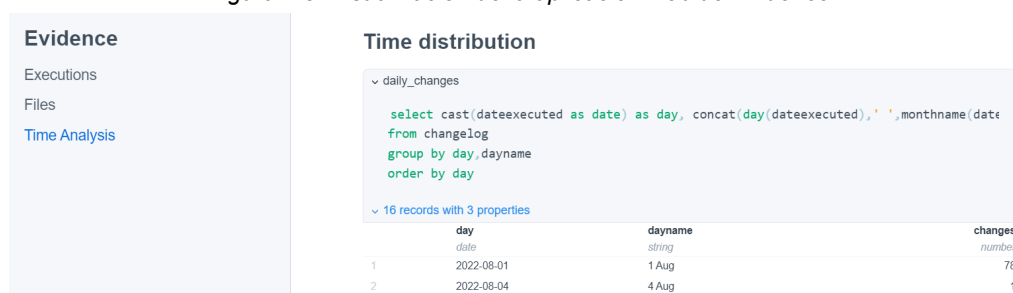


Figura 2.4: Uso de bloques SQL en una página de Evidence.

### 5 **E** Evidence

Evidence es una herramienta para generar documentos web orientados a Business Intelligence a partir de documentos Markdown con anotaciones especiales que gestionan datos.

### 6 **M↓** Markdown

Markdown es un lenguaje simple de marcado para texto plano que modifica el formato del documento para darle opciones adicionales de legibilidad y conversión fácil a HTML.



### 2.2.3. Mprove

Mprove <sup>7</sup> es una herramienta Open Source de Business Intelligence especializada en control de versiones. Está basada en modelos SQL escritos en YAML <sup>8</sup>, un lenguaje de serialización.

La principal ventaja de mprove es su flexibilidad de dimensiones y su comodidad para hacer acciones multitabla entre ellas. Las dimensiones se definen como columnas de una tabla o transformaciones sobre archivos YAML. El código es transparente en estos archivos.

Su principal desventaja es su nula portabilidad (RF-005), ya que los archivos no son accesibles fácilmente y son de un lenguaje muy concreto. Asimismo, pese a estar integrado con Git, si decides hacer un proyecto en local asigna su propio control de versiones, y no permite migrarlo.

La instalación se puede hacer tanto con Kubernetes como con Docker Compose <sup>9</sup> (nuestro caso). Tanto para la instalación como para el uso del programa se recomienda seguir guías dado que la estructura YAML que necesita Mprove es muy concreta y el programa en sí funciona con sus propias herramientas, como la interfaz web o un sistema *drag and drop* de creación de dashboards.

Para generar Dashboards requiere crear una organización, un proyecto, una conexión, archivos de modelos, archivos de vistas y luego crear los elementos del dashboard mediante un sistema de código poco transparente de combinación de tablas.

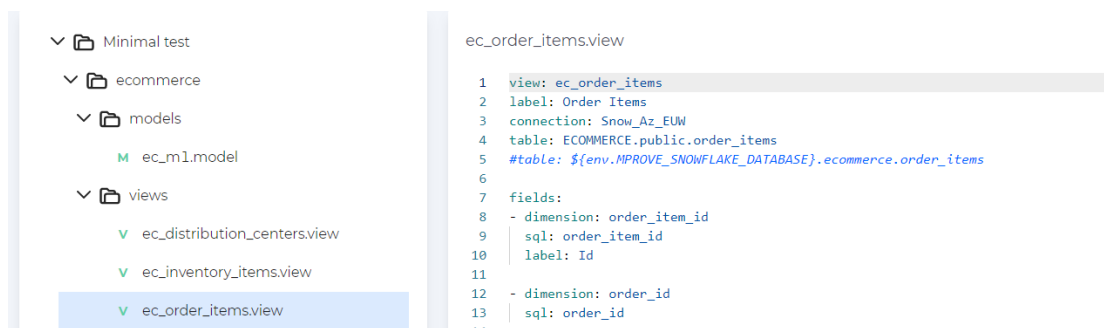


Figura 2.5: Uso de lenguaje YAML para definir dimensiones en Mprove.

### 7 Mprove

Mprove es una herramienta de BI pensada para DataOps, a través de crear modelos con SQL y BlockML, visualizaciones y dashboards con un fuerte peso de control de versiones.

### 8 YAML

YAML es un lenguaje de serialización de datos generalmente utilizado para archivos de configuración. Es muy útil para la ingesta o el movimiento de datos.

### 9 Docker

Docker es un proyecto open source que despliega aplicaciones mediante contenedores de software. Es una forma para tener aplicaciones portátiles y setups sencillas sin instalación. Docker compose es el despliegue de un conjunto de varios de estos contenedores en uno.

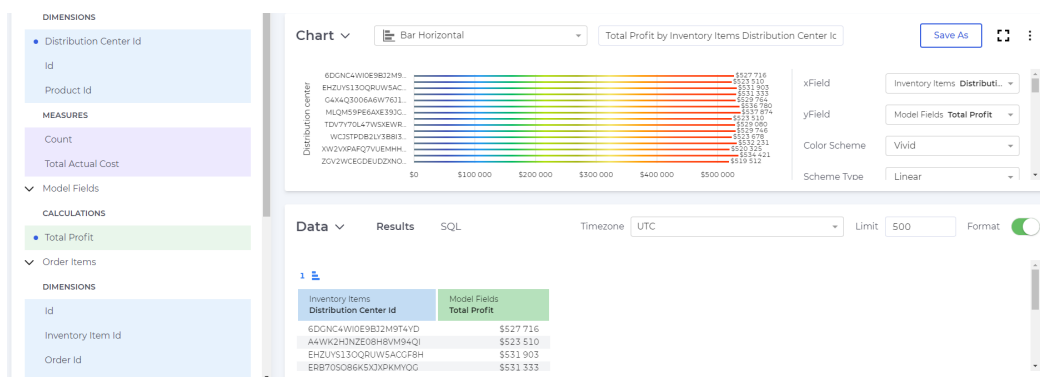


Figura 2.6: Uso de las dimensiones para generar un BarChart.

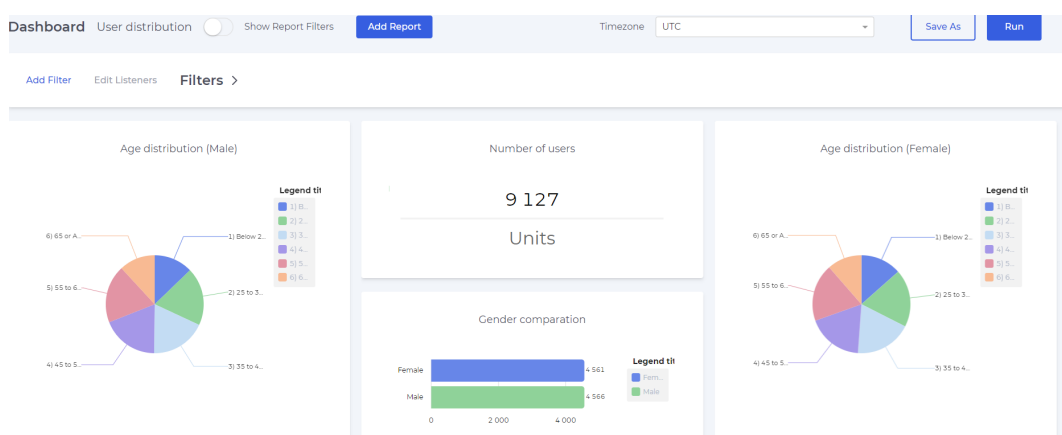


Figura 2.7: Ejemplo de un dashboard en el entorno de Mprove.

## 2.3. Definición de criterios

A continuación, basado en lo que necesitamos y en lo que hemos ido experimentando, delimitaremos los criterios a partir de los cuales evaluar las herramientas anteriores.

Estos serán:

- Instalación y setup sencilla y rápida.
- Portabilidad con docker
- Código visible y simple
- Conexión con Snowflake
- Posibilidad de control de versiones con Git
- Opciones de generación de gráficos y dashboards
- Soporte y documentación
- Herramientas adicionales

Aparte de estos criterios, también se tendrá en cuenta la Quality of Life de cada herramienta, cómo de cómoda es, cómo de profesional se siente, cuánto potencial se ve de cada herramienta...

