Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Ampliación del Lenguaje de Validación de Datos PALADIN para su Uso en Bases de Datos NoSQL

Autor: Celia Martín Muñoz

Tutor: Antonio Jesús Díaz Honrubia

Madrid, julio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Ampliación del Lenguaje de Validación de Datos PALADIN para su Uso en Bases de Datos NoSQL
Julio 2025

Autor: Celia Martín Muñoz

Tutor:
Antonio Jesús Díaz Honrubia
Lenguajes y Sistemas Informáticos e Ingeniería del Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Para cualquier ámbito de investigación, más frecuentemente en espacios científicos, se requiere la utilización de bases de datos con el fin de almacenar la información de una manera más ordenada en conjuntos que puedan ser útiles durante el seguimiento de un estudio que se esté realizando. En estos estudios, las bases de datos pueden ser analizadas y utilizadas para más tarde tomar decisiones diversas teniendo en cuenta los valores de los datos. El problema principal de estos casos es que en muchas ocasiones pueden contener errores lógicos o semánticos debido a factores humanos o errores en su generación ya que se hizo uso de modelos de Inteligencia Artificial (IA) que en el proceso cometieron errores; debido a esto, muchos datos deben pasar por un proceso de limpieza antes de ser utilizados. Para ello, se encuentran los lenguajes de validación que realizan el proceso de comprobar si los datos de entrada dados son correctos respecto a criterios especificados para el estudio del que se haga la investigación deseada.

Entre estos lenguajes de validación se encuentra el lenguaje de validación PALADIN. Éste se encarga específicamente de la verificación de que los valores de las bases de datos dadas como entrada cumplan unos criterios concretos formados en base a sucesivas consultas individuales que acaban dando lugar a un conjunto con los datos que cumplen las validaciones aplicadas en las consultas. Para este lenguaje se usó como caso de prueba información sobre los tratamientos de pacientes con cáncer de mama con un gen HER2 amplificado, para que se garantice una integridad segura en la futura utilización de estos datos para los estudios en los que se pueda necesitar el seguimiento de estos tratamientos.

Para la realización de este TFG se pensó en una posible ampliación del lenguaje de validación PALADIN para bases de datos de tipo no relacional, más concretamente, en lenguaje MongoDB. Así se podría aumentar el número de tipos de bases de datos aceptables a validar por este prototipo. Previamente al nuevo diseño del prototipo PALADIN, se tuvieron que adaptar los ficheros JSON que contienen las configuraciones correspondientes al árbol que contiene las diferentes validaciones y por ello las consultas necesarias para el proceso a consultas en búsqueda de documentos para bases en MongoDB.

A parte de PALADIN, se ha llevado a cabo la ampliación del Synthetic Data Generator o SDG, el cual se encarga de generación de bases de datos sintéticos en formatos MySQL, CSV y grafos de conocimiento en Resource Description Framework (RDF) del caso de prueba anteriormente nombrado. Para este TFG, también se aumentó el rango de estos formatos implementando la obtención de una base de datos, pero en MongoDB.

Tras esto, se decidió analizar los resultados respecto al rendimiento de PALADIN con esquemas de diferente número de nodos, es decir, dependiendo de la dificultad a la que se especifica con una consulta. Para esta ampliación, el tiempo de ejecución resultó ser más elevado comparado con otros formatos siendo la funcionalidad que más tarda seguida de la implementación para grafos de conocimiento RDF y, tras esta, la hecha para MySQL.

Abstract

For any field of research, most often in scientific spaces ranging from medical to computer science, the use of databases is required to store information in a more organized and structured way in sets that can be useful during the follow-up of a study being conducted. In these studies, the databases can be analysed and used for later decision-making based on data values. The main problem of these cases is that in many cases they may contain logical or semantic errors due to human factors or simply errors in their generation since it was made use of models of Artificial Intelligence (AI) which in the process made mistakes; Because of this, many data must go through a verification process before being used. For this, we find validation languages which are a type of language that carry out the process of checking whether the given input data meet certain specified criteria for the study from which the desired research is made.

Among these validation languages is the PALADIN validation language [1]. This is specifically responsible for checking that the values of the databases given as input meet specific criteria formed based on successive individual queries which eventually result in a set with data meeting the validations applied in the consultations. This language was used as a test case for information on treatments of breast cancer patients with an amplified HER2 gene, to ensure safe integrity in the future use of these data for studies where follow-up of such treatments may be required.