## 2.4. Comparativa de herramientas

Tabla 2.2 - Comparativa de herramientas de BI			
	Graphext	Evidence	Mprove
Orientación principal	Machine learning y Data Science	Presentación de datos	Visualización de datos
¿Es Open Source?	Tiene versión gratuita	Es Open Source	Es Open Source
Formato	Aplicación web	Página web	Aplicación web
Instalación y setup			
Sin requisitos previos	✓	✗ NPM y Node.js	✗ Docker o Kubernetes
Instalación sencilla	✓	✓	✓
Instalación rápida	✓	✓	✓
Setup sencilla	✓	✓	✗
Portabilidad			
Portable por defecto	✓	✗	✗
Portable con Docker	-	✗ Tiene Docker pero no funciona la imagen.	✓
Alternativas a Docker	-	✗	✓ Kubernetes


































Tabla 2.3 - Comparativa de herramientas de BI (2)			
	Graphext	Evidence	Mprove
Código			
¿Todo el código es visible?	 No visible		 Parcialmente visible
Lenguaje	-	Markdown anotado	YAML
Librerías externas		 Para gráficos JS	 Para datasets
Conexión con Bases de Datos			
Acepta archivos csv			
Acepta conexión con Snowflake			
Tiene conexiones con otras BD	 BigQuery,SQLserver,S3 Postgre,Azure,MySQL.	 BigQuery, Postgre.	 BigQuery, Postgre, Clickhouse.
Control de versiones			
Acceso a Git			
Sincronización directa con Github			
Control local de versiones			
Gráficos y dashboards			
Gráficos en n variables			
Gráficos preprogramados			
Genera dashboards			

Tabla 2.4 - Comparativa de herramientas de BI (3)			
	Graphext	Evidence	Mprove
Soporte y documentación			
Buen soporte	✓	✗	✗
Documentación amplia	✗	✗	✗
Documentación de calidad	✗	✓	✓
Funcionalidad adicional			
¿Qué funcionalidad adicional tiene?	Foro abierto.	Extensión de VScode. Bloques de código propios.	Validación instantánea de datos de dashboards. Sistema de roles. Filtros en dashboard.

Dado que es una herramienta no-code, Graphext rompe la idea de este trabajo por lo que está descartado. De todas formas, sigue siendo una herramienta, que para mí, tiene como caso de uso principal generar gráficos simples de una tabla de la forma más rápida posible para identificar cuáles de ellos aportan información más relevante.

Tanto Evidence como Mprove son herramientas que generan dashboards a partir de datos.

Mprove es más apropiado para sistemas que tienen dificultades de instalación y trabajan con docker. También considero que ofrece resultados más profesionales y en general ofrece un producto más estético. Sin embargo, no me resulta cómodo que sea una herramienta burbuja, que los archivos tengas que escribirlos en la propia aplicación como modo principal de inserción de datos, y sobre todo que no genere gráficos cuyo código no sea visible.

Considero que Evidence es la mejor opción por dos principales razones:

La primera es su transparencia, ya que todo el código escrito en tus archivos markdown se plasma literalmente al desplegar el programa.

La segunda es el potencial que considero que tiene la herramienta. Al ser un programa que lee anotaciones de markdown y permite cosas como bucles o resultados almacenados de sentencias sql, considero que se pueden explorar estos campos y sacar todo lo posible de la herramienta

## Capítulo 3: Sprint 2. Desarrollo de un programa de BI

### 3.1. Preparación del entorno de Snowflake

El archivo de origen con el que contaremos es un .csv, así que nuestra primera misión es importarlo a Snowflake.

Mediante SnowSQL (línea de comandos de snowflake), es posible crear stages, una estructura donde se almacenan datos para insertarlos en tablas. A través de ellos se pueden importar datos desde un .csv <sup>10</sup> hasta Snowflake.

La estructura del .csv generó algo de problema con las strings que incluyen comas. Por ejemplo, imaginemos una línea con la siguiente estructura:

*campo1,campo2,"campo3,string",campo4,campo5*

El campo 3 supone un problema de formato, ya que el lector de formatos de Snowflake tiene una propiedad *FIELD\_OPTIONALLY\_ENCLOSED\_BY=' "* ' que permite leer los campos entrecomillados. Para que Snowflake pueda leer este tipo de cadenas con esta propiedad, todos los campos deben estar entre comillas.

A pesar de poder hacerse de forma manual y sencilla, prefiero tener una forma escalable de poder realizar estas transformaciones.

Tras desarrollar un programa simple en Java que transforme este .csv en una estructura

*"campo1","campo2", "campo3,string","campo4","campo5"*

Snowflake es capaz de importarlo y, usando una warehouse "WHOUSE", una base de datos "LIQUIBASE\_DATA" y un schema "PUBLIC", genera la tabla "CHANGELOG" (figura 8)

ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECUTED	EXECTYPE	MD5SUM
DESCRIPTION	COMMENTS	TAG	LIQUIBASE	CONTEXT	LABELS	DEPLOYMENT_ID

Figura 3.1: Campos de la tabla CHANGELOG

En los siguientes apartados ahondaremos en algunos de estos campos, pero de momento no necesitamos más detalle

---

<sup>10</sup>  **CSV**

Comma Separated Values (csv) es un tipo de documento para la organización de datos.

## 3.2. Requisitos del proyecto de Evidence

Ahora que tenemos el entorno Snowflake que nos permite tener un conjunto de datos manejable, necesitamos ver qué funcionalidades ofrece Evidence para poder tratarlos.

Para ello, debemos explorar las opciones que ofrece Evidence en su documentación.

- Evidence permite ver resultados de consulta directamente al almacenar una consulta para su uso en una página, y también puede formatear como tabla.
- Evidence tiene con sintaxis sencilla algunos gráficos como los de barras, líneas, área, burbujas, dispersión o histogramas.
- Evidence permite hacer gráficos mixtos, combinando varios de los anteriores.
- Cuenta con la posibilidad de leer gráficos de Echarts, una librería externa de JavaScript.
- Evidence permite crear bloques condicionales y bucles foreach.
- Estructuración del programa en un árbol de páginas

Basado en ello vamos a analizar el alcance de nuestro programa en Evidence (E-101):

Tabla 3.1 - Tareas a realizar en el E-201	
Código	Descripción
T-201	Alguna página debe tener una tabla de datos visible
T-202	Alguna página debe incluir gráficos básicos
T-203	Alguna página debe incluir un gráfico mixto
T-204	Alguna página debe incluir un gráfico de Echarts
T-205	Alguna página debe incluir una estructura que combine bucles y bloques condicionales.
T-206	El proyecto en Evidence debe tener una paginación lógica.
T-207	El proyecto de Evidence deberá estar conectado a Snowflake y obtener de ahí todos sus datos
T-208	El markdown del proyecto debe estar estructurado, ser coherente y utilizar herramientas propias como titulado
T-209	El código realizado se irá subiendo a Bitbucket para su control de versiones

### 3.3. Estructura del proyecto de Evidence

Al abrir el proyecto Evidence lo primero que se ejecuta es un archivo *index.md*. Desde este fichero aprovecharemos para mostrar algo de información general y poner en él hipervínculos para poder navegar a los archivos de visualización de campos.

Los campos que he elegido para ello son aquellos que más capaces son de tener alguna agrupación: Author, filename, dateexecuted y exectype y los desarrollaremos más en la siguiente estructura del E-201 (cumpliendo T-206)

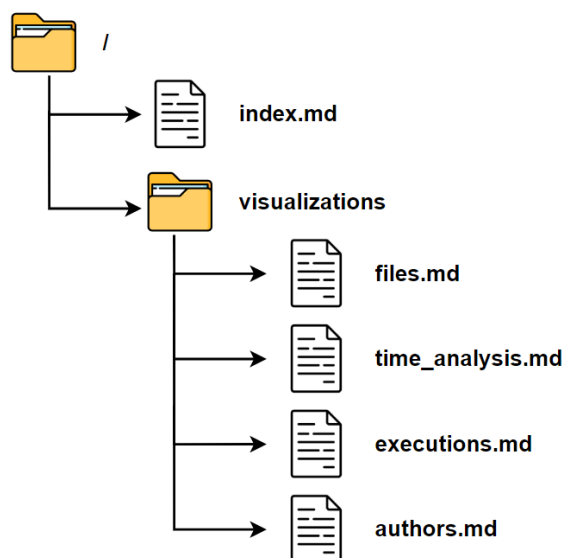


Figura 3.2: Estructura del proyecto Evidence

El primer fichero a desarrollar es *files.md*.

Para este fichero utilizaremos bloques de código sencillos, gráficos simples (T-202) y una DataTable (T-201).

La forma en la que se declara un bloque sql como resultset para su uso en otras funciones es con anotaciones ```. top\_files hace referencia a los 15 ficheros que más cambios han recibido:

```

```top_files
select top 15 filename as file, count(*) as amount
from changelog
group by file
order by count(*) desc
```

```

Este tipo de bloques sirven para ser utilizados en otros elementos, como valores o gráficos Evidence.

```

<DataTable data={top_files}/>
<BarChart data={top_files} x=file y=amount swapXY=true/>

```



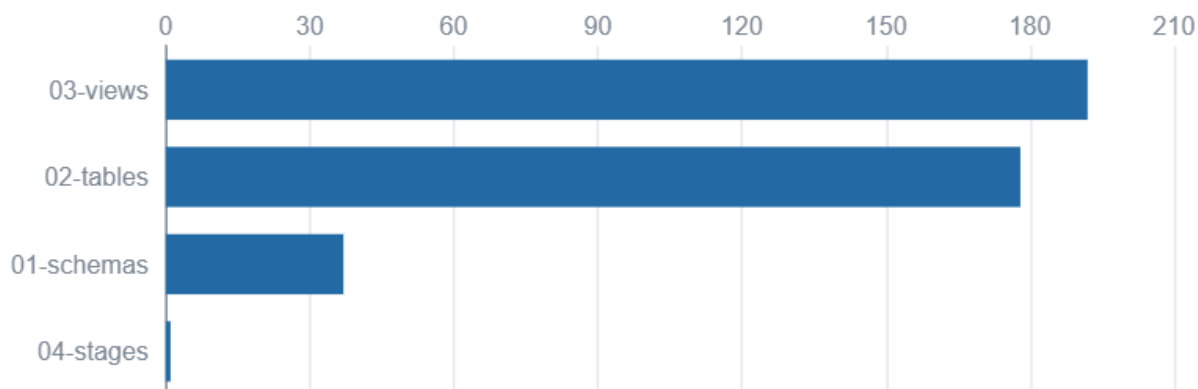


Figura 3.2: Gráfico de barras en Evidence

Con estas nociones desarrollaremos el resto de ficheros:

En *time\_analysis.md* ahondamos un poco más en los gráficos Evidence, y conseguimos generar un “gráfico mixto” (T-203).

```
<Chart data={daily_changes_accum}>
  <Bar x=day y=amount name="daily changes"/>
  <Line x=day y=sumamount name="total changes"/>
</Chart>
```

Por ejemplo, en el caso aquí expuesto tenemos un gráfico de barras que muestra la cantidad de cambios que se ejecutan cada día, y un gráfico de líneas que muestra de forma incremental la cantidad de cambios que ha habido en el histórico de tiempo.

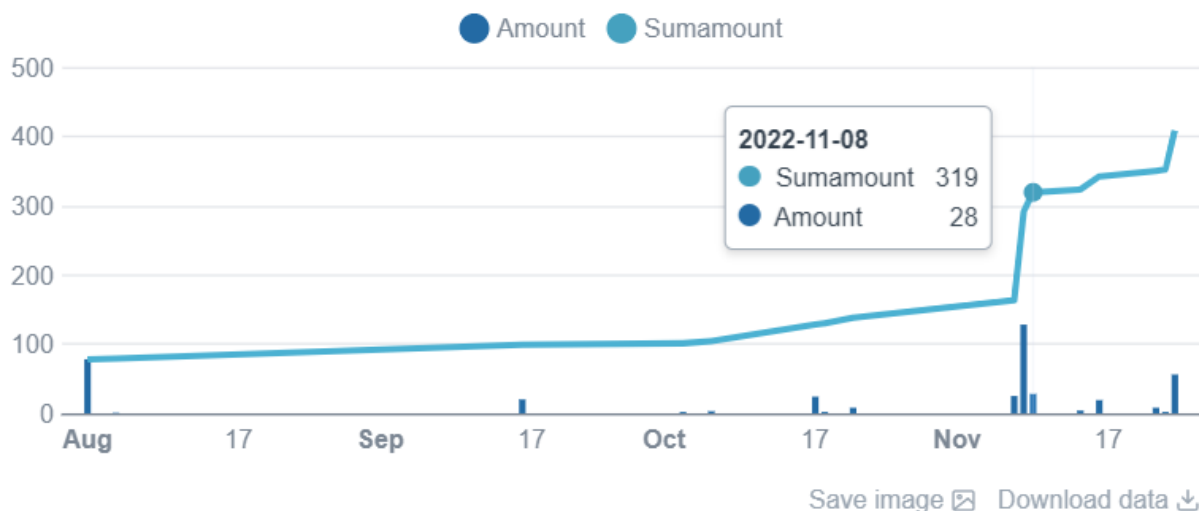


Figura 3.3: Gráfico mixto en Evidence

En *executions.md* vamos a tratar los condicionales y bucles de Evidence (T-205):

```
{#each error_details as error_row}
Error at file <Value data={error_row} column=filename/> <br>
Error date: <Value data={error_row} column=dateexecuted/> <br>
{#if error_row.md5sum==""}
This error has no MD5 value
{:else}
For this error, MD5 value is <Value data={error_row} column=md5sum/>
{/if}
{/each}
```

En este ejemplo hemos tratado las últimas filas por fichero cuyo valor “executed” no es EXECUTED (es decir, las últimas ejecuciones pospuestas o fallidas de cada fichero que cause errores). Para seguir la traza se muestran todas en el fichero de visualización, y en el caso de que lo tengan se mostrará su hash MD5.

Para ello hemos utilizado tanto bucles each cómo condicionales. Estas herramientas sin embargo tienen muchas limitaciones, ya que Markdown+Evidence no tiene el concepto de variable. Las consultas SQL tendrían que ser muy meticulosamente medidas para hacer de forma precisa aquello que queremos.

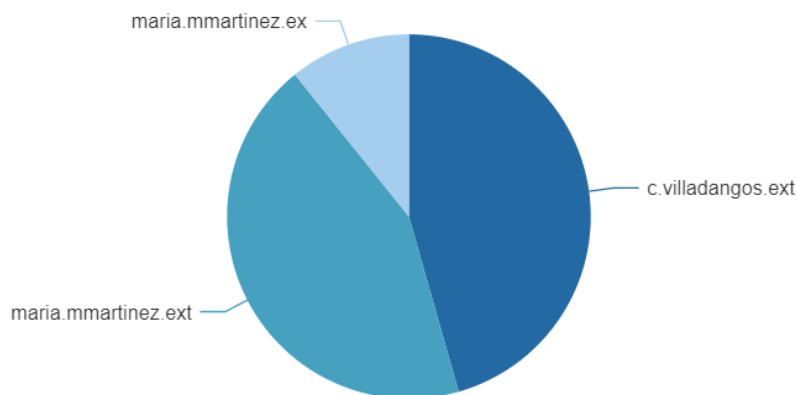
Por último, en *authors.md* desarrollamos la estructura de la librería Echarts (T-204), una herramienta externa de generación de gráficos. La estructura consta de 3 partes: Los datos origen, expresados explícitamente, una transformación y un bloque que genera el Echart.

```
```pie_query
select 'c.villadangos.ext' as pie, 186 as count
union all
select 'maria.mmartinez.ext' as pie, 178 as count
union all
select 'maria.mmartinez.ex' as pie, 44 as count
```
```

```
```pie_data
select pie as name, count as value
from ${pie_query}
```
```

```
<ECharts config={
  {
    tooltip: {
      formatter: '{b}: {c} ({d}%)'
    },
    series: [
      {
        type: 'pie',
        data: pie_data,
      }
    ]
  }
}>
```

Esta es la forma que tiene Evidence para generar un gráfico de sectores, ya que no existen en la aplicación por sí misma, sino que deben ser generados mediante esta librería externa.



*Figura 3.4: Gráfico de sectores en Evidence, mediante ECharts*

Aquí encontramos otro problema claro. Los datos que aquí se están introduciendo son estáticos. Y tampoco se pueden generar dinámicamente porque los bloques each no pueden detectar el número de filas o el fin de fichero para dejar de escribir la sintaxis con “union all”.

## 3.4. Conclusiones del Sprint 2

Snowflake ha resultado ser una herramienta muy útil, muy cómoda y muy potente sobre la que establecer una base de datos.

Aunque más manual y directo, la versión de Markdown y Evidence resulta lo suficientemente cómoda y comprensible para hacer algo transparente, como piden los requisitos.

Sin embargo estas herramientas presentan limitaciones: no parece que se pueda innovar demasiado o hacer algo demasiado potente en este ámbito.

Un problema molesto es la necesidad de una librería externa como Echarts para generar gráficos muy básicos y necesarios.

Por otra parte, trabajar sobre datos reales, generados por mis compañeros, en un caso de uso es algo que me ha sido enriquecedor.

## Capítulo 4: Sprint 3. Adaptación de formato.

### 4.0. Introducción sobre el Sprint 3

Entrando al Sprint 3, desarrollaremos el diseño y los objetivos del entregable E-301

El entregable E-301 es un sistema que adapta lo realizado en el Sprint 2 y lo adapta a herramientas externas que son utilizadas en la empresa, como DBT y Airflow.

El entregable E-301 es un “lavado de cara” de lo mostrado en el E-201, buscando mejoras de automatización, de comprensión, de formato y siguiendo los principios SOLID con mejoras de adaptabilidad e independencia.

Este entregable se divide en tres partes:

- La primera de ellas (correspondiente al E-302) es una revisión de E-201 orientada a tener un formato compatible con las demás herramientas que vamos a utilizar.
- La segunda de ellas (correspondiente al E-302) es un cambio sobre la base de datos de Snowflake. Esta constará en realizar transformaciones sobre ella con DBT.
- La tercera parte es un flujo de trabajo orquestado. Esto es una forma de aplicar las transformaciones necesarias de DBT y las actualizaciones de la BD antes de ejecutar el programa cada vez que se quiera. Esto permite tener información actualizada. Esto se realizará por medio de la herramienta Apache Airflow.