For the realization of this TFG it was thought of a possible extension of the validation language PALADIN for non-relational databases, more specifically in MongoDB language. This would amplify the number of types of available input databases with which such data validations could be carried out. Prior to the redesign of the PALADIN prototype, we had to adapt the JSON files containing the configurations corresponding to the tree that contains the different validations and therefore the queries necessary for the process to queries in search of documents for bases in MongoDB.

In addition to PALADIN, the Synthetic Data Generator [2] or SDG has been extended, which is responsible for generating synthetic databases in MySQL, CSV and knowledge graphs in Resource Description Framework (RDF) of the previously named test case with medical information from patients with the amplified HER2 gene. For this TFG, we also increased the range of these formats by implementing database retrieval, but in MongoDB.

After this, it was decided to implement a part to analyse the results regarding the execution time of PALADIN depending on the number of nodes that are used in the PALADIN scheme, that is, depending on the difficulty specified with a query. For this extension, the execution time turned out to be higher compared to its use with other formats being the functionality that is most late followed by the implementation for MySQL, and, after this, the one made for RDF knowledge graphs.

Tabla de contenidos

1 Introducción

En ocasiones, hay estudios que necesitan un seguimiento restrictivo de un proceso perteneciente a entidades concretas, por ejemplo, en casos médicos con pacientes que necesitan una operación, pero dependiendo del tiempo que lleven usando un tratamiento determinado o de si han sido ya operados podrían necesitar otro tratamiento diferente o la realización de otra operación especifica dependiendo de su propio caso. Para la realización de dichas investigaciones, se realizan búsquedas concretas de esas condiciones en los conjuntos donde se almacenen, tomando como ejemplo el caso médico anterior, los historiales médicos de los diferentes pacientes en el estudio.

Esa acción es de la que se encargan los llamados lenguajes de validación, cuya función es realizar dicho seguimiento añadiéndolas como base de datos y clasificarlas como válidas o inválidas según las restricciones definidas para hacer una limpieza de los datos que no cumplen dichos criterios y devolver como resultado los correctos respecto a las especificaciones dadas. Como resultado, ofrecerían el conjunto de entidades que cumplirían o no con las reglas de validación del esquema ofrecido diciendo cuantas coinciden con las validaciones y cuantas no las cumplen.

1.1 Motivación

Actualmente, se almacenan datos de todo tipo y, además en la mayoría de los ámbitos en los que se necesiten, la cantidad de datos que se producen es extremadamente elevada. Esta gran acumulación de datos implica que muchos de ellos al ser almacenados se generen de manera errónea o llenos de inconsistencias ente ellos, ya sea debido a fallos humanos corrientes o que, debido al moderno avance de los modelos de inteligencia artificial, se hayan generado gracias al uso de estas máquinas y, debido a que suelen ser levemente imprecisas, suelen cometer errores en la producción de esta información. A estos datos inexactos se les denomina “datos sucios” o Dirty Data; generalmente, los principales problemas que se pueden encontrar en ellos son valores atípicos según el contexto dado, por ejemplo, que en un caso médico por error se haya puesto que el paciente que continua estando vivo tenga 210 años y debido a un error de escritura a la hora de poner dicha edad se haya almacenado dicho dato erróneo; otros errores que se pueden observar en este tipo de dato es que se generen datos duplicados, incompletos o desactualizados de manera que, cuan mayor es la cantidad de estos datos, se reduce la fiabilidad de la información que se acaba almacenando disminuyendo así la calidad de ellos.

Para mejorar la calidad de las bases de datos, se llevan a cabo limpiezas en ellas en las cuales se comprueba la veracidad de los datos contenidos mediante validaciones para comprobar si estos datos cumplen o no con ellas. De esta función se encargan los lenguajes de validación, los cuales se encargan de que los datos usados como objetos de entrada cumplan unas consistencias semánticas necesarias para que, si posteriormente son necesarios para ser

utilizados, se puedan usar con la tranquilidad de que el estudio a realizar se vaya a llevar a cabo con datos de alta calidad y completamente limpios.