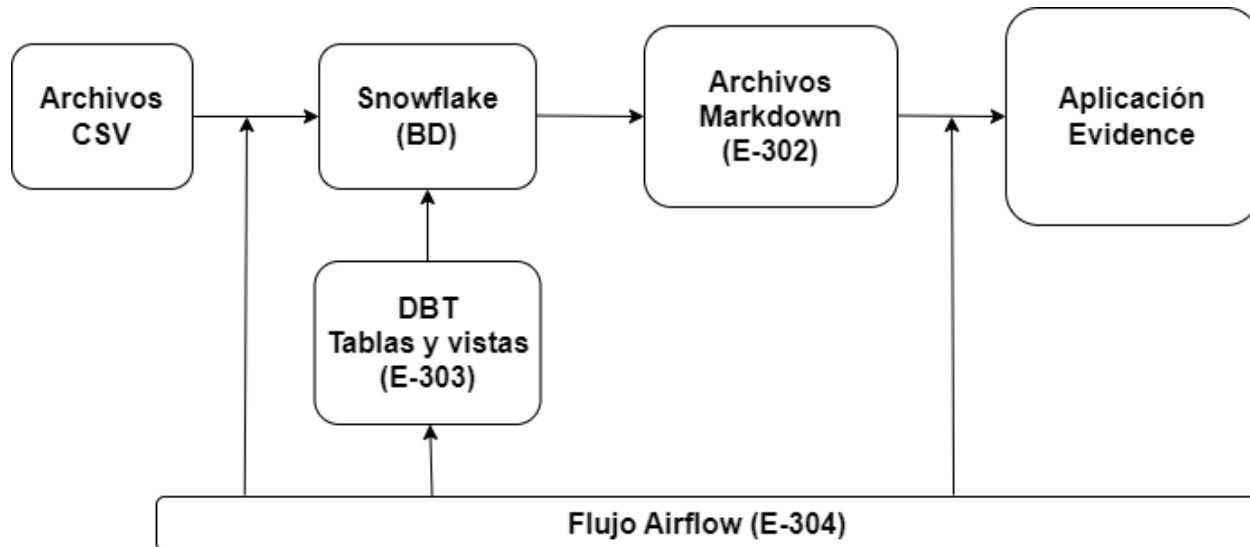


Figura 4.1: Desglose de E-301

## 4.1. Análisis del E-302

El objetivo del entregable E-302 es ofrecer un formato correcto para Evidence. En nuestro caso vamos a tratar de solucionar los problemas vistos en las conclusiones del Sprint 2.

La idea principal es ser capaces de generar Echarts de forma dinámica. Dado que se ejecuta a partir de markdown estático, una idea que suena muy correcta es escribir esta parte estática del código dinámicamente usando un lenguaje de programación clásico.

Hablando con los clientes concluimos que una buena idea sería crear un programa en Python para hacer este escritor de código y utilizar Jinja (un lenguaje de templating) para crear plantillas con los diferentes gráficos de Echarts.

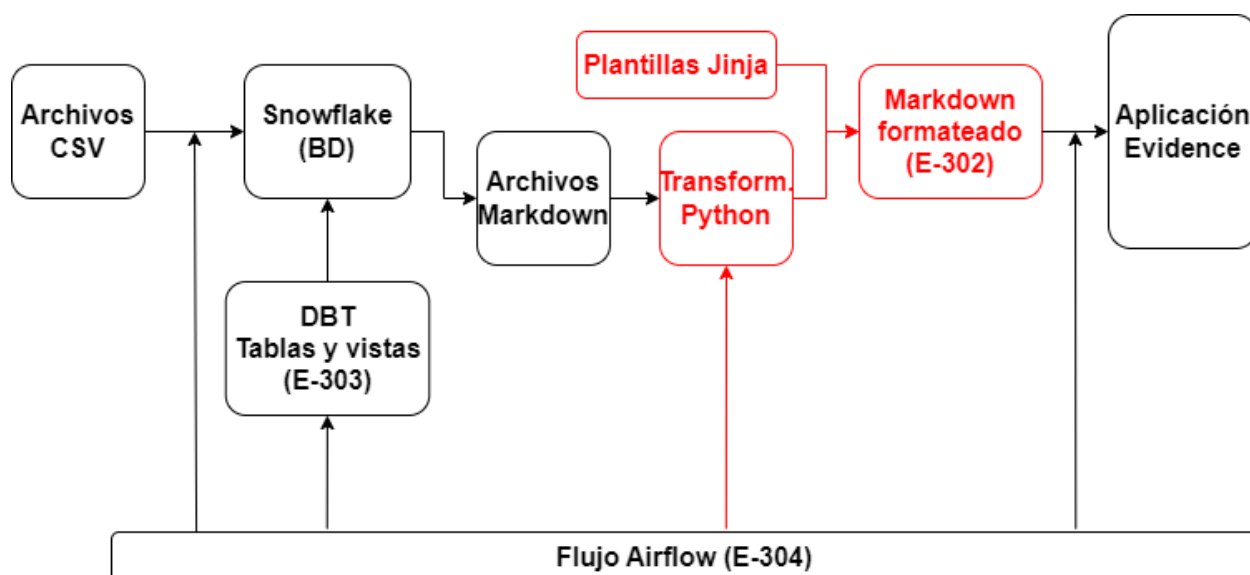


Figura 4.2: Rediseño de E-301

## 4.2. El transformador de Markdown en Python

### 4.2.1 Conocimientos de Python requeridos

Dado que apenas he trabajado nada con Python <sup>11</sup>, veremos si es posible tener algunas funcionalidades que serán útiles para el transformador.

Para ello crearemos un proyecto con PyCharm <sup>12</sup> donde haremos diversas pruebas

---

<sup>11</sup>  **Python**

Python es un lenguaje de programación general de alto nivel muy versátil para el desarrollo de programas y el scripting.

<sup>12</sup>  **Pycharm**

Pycharm es un entorno de desarrollo integrado (IDE). Esta herramienta está especializada en Python, así que resultará útil para el trabajo con proyectos en este lenguaje.

- **Lectura y escritura de ficheros**

Esto resulta ser algo simple de hacer, ya que se puede tomar un cursor de fichero como objeto con `f = open("filename", ["r"|"w"])`.

Existen instrucciones ya conocidas de mi uso de Java como `write()` o `readLines()`.

- **Creación de ficheros**

Es tan solo un caso de uso más de la instrucción anterior, incluyendo un fichero inexistente como filename.

- **Trabajo con entornos virtuales**

Mediante el acceso que proporciona Pycharm a una consola python podemos crear una carpeta que funciona como un entorno virtual.

- **Conexión con Snowflake y lectura de datos**

Existe una librería llamada `snowflake.connector` que permite conectar una base de datos Snowflake sobre Python para acceder a sus datos

En Python se hace un ejemplo con este ejemplo de código (con los campos relacionados con el conector adecuados para la situación). Este ejemplo simplemente es una consulta que devuelve los campos `id` y `author` de la tabla:

```
import snowflake.connector

con = snowflake.connector.connect(user='x',password='x',account='x',
warehouse='x',database='x',schema='x',autocommit=True)
cur = con.cursor()
try:
    id,author = cur.execute('select id,author from CHANGELOG').fetchone()
    print('{0}, {1}'.format(id,author))
finally:
    cur.close
```

Éste es un caso en el que puede aparecer una falla de seguridad ya que datos comprometidos pueden estar muy expuestos. Para solucionar este problema haremos uso de las variables de entorno, lo que veremos más tarde en la sección 4.4.

### 4.2.2 Análisis del transformador de Markdown

Como ya mencionamos anteriormente en el punto 3.3, los bloques de código Echarts se basan en código estático. La idea de esta sección es crear una forma de escribir dinámicamente esos bloques de código. La parte dinámica se basa en:

- El resultado de una query “estructurado”
- El tipo de gráfico
- Una forma de identificar cada uno de los bloques para que no interactúen o repitan estructuras, o incluso se puedan reutilizar algunas usadas.

Por lo que necesitamos una forma de tener esos datos, únicos para cada bloque Echarts, almacenados de alguna manera. Las 2 principales opciones son tenerlos escritos con el resto son:

- A través de un fichero YAML
- Escritos en el propio Markdown a transformar

### 4.2.3 Diseño del transformador de Markdown

Por simplicidad para el usuario, en este diseño se optará por tener de entrada un solo fichero markdown. Para ello se utilizará un solo fichero markdown (el mismo que se utilizará con evidence) en el que pondremos unas anotaciones personalizadas que podremos leer en el propio programa, para generar un YAML y leerlo, sustituyendo el texto entre anotaciones por su bloque Echarts correspondiente.

El fichero de entrada tendrá, entre su código Markdown + Evidence, un bloque del siguiente estilo cuando quiera insertar un bloque Echarts.

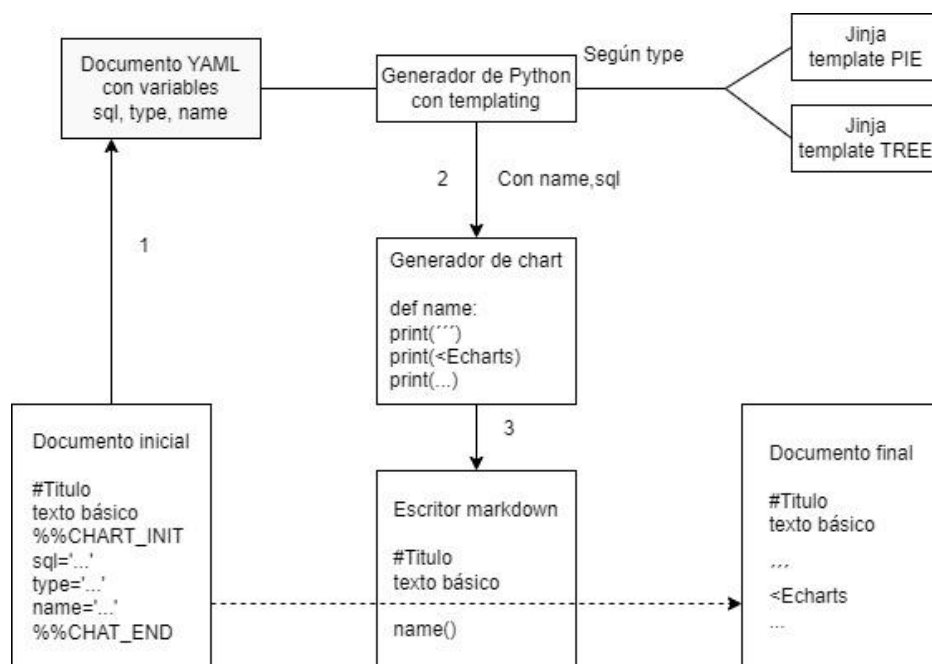


Figura 4.3: Diseño del E-302

Éste esquema muestra el orden en el que se ejecutan las acciones que ocurren en el escritor Markdown

- En primer genera un YAML con la estructura que viene
- En segundo lugar genera, según el tipo de gráficos que sea y con una plantilla Jinja, una “string” con el código del bloque Echarts, insertando el sql y el nombre en la plantilla elegida.
- En tercer lugar, desde la propia clase principal del escritor, reescribe las líneas del documento inicial fuera de las anotaciones en el documento final, y sustituye las partes anotadas por su correspondiente Echart generado (esto se hace mediante el mencionado identificador de gráfico).

#### 4.2.4 Implementación del transformador de Markdown

En este punto se utilizan los conocimientos aprendidos en el punto 4.2.1, así como una estructura de sustitución de elementos de templates de Jinja.

En el escritor utilizaremos la mayoría de utilidades aprendidas, como el método de reescritura:

```

def reescribir(f1, f2):
    linea = f1.readline()
    while ("@@CHART_START" not in linea and linea!=""):
        f2.write(linea)
        linea = f1.readline()
    return linea
  
```



O el generador de YAMLS de la función de sustitución de los bloques etiquetados:

```
#Almacenamiento datos en un YAML
filename= lineaname.split("'")[1]
fyaml = open('%s.yaml' % filename, 'w')
fyaml.write(lineasql)
fyaml.write(lineatype)
fyaml.write(lineaname)
fyaml.close()
```

Aun así la parte más diferencial es el generador de Jinja, ya que trabajamos con una tecnología mucho más interesante y potente:

```
from jinja2 import Environment, FileSystemLoader
import yaml
import os

def generate(name):

    environment = Environment(loader = FileSystemLoader("templates/"))

    with open(f"./yaml/credentials.yaml", "r") as input_file:
        input_data = yaml.safe_load(input_file)

        if input_data['type'] == "pie":
            template = environment.get_template("piechart.jinja2")
        elif input_data['type'] == "tree":
            template = environment.get_template("treechart.jinja2")
        else:
            template = environment.get_template("nochart.txt")

    with open(f"./yaml/credentials.yaml", "r") as input_file:
        input_data = yaml.safe_load(input_file)
    content=template.render(input_data)

    output_file=f"credentials.py"
    with open(output_file, mode="w", encoding="utf-8") as display:
        display.write(content)
        print(f"... credentials updated!")
```

Éste programa es capaz de leer una template, El formato de los archivos .jinja2 es el siguiente, un fichero de texto con anotaciones delimitadas por {{...}} que son susceptibles de ser sustituidas en el generador de arriba. Éstas son algunas líneas de ejemplo de *piechart.jinja2*:

```
def {{ name }}(f, con):
    f.write("`{{ name }}_rows" + os.linesep)
    cur = con.cursor().execute('{{ sql }}')
```

### 4.3. El potencial de Jinja

Antes de detallar el funcionamiento del generador, podemos ver que el sistema entero de la Figura 4.3 es un sistema de anotaciones que se sustituyen. Ante esta situación, en una reunión de seguimiento de requisitos, Xabier me sugirió la posibilidad de aplicar anotaciones Jinja <sup>13</sup> sobre el propio fichero inicial, generando un “doble templating”.

Este diseño supondría algo menos complejo y sucio, más elegante y más eficiente, por lo que el diseño anterior queda medianamente obsoleto y debe tener algunas modificaciones.

- Los YAML se dejan almacenados directamente, sin generarlos ni moverlos.
- No se requiere de una clase escritora, ya que el Markdown ya está escrito y reescribirlo llamando a funciones de modificación de ficheros es innecesariamente pesado.
- Simplemente se debe desarrollar el generador, más complejo y detallado.
- Las templates se mantienen pero con un formato más simple. Ya no tienen que llamar a una función `f.write(...)` que al pasar al escritor escriba todas sus líneas en el fichero de salida, sino que simplemente se debe tomar el texto final, limpiando mucho el formato.

El código del generador se ubica en el Anexo I, pero se puede resumir lo que queremos que haga paso a paso:

1. Creamos un environment, un “lienzo” donde se situarán las plantillas.
2. A través de la dependencia `jinja2schema` y mediante su método `infer`, obtenemos la lista de nombres de los gráficos (identificador que aparece como nombre del YAML).
3. Para cada una de estas dependencias, abrimos el YAML con su nombre, y, según el atributo `type` que estos tengan, se elige una template y se llama a la función `query`, que devuelve una string con el resultado de la query.  
Ésta no puede ser una función única ya que hay templates que tienen más de 2 dimensiones, como los gráficos de burbujas.
4. Se hacen los dos renders (un render es una sustitución de anotación por string) en cascada. En el primer render se sustituyen las anotaciones por las templates, dejando anotaciones para el nombre de los métodos y la query, que se sustituyen en el segundo render.

El resultado de este proceso resulta en la siguiente transformación:

---

<sup>13</sup>  **Jinja**

Jinja es un motor de plantillas web para el lenguaje Python. Es una evolución del motor de plantillas Django que permite mucha más customización.

**Estado inicial**

```
{{employees_per_country}}
```

**Texto tras el primer render**

```
```employees_per_country_rows
{{sql}}
```

```employees_per_country_data
select {{type}} as name, count as value
from ${employees_per_country_rows}
```

<ECharts config={{
  tooltip: {formatter: '{b}: {c} ({d}%)'},
  series: [{
    type: '{{type}}',
    data: employees_per_country_data,
  ]}]
/>
```

**Texto tras segundo render**

```
```employees_per_country_rows
select 'United States of America' as pie, 30 as count
union all
select 'United Kingdom' as pie, 7 as count
union all
select 'Canada' as pie, 2 as count
union all
select 'Germany' as pie, 1 as count
```

```employees_per_country_data
select pie as name, count as value
from ${employees_per_country_rows}
```

<ECharts config={{
  tooltip: {formatter: '{b}: {c} ({d}%)'},
  series: [{
    type: 'pie',
    data: employees_per_country_data,
  ]}]
/>
```

## 4.4. El problema de las variables de entorno

Algunos de los documentos con los que se trabaja tienen datos sensibles. Son documentos que muestran de forma explícita estos datos, como credenciales de cuentas de Snowflake.

La propuesta recibida en una reunión con Xabier fue de utilizar las variables de entorno de Visual Studio Code <sup>14</sup>.

Para ello, abriremos en el menú Run → Add Configuration... Esto abre un fichero settings.json donde podremos añadir el siguiente bloque para introducir variables de entorno:

```
"env": {
    "SNOWFLAKE_USER": "miguel",
    "SNOWFLAKE_PWD": "contraseña"
    ...
},
```

Cuando queremos acceder a estos datos solo tenemos que seguir la siguiente estructura:

```
os.environ.get('SNOWFLAKE_USER')
```

lo cual solo sustituye este campo por su valor equivalente.

Esto nos será de utilidad para cumplir con los principios de seguridad esperados en nuestro proyecto.

De esta forma así es como se llama desde python a una consulta SQL en Snowflake:

```
import os
import snowflake.connector

con =
snowflake.connector.connect(user=os.environ.get('SNOWFLAKE_USER'),
password=os.environ.get('SNOWFLAKE_PASSWORD'),
account=os.environ.get('SNOWFLAKE_ACCOUNT'),
warehouse=os.environ.get('SNOWFLAKE_WHOUSE'),
database=os.environ.get('SNOWFLAKE_DATABASE'),
schema=os.environ.get('SNOWFLAKE_SCHEMA'), autocommit=True)

cur = con.cursor().execute(sql)
```

---

<sup>14</sup>  **Visual Studio Code**

Visual Studio Code es un editor de código con muchas funcionalidades adicionales que lo optimizan para desarrolladores, como depuración o refactorización de código.

## Capítulo 5: Sprint 3. Control de estado de la BD.

### 5.1. Traslado a una BD multitabla

Hasta ahora hemos estado usando como ejemplo una BD changelog, una tabla de doble entrada en formato csv desde la que se han realizado todas las transformaciones y todas las páginas de BI.

Para el E-301 se va a utilizar DBT <sup>15</sup>, una herramienta algo más potente de transformación de tablas, así que se basará en una BD multitabla. En este caso usaremos como ejemplo una BD de ejemplo de una empresa.