Entre muchos de los lenguajes de validación con estas características se encuentra el lenguaje de validación PALADIN [1]. Este lenguaje de validación se encuentra diseñado para el análisis de criterios en estudios o investigaciones concretas en las cuales las bases usadas como objeto estén basadas en procesos determinados, es decir, con datos cuyo valor pueda variar dependiendo de la fase en la que se encuentre del proceso. Como caso de prueba se usó el seguimiento de bases de datos probadas para un estudio sobre la pauta de tratamiento para los pacientes con cáncer de mama con el gen HER2 amplificado para, a partir de ahí, determinar el siguiente paso en el tratamiento o medicaciones de un paciente dependiendo de los tratamientos o las operaciones a los que han sido sometidos o de otras condiciones, por ejemplo, si un paciente ha recibido tratamiento neoadyuvante, es decir, si antes de una cirugía ha sido tratado con una medicación específica para que el tumor reduzca su tamaño. El principal problema es que PALADIN actualmente solo se encuentra implementado para bases de datos relacionales en lenguaje MySQL[20] y para grafos de conocimiento RDF[21].

1.2 Objetivos

El objetivo principal de este TFG es ampliar la implementación de este prototipo PALADIN para que pueda operar con bases de datos no relacionales (o NoSQL), más concretamente en MongoDB, es decir, bases orientadas a documentos y así poder ampliar el rango de bases de datos que puedan ser comprobadas y, además, facilitar la implementación y transcripción de bases de datos de MySQL a MongoDB. Para alcanzar este objetivo, se proponen unos objetivos específicos que se llevarán a cabo para el completo desarrollo de este proyecto, estos objetivos específicos son los siguiente:

1. **Unificación y compatibilidad entre el prototipo PALADIN y las herramientas necesarias para su usabilidad:** Para la correcta interoperabilidad entre el prototipo PALADIN y las bases de datos NoSQL se necesitará un conjunto de datos de prueba y un fichero JSON con los criterios a analizar en dichos conjuntos. Este conjunto se obtiene del traspaso de una base de datos relacional ya existente en MySQL a una base de datos no relacional en MongoDB mediante un programa auxiliar que se implementó. Este programa se conectará y recibirá una base de datos en MySQL y pasará las tablas de dentro de esta base de datos a coleccione en una base de datos en MongoDB. Tras esto, se realizará el paso de las consultas para datos sintéticos en MySQL del fichero JSON que ya se tenía de proyectos anteriores a consultas orientadas a documentos para bases en MongoDB. Finalmente, PALADIN hará uso de dicho fichero de consultas el cual recibirá el link URI, que permite la conexión a un clúster de MongoDB (en este caso, se hizo uso de MongoDB Atlas UI y de MongoDB Compass para el almacenamiento de una manera

más cómoda y portable de estas bases de datos) y la base de datos en la cual se quiere hacer la validación. Tras especificar el fichero JSON que se quiere usar y un algoritmo de búsqueda el cual se usará en el árbol en el que se hará la validación, devolverá el número de datos totales y, por consultas, se devolverán las validaciones que va realizando y las violaciones de estas validaciones que va encontrando.

2. **Ampliación del prototipo PALADIN:** Tras haber obtenido las herramientas necesarias para el correcto funcionamiento de PALADIN, se llevará a cabo la aceptación de este prototipo de bases de datos no relacionales basadas en documentos como lo es MongoDB. Para ello, se hará uso de las configuraciones que se encuentran ya disponibles y funcionales, además de los conjuntos de datos de prueba que se encuentran tanto en bases de datos relacionales como en gafos de conocimiento, los cuales serán necesarios para su compresión y, así, realizar su ampliación para

3. **Ampliación del Generador de Datos Sintéticos:** Este será obtenido utilizando el Synthetic Data Generator o SDG[2], el cual produce conjuntos de datos sobre el tratamiento del cáncer de mama siguiendo la distribución de una población real de pacientes con cáncer de mama con un gen HER2 amplificado. Actualmente, este generador también se encuentra solamente implementado para ofrecer conjuntos de datos en formato de bases de datos relacionales y, a partir de estos, de grafos de conocimiento; de esta manera también formaría parte de este TFG la ampliación del generador para que también produzca conjuntos de bases de datos no relacionales en MongoDB.

4. **Análisis de los resultados obtenidos:** Se terminará realizando un análisis basado en la comparación de los datos obtenidos por PALADIN respecto a un conjunto de otros lenguajes de validación y de otros tipos de configuraciones y formatos de bases de datos para poder observar la diferencia de errores entre el prototipo y otras maneras de hacer uso de este lenguaje. Para llevar a cabo este objetivo, se tendrán en cuenta las trazas y las métricas obtenidas tras el cumplimiento satisfactorio de los anteriores objetivos.

Figura 1: Logo del lenguaje de validación PALADIN. Fuente: [1].

1.3 Planificación Final del Trabajo

Para la correcta organización de este trabajo, se llevó a cabo la determinación de numerosas tareas que ayudaban a seguir un orden determinado en el estudio y la implementación de este proyecto. Estas tareas que forman un resumen del desarrollo de este TFG son las siguiente:

1. Organización del proyecto respecto a las entregas a realizar.
2. Estudio de las bases de datos NoSQL, más específicamente MongoDB.
3. Estudio de las características del prototipo PALADIN.
4. Estudio del lenguaje de programación Python y su uso asociado a las bases de datos.
5. Diseño y desarrollo de la nueva arquitectura del prototipo PALADIN.
6. Adaptación del generador de traducción del testbench de conjuntos de datos a una base de datos NoSQL.
7. Obtención del testbench en formato de base de datos NoSQL.
8. Implementación de la interoperabilidad entre el prototipo y bases de datos NoSQL.
9. Validación de la implementación.
10. Análisis de las pruebas obtenidas mediante la implementación.
11. Documentación de todo proceso realizado para la realización del TFG.

Se siguió un plan para ir completando estas tareas se dividió el trabajo por semanas siendo la primera semana la correspondiente al día 24 de febrero de 2025 y la última semana (S19) la correspondiente al día 2 de julio de 2025. Esta organización se representó en un Diagrama de Gantt, como se puede observar en la Figura 2, que se ha ido actualizando hasta el último momento de este proyecto

TAREA	ASIGNADO A	INICIO	FIN	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19
Estudio de las bases de datos NoSQL y del prototipo PALADIN																						
Tarea 1	Organización entregas del proyecto	24-2-25	1-3-25																			
Tarea 2	Estudio de bases de datos NoSQL (MongoDB)	24-2-25	9-3-25																			
Tarea 3	Estudio de las caracctersticas del prototipo PALADIN	24-2-25	9-3-25																			
Tarea 4	Estudio del lenguaje de programación Python	24-2-25	9-3-25																			
Diseño de la nueva arquitectura del prototipo de PALADIN																						
Tarea 5	Diseño y desarrollo del prototipo PALADIN	17-4-25	15-6-25																			
Adaptación del generador de traducción del testbench de conjuntos de datos a una base de datos NoSQL																						
Tarea 6	Actualización del traductor a bases de datos NoSQL	10-3-25	19-6-25																			
Tarea 7	Obtención del testbench a una base de datos NoSQL	10-3-25	27-4-25																			
Implementación de la interoperabilidad entre el prototipo y bases de datos NoSQL																						
Tarea 8	Implementación de la conexión entre PALADIN y las bases de datos NoSQL	17-4-25	22-6-25																			
Validación de la implementación																						
Tarea 9	Validación de la implementación	26-5-25	24-6-25																			
Tarea 10	Análisis de las pruebas obtenidas mediante la correcta implementación	26-5-25	24-6-25																			
Documentación necesaria para el TFG																						
Tarea 11	Documentación de todo proceso realizado para la realización del TFG	24-2-25	2-7-25																			

Figura 2: Diagrama de Gantt correspondiente al plan de proyecto de este TFG.

1.4 Estructura de la Memoria

La estructura de la memoria correspondiente a este Trabajo de Fin de Grado estará organizada por capítulos. Comenzando por el Capítulo 2, en él se comentan los diversos Trabajos Previos realizados para dar un leve contexto en relación con el proyecto que se ha realizado; el Capítulo 3 abarca las principales Tecnologías que se usaron para la parte de implementación y diseño; el Capítulo 4 está dedicado a la explicación del prototipo PALADIN junto con su arquitectura y las características necesarias para comprender su funcionalidad; el Capítulo 5, es similar a su capítulo anterior, ya que en él se explicarán las características y el funcionamiento del generador SDG; el Capítulo 6 se centra en el Desarrollo y la Implementación de los prototipos explicados en los Capítulos 4 y 5; el Capítulo7 va ligado a su capítulo anterior ya que muestra una Evaluación de los Resultados obtenidos; en el Capítulo 8 se realiza un avance a los posibles Trabajos Futuros que se podrían llevar a cabo aparte de las Conclusiones que se han obtenido tras este TFG; y, finalmente, en el Capítulo 9 se comenta un Análisis del Impacto que ha provocado este proyecto con respecto a diferentes ámbitos. Por otra parte, se hará una Bibliografía en la que se dará referencia a las posibles citas y referencias que se hayan usado para la realización de este TFG.