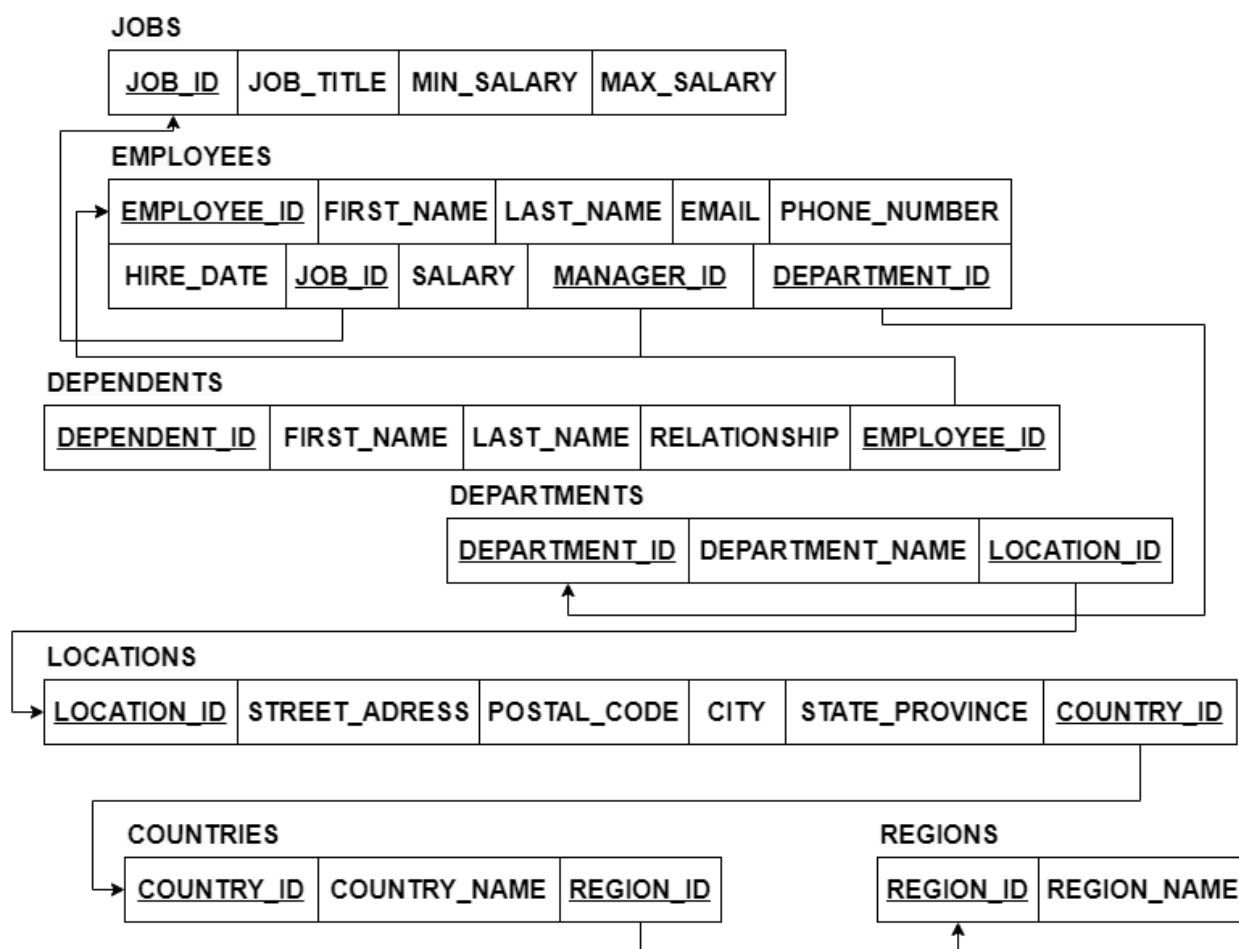


Figura 5.1: Base de datos "company"

<sup>15</sup>  **DBT**

DBT es una herramienta open source de transformación de tablas y ejecución de consultas sobre bases de datos SQL.

## 5.2. DBT como herramienta

La principal utilidad de DBT es construir vistas y tablas. A diferencia de usar Snowflake con SQL para trabajar directamente, DBT permite crear dependencias entre vistas y tablas de forma dinámica: esto significa que existen dependencias entre tablas, por ejemplo con campos precalculados a partir de otros, que se actualizan en tiempo real con dbt.

¿Por qué motivos elegimos DBT?

- Se trata de una solución algo más elegante y mucho más escalable que hacer las tablas a mano desde una consola Snowflake con SQL.
- Es una herramienta utilizada a lo largo de la empresa.
- Puede “comunicar” tablas y transformarlas, lo que nos da utilidad en BD multitabla, como es este nuevo caso.
- Puede usar lenguaje Jinja e integrarlo.
- La instalación de Evidence tiene una opción para hacerse con DBT.

### 5.2.1. Uso y sintaxis de DBT

Más allá de la instalación de DBT core, la cual se realizó de forma sencilla con un comando pip install, abordaremos la necesidad que tenemos de creación de vistas y tablas en DBT.

#### ¿Qué queremos ejecutar con DBT?

Queremos utilizar tan solo 2 instrucciones:

- dbt init nos sirve para inicializar un proyecto con su estructura de directorios, donde al ejecutar evidence se quiere tener una carpeta reports con el proyecto evidence dentro.
- dbt run, que ejecutará los documentos .sql escritos en sintaxis de DBT para generar y actualizar tablas y vistas en consecuencia. Este último queremos que se ejecute antes de cargar Evidence, para tener la BD actualizada en consecuencia cuando se ve la información.

#### ¿Cómo conectamos DBT a Snowflake?

Para cumplir RF-003 se requiere de una conexión a Snowflake

Aunque podamos añadir archivos csv directamente en la carpeta seeds, esto no cumpliría el requisito, por lo que debemos añadir sources (mucho más recomendado) modificando tanto el fichero *dbt\_project.yml* como el fichero de configuración *profiles.yml*. De esta forma las tablas que tengamos en Snowflake se guardarán como sources de DBT.

### Acceso desde DBT a nuestra BD

Podemos realizar consultas en documentos sql. Para ello llamaremos a las tablas almacenadas en las sources con `{{ source('database', 'table') }}`. También podemos llamar a vistas con `{{ ref('nombre_vista') }}`. En nuestro caso usaremos DBT para crear vistas y tablas.

Un ejemplo de consulta sería la siguiente:

```
SELECT region_name, coalesce(sum(num_employees),0) as region_employees,
round(sum(avg_salary*num_employees)/sum(num_employees),2) as avg_salary
FROM {{ source('company', 'regions') }} r
LEFT JOIN {{ source('company', 'countries') }} c on c.region_id=r.region_id
LEFT JOIN {{ ref('employees_per_country') }} e on e.country_name=c.country_name
GROUP BY region_name
ORDER BY coalesce(sum(num_employees),0) desc
```

*Figura 5.2: Creación de una vista con DBT*

Para modificaciones del resultado de la consulta generar vista o tabla

`{{ materialized="view" }}` o `{{ materialized="table" }}` al inicio del fichero.

DBT puede ejecutar bloques Jinja sin problema. Se han hecho algunas pruebas con creaciones de variables o ..., que han resultado funcionales pero no han dado los resultados deseados, y, aunque al final no se han incluido, es bueno contar con este tipo de recursos.

Al tener DBT y Snowflake conectados mediante las sources, podemos usar funciones de Snowflake. Estas sí que han sido de más utilidad, como por ejemplo, para el siguiente ejemplo:

```
SELECT employee_ID, manager_ID, first_name
FROM employees
  START WITH title = 'President'
  CONNECT BY
    manager_ID = PRIOR employee_ID
ORDER BY employee_ID;
```

En este ejemplo se han utilizado unas funciones herencia de Snowflake. Son funciones que permiten acceder a filas “padre” o “hijo” si en una misma tabla se establece una dependencia entre dos campos.

De esta forma estará DBT integrado en nuestro proyecto, lo que nos ayudará a dar escalabilidad (RNF-003) a nuestra gestión de datos

## Capítulo 6: Sprint 3. Orquestación.

Hemos creado a lo largo del proyecto muchas capas que debemos gestionar paso a paso para ejecutar el programa:

- La base de datos debe ser actualizada mediante DBT.
- Se debe utilizar (y crear en caso de no existir) un entorno virtual donde ejecutar de forma segura las transformaciones.
- El transformador de python se ejecuta.
- Se despliega Evidence mediante npm.
- Entre todos los pasos anteriores hay múltiples cambios de directorio.

Esto rompe uno de los principios principales, el de ofrecer simplicidad para el cliente, y no hacerle pasar por carpetas y copiando y pegando comandos.

Para ello usaremos un orquestador. Un programa con el que programar ejecuciones, en nuestro caso dejar preparada una ejecución que cumpla todos los pasos listados.

### 6.1. Airflow, DAGs y operadores

El orquestador que usaremos es Apache Airflow <sup>15</sup>.

Para nuestro proyecto necesitamos crear unos ficheros llamados DAGs. Los DAGs son unas definiciones de flujos de instrucciones. Dentro de la carpeta dags, declaramos unos archivos .py que podemos lanzar desde una interfaz web.

Para nuestro DAG necesitamos alguna funcionalidad:

- Funciones basadas en operadores, que permiten ejecutar acciones como instrucciones Bash o Python.

```
bash_command = BashOperator(
    task_id="bash_command ",
    bash_command="ls -a
```

- Un orden de ejecución final, que nos ayudará a establecer el orden en el que se ejecutarán las funciones declaradas. Pueden hacerse árboles de decisión en este apartado.

```
bash_command >> python_build >> python_executi
```



## 6.2. Volúmenes y Docker Compose

Ya se han visto anteriormente algunas pinceladas sobre la tecnología Docker. Sin embargo, la distribución que se ha usado de Airflow viene empaquetada en Docker Compose. Al ejecutarse arranca múltiples contenedores, como un worker o un inicializador (que no sabemos en detalle qué hacen).

Lo que necesitamos en airflow es ejecutar un DAG que pueda acceder a nuestro código y ejecutarlo. Para ello necesitamos tanto poder ejecutar DAGs como conocer las rutas relativas desde los archivos .py hasta nuestro proyecto

Para acceder a la interfaz web simplemente tenemos que realizar un `docker-compose up` en la consola y desde allí se administra que DAGs ejecutar o tener activos.

Para localizar el proyecto vamos a crear un volumen nuevo (aparte de los ya existentes), donde poner nuestro proyecto

Al ver como el archivo *docker-compose.yaml* genera los directorios del proyecto Airflow se aprecia esto:

```
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
  - ./plugins:/opt/airflow/plugins
  - ./airflow-data/airflow.cfg:/opt/airflow/airflow.cfg
```

Estos volúmenes se corresponden con las carpetas dags, logs y plugins que se generan en el directorio, por lo que incluiremos nuestro proyecto en una nueva carpeta llamada *company*, y añadiremos un nuevo volumen “ `- ./company:/opt/airflow/company` ”

Para identificarlo usaremos un DAG simple que vaya haciendo distintos comandos ls y cd hasta llegar a un directorio conocido. Con “ `cd ../../opt` ” es posible acceder al volumen equivalente a nuestra carpeta airflow, donde están las carpetas dags, logs, plugins y company.

Así ejecutaremos nuestro DAGs que haga las tareas mencionadas anteriormente. También se han hecho otros DAGs de apoyo que ejecutan cosas que solo son necesarias una vez, cómo instalar DBT o crear un entorno virtual Python.

## Capítulo 7: Estructura global del proyecto.

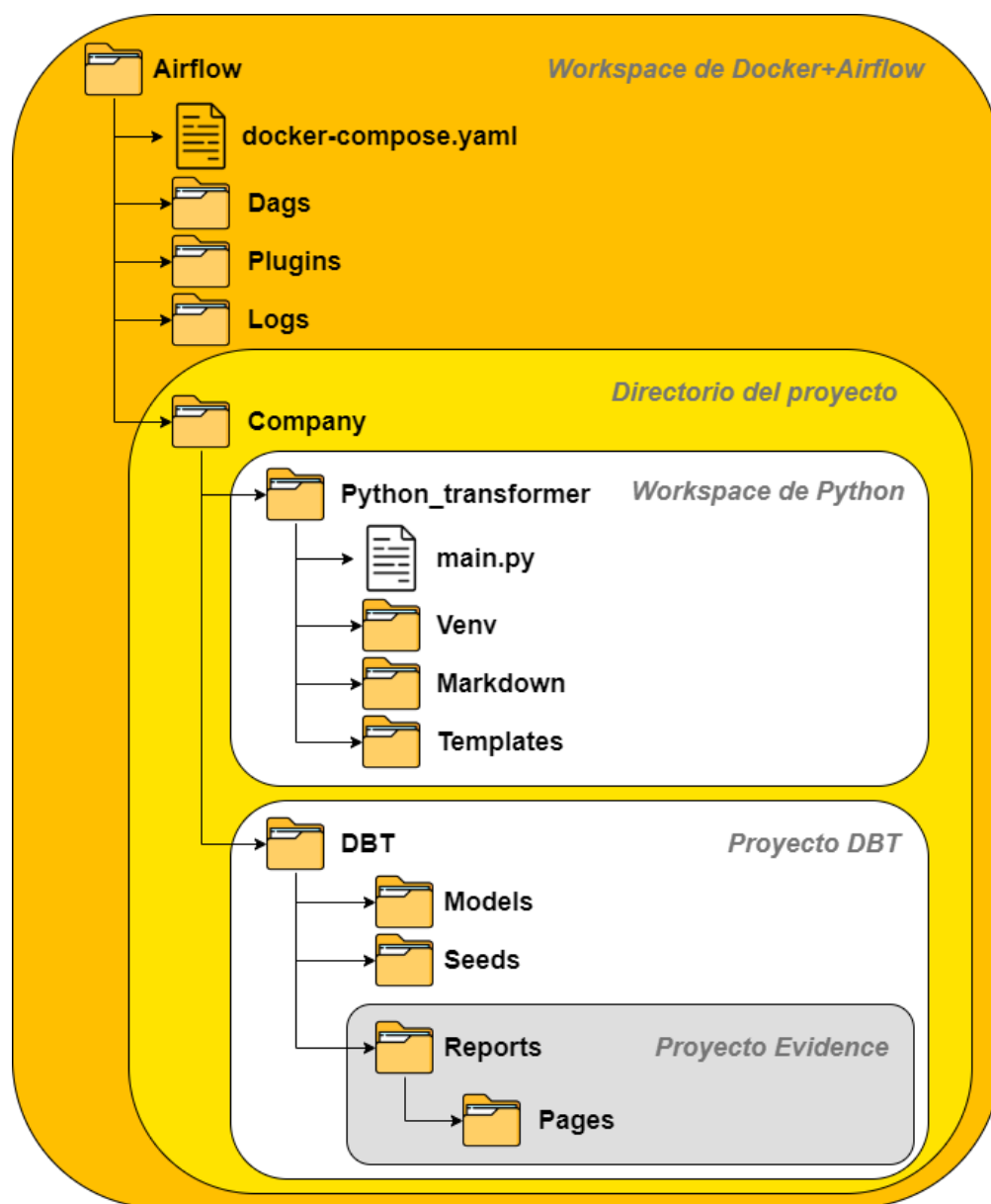


Figura 7.1: Estructura global del proyecto

La figura 7.1 muestra un punto de vista general sobre el Sprint 3. Cada vez que se añaden nuevas herramientas suele requerir un entorno de trabajo propio. Al querer “evolucionar” el mismo producto se acaba con un escenario un poco confuso.

Con esta figura se pretende tener una perspectiva comprensible de cómo ha quedado el proyecto, así como ser de ayuda para entender las integraciones.

## Capítulo 8: Revisión de la planificación.

## 8.1. Revisión de requisitos

A nivel general hay bastante satisfacción con los requisitos. Sin embargo han surgido problemas con dos de ellos:

- **RF-005:** El producto final contará con opciones de portabilidad.

El hecho de que la instalación portable de Evidence no funcionase ha supuesto un problema en este requisito, ya que aunque parte del producto sea portable, el producto en sí no lo es.

El coste de realizar la infraestructura necesaria para la portabilidad se ha considerado excesiva y se ha preferido no cumplir RF-005.

- **RF-006:** El producto final soportará inserciones de metadatos.

Por falta de tiempo no se ha generado ningún flujo de datos que alimente la BD en tiempo real. Solamente se han hecho pruebas con datos estáticos o añadidos manualmente.

## 8.1. Revisión temporal

A continuación se exponen los diagramas de Gantt y de hitos revisados:

| Tabla 8.1 - Diagrama de Gantt - Revisión (Semanas 1-10) |    |    |    |    |    |    |    |    |    |     |
|---|----|----|----|----|----|----|----|----|----|-----|
| Tarea   | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 |
| S1 - Estudio y comparativa                              |    |    |    |    |    |    |    |    |    |     |
| Tiempo real   |    |    |    |    |    |    |    |    |    |     |
| S2 - Análisis y diseño                                  |    |    |    |    |    |    |    |    |    |     |
| Tiempo real   |    |    |    |    |    |    |    |    |    |     |
| S2 - Implementación                                     |    |    |    |    |    |    |    |    |    |     |
| Tiempo real   |    |    |    |    |    |    |    |    |    |     |
| S3 - Formación  |    |    |    |    |    |    |    |    |    |     |
| Tiempo real   |    |    |    |    |    |    |    |    |    |     |
| S3 - Análisis y diseño                                  |    |    |    |    |    |    |    |    |    |     |
| Tiempo real   |    |    |    |    |    |    |    |    |    |     |

| Tabla 8.2 - Diagrama de Gantt - Revisión (Semanas 1-10), cont. |  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|--|
| S3 - Implementación  |  |  |  |  |  |  |  |  |  |  |
| Tiempo real  |  |  |  |  |  |  |  |  |  |  |
| Desarrollo de la memoria                                       |  |  |  |  |  |  |  |  |  |  |
| Tiempo real  |  |  |  |  |  |  |  |  |  |  |
| Seguimiento y control  |  |  |  |  |  |  |  |  |  |  |
| Tiempo real  |  |  |  |  |  |  |  |  |  |  |

| Tabla 8.3 - Diagrama de Gantt - Revisión (Semanas 10-27) |     |     |     |     |     |     |     |     |     |     |
|--|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Tarea  | S11 | S12 | S13 | S14 | S15 | S16 | S17 | S18 | S25 | S27 |
| S3 - Implementación                                      |     |     |     |     |     |     |     |     |     |     |
| Tiempo real  |     |     |     |     |     |     |     |     |     |     |
| Desarrollo de la memoria                                 |     |     |     |     |     |     |     |     |     |     |
| Tiempo real  |     |     |     |     |     |     |     |     |     |     |
| Seguimiento y control                                    |     |     |     |     |     |     |     |     |     |     |
| Tiempo real  |     |     |     |     |     |     |     |     |     |     |
| Preparación de la defensa                                |     |     |     |     |     |     |     |     |     |     |
| Tiempo real  |     |     |     |     |     |     |     |     |     |     |
| Depósito de la memoria                                   |     |     |     |     |     |     |     |     |     |     |
| Tiempo real  |     |     |     |     |     |     |     |     |     |     |
| Defensa del trabajo                                      |     |     |     |     |     |     |     |     |     |     |
| Tiempo real  |     |     |     |     |     |     |     |     |     |     |