2 Trabajos Previos

Como anteriormente se ha comentado, el lenguaje de validación PALADIN ya se encuentra implementado para ser capaz de analizar bases de datos relacionales, como MySQL, y de grafos de conocimiento, más concretamente en Resource Description Framework (RDF)[21] y así devolver si los documentos recibidos son válidos o no teniendo en cuenta las restricciones definidas para el seguimiento que se esté realizando en el estudio. Para ello, se realizó el estudio de la arquitectura que iba a llevar a cabo para poder realizar los esquemas de validación junto con los esquemas de búsqueda que se realizarán para la comprobación de estos datos basados en procesos. Esta arquitectura consistía en dos componentes, el Shape Traversal Planner y el Shape Traversal Engine[1]. El primer componente nombrado se encarga de la generación de un plan de evaluación para las formas del esquema PALADIN; y el segundo, se encarga de la evaluación del esquema PALADIN siguiendo las restricciones de la estrategia de recorrido elegidas por el planificador anterior.

También, se han realizado análisis de los resultados recibidos por PALADIN entre los cuales se observó su comportamiento respecto a los test realizados, el tiempo de ejecución y las respuestas con datos reales junto con el tiempo de ejecución de PALADIN para las diferentes bases de datos. Estos test se llevaron a cabo mediante la utilización de un conjunto de testbenchs que se recopilaron cuando inicialmente se creó PALADIN para usar como caso de prueba. En ellos se encuentran datos limpios, sucios e intermedios entre ello aparte de encontrarse organizados por registros de 1000, 10000 y 100000 en el formato de bases de datos aceptadas para PALADIN. Para la evaluación de estos datos, el lenguaje de validación PALADIN se encuentra actualmente implementado para formatos de tipo base de datos relacional, más específicamente MySQL, y para grafos de conocimiento basados en Resource Description Framework (RDF)[21]; aparte de que pueden ser evaluados mediante diferentes algoritmos de búsqueda a elección: el Breadth-First Search (BFS), Depth-First Search (DFS) y con una búsqueda Recursiva (REC).

A parte, también se realizó una comparación respecto al tiempo de ejecución de los diferentes tests con otros prototipos con una funcionalidad similar y también capaces de realizar la validación de grafos y datos, como Trav-SHACL[3] y PyShEx y Shaclex (ShEx)[4]; aunque, según los resultados de estos análisis, la validación realizada por el prototipo SHACL con Shaclex fue el peor caso devuelto, es decir, el resultado con peor tiempo de ejecución de entre todos los prototipos utilizados.

Haciendo referencia a la otra parte del trabajo, también se llevó a cabo el desarrollo de un Generador Sintético de Datos (Synthetic Data Generator o SDG[2]), cuya licencia pertenecería al TIB (*The German National Library of Science and Technology*[2]), el cual también se encuentra implementado únicamente para devolver como resultados conjuntos de datos tanto relacionales como de grafos de conocimiento, es decir, en los mismos formatos en los que se encontraba desarrollado PALADIN añadiendo también la disponibilidad de exportar la base en formato CSV. Para poder realizar este generador se llevó a cabo un estudio sobre la pauta de tratamiento para los

pacientes con cáncer de mama con el gen amplificado HER2 y, tras esto, determinar los rangos aproximados entre los cuales se pueden encontrar los síntomas y tratamientos comunes que se suelen llevar a cabo en estos pacientes para así devolver un conjunto de bases de datos lo más parecido a los tratamientos correspondientes a posibles pacientes reales. También se ofrece un conjunto de esquemas de bases de datos sintéticos con las condiciones de validación necesarios para ser utilizados en PALADIN con casos de prueba que también ofrecen, aparte de ofrecerse otros esquemas para probar el rendimiento y la efectividad con árboles de diferente número de nodos desde 16 hasta 1024. El objetivo principal de este Generador es la rápida y precisa obtención de bases de datos sintéticas con información y variables lo más similares a la real para poder tomarla como referencia para tests con un margen de error óptimo.

3 Tecnologías usadas

Debido a que este TFG se desarrolla a partir de la ampliación de proyectos ya existentes en lenguaje de programación