Revisando lo planeado y lo cumplido, a grandes rasgos la planificación se ha cumplido. Ha habido algunas incongruencias como la semana 6, donde tuve días de descanso, o algunas semanas al final del proyecto donde se quedó la redacción de la memoria algo descolgada.

| Tabla 8.4 - Diagrama de hitos revisado |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |
|--|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| Semana                                 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 25 | 27 |
| E-101                                  | ◆ |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |
|  | ◆ |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |
| E-201                                  |   |   | ◆ |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |
|  |   |   | ◆ |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |
| E-301                                  |   |   |   |   |   |   |   |   |   |    |    |    |    | ◆  |    |    |    |    |
|  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | ◆  |    |    |
| E-302                                  |   |   |   |   |   |   |   | ◆ |   |    |    |    |    |    |    |    |    |    |
|  |   |   |   |   |   |   |   |   |   | ◆  |    |    |    |    |    |    |    |    |
| E-303                                  |   |   |   |   |   |   |   |   |   |    |    | ◆  |    |    |    |    |    |    |
|  |   |   |   |   |   |   |   |   |   |    |    |    |    | ◆  |    |    |    |    |
| E-304                                  |   |   |   |   |   |   |   |   |   |    |    |    | ◆  |    |    |    |    |    |
|  |   |   |   |   |   |   |   |   |   |    |    |    |    |    | ◆  |    |    |    |
| E-401                                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    | ◆  |    |
| E-402                                  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    | ◆  |

La implementación del sprint 3 se ha alargado más de lo esperado dados los contratiempos existentes con el rediseño del E-302, pero ha resultado ser algo muy bueno para cumplir mejor con RNF-003 (escalabilidad) y RNF-006 (optimización de código).

Asimismo, en el diagrama de hitos podemos ver que tanto el parón de la semana 6 como el rediseño del E-302 han afectado a las fechas objetivo de los entregables del Sprint 3

## Capítulo 9: Conclusiones del trabajo.

### 9.1. Conclusiones del proyecto

Estoy bastante satisfecho a nivel general con el resultado del trabajo. Puede que no haya sido tan amplio o tan depurado como me hubiera gustado, pero considero que he hecho muchas más cosas de las que esperaba en un inicio y he sido capaz de entregar un producto completo, funcional y propio.

He trabajado con muchas tecnologías nuevas y las he sido capaz de integrar en un proyecto propio. Considero que son herramientas que pueden ser de utilidad para otros proyectos y con las que contar como recursos.

Si tuviera que dedicarle más tiempo trataría de hacerlo más user-friendly, de forma que la portabilidad sea completa y el proceso esté más abstraído. La forma de ejecutarlo es algo laberíntica y es la mayor pega que podría poner al proyecto.

La empresa me ha proporcionado tanto personal de ayuda, como casos de uso sobre los que trabajar, como unas muy buenas condiciones. Estoy agradecido tanto con su ayuda como con la de la universidad, y con sus respectivos tutores y su gran trabajo y atención.

### 9.2. Conclusiones personales

A nivel de lecciones personales he descubierto lo complicado que es seguir una planificación. He notado que requiere, en mucha más dimensión de la que esperaba, de experiencia en medir tiempos y prever problemas desconocidos y la capacidad de solucionarlos.

También me he sorprendido gratamente a mi mismo a la hora de encontrar soluciones para los problemas que iban surgiendo, y la capacidad de solventarlos por cuenta propia, por medio de búsquedas, tests unitarios, bibliotecas, etc.

He encontrado mucho gusto en explorar tanto Snowflake como herramientas de BI, buscando funcionalidad aplicada al diseño y a la visualización de la información. No me importaría mas indagar en este ámbito en el futuro.

# Bibliografía

Cómo sincronizar proyectos con Bitbucket

<https://www.youtube.com/watch?v=8wkRZHn4FIE>

Documentación Mprove

<https://docs.mprove.io/top/topics/environments>

Guía de Sintaxis de Markdown

<https://www.markdownguide.org/basic-syntax/>

Documentación Evidence

<https://docs.evidence.dev/>

Editor de gráficos Echarts en línea

<https://echarts.apache.org/examples/en/index.html>

Documentación de Snowflake: Carga y migración de datos

<https://docs.snowflake.com/en/sql-reference/commands-data-loading>

Documentación de Snowflake: Funciones herencia

<https://docs.snowflake.com/en/user-guide/queries-hierarchical>

Documentación de Snowflake: Conexión con Python

<https://docs.snowflake.com/developer-guide/python-connector/python-connector>

Cómo añadir variables de entorno a Python y Visual Studio Code

<https://stackoverflow.com/questions/5971312/how-to-set-environment-variables-in-python>

Documentación de Jinja: sintaxis y notación.

<https://jinja.palletsprojects.com/en/3.1.x/>

Setup de Dbt Core.

<https://docs.getdbt.com/docs/core/about-core-setup>

Distribución de docker en Airflow: Código fuente y guía de instalación.

<https://github.com/marclamberti/docker-airflow/blob/main/docker-compose.yml>

<https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html>

Tutoriales sobre Airflow y DAGs.

<https://www.youtube.com/@MarcLamberti>

## Anexo I: Código del generador Python

```

from jinja2 import Environment, FileSystemLoader
from jinja2schema import infer
from queries import *

#Parámetro de entrada: fichero a procesar
def transform(file):
    environment = Environment(loader = FileSystemLoader("."))

#Paso 1: Identificamos que gráficos aparecen en el fichero
    namelist=[]
    with open(f"{file}", "r") as raw:
        strraw = str("".join(raw.readlines()))
        chartnames = infer(strraw)
        for chartname in chartnames.items():
            namelist.append(chartname[0])

#Paso 2: Declaramos un diccionario donde almacenaremos los nombres de los gráficos y
la parte dinamica para el paso 5
    charts_dict={}

#Paso 3: Abrimos los YAML de los datos de los gráficos del fichero
    for filename in namelist:
        with open(f"yaml/{filename}.yaml", "r") as input_file:
            input_data = yaml.safe_load(input_file)

#Paso 4: Insertamos datos en la template de gráfico según su type.
#Obtenemos el resultado dinámico de procesar el sql y lo almacenamos como query
        if input_data['type'] == "pie":
            chart_template = environment.get_template("templates/piechart.jinja2")
            query=piequery(input_data['sql'])
        elif input_data['type'] == "tree":
            chart_template = environment.get_template("templates/treechart.jinja2")
            query = treequery(input_data['sql'])
        else:
            chart_template = environment.get_template("templates/chartdisplay.txt")
            query="*ERROR: type not found*"

#Paso 5: Primer render, creamos los gráficos con las queries y el nombre y los
almacenamos por nombre en el diccionario
        chart=chart_template.render(name=input_data['name'],query=query)
        charts_dict.update({input_data['name']:chart})

#Paso 6: Segundo render, creamos el markdown con los charts
    str_template="/" + file
    template=environment.get_template(str_template)
    content=template.render(charts_dict)

#Paso 7: Fichero de salida
    output_file=f"{file}"
    with open(output_file, mode="w", encoding="utf-8") as display:
        display.write(content)
        print(f"{output_file} written!!!")

```



## Anexo II: Código del DAG de ejecución principal

```

from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from datetime import date, datetime

with DAG("main_dag", start_date= datetime(2023,4,12),
schedule_interval="@daily", catchup=False) as dag:

    test_pwd = BashOperator(
        task_id="test_pwd",
        bash_command="pwd && \
                      cd .. && \
                      cd .. && \
                      cd /opt/airflow/company && \
                      ls -a"
    )
    python_execution = BashOperator(
        task_id="python_execution",
        bash_command="cd ../../opt/airflow/company/transformer && \
                      source venv/bin/activate && \
                      python main.py && \
                      cd .."
    )
    move_files = BashOperator(
        task_id="move_files",
        bash_command="cd ../../opt/airflow/company && \
                      rsync -a transformer/raw/* company/reports/pages && \
                      rsync -a transformer/markdown/* company/reports/pages"
    )
    dbt_run = BashOperator(
        task_id="dbt_run",
        bash_command="cd ../../opt/airflow/company/company && \
                      pip install \
                      dbt-core \
                      dbt-snowflake && \
                      dbt run --project-dir . --profiles-dir . && \
                      cd .."
    )

    init_evidence = BashOperator(
        task_id="init_evidence",
        bash_command="cd ../../opt/airflow/company/company && \
                      pip install nodejs && \
                      pip install npm && \
                      npm --prefix ./reports run dev"
    )

test_pwd >> dbt_run >> python_execution >> move_files

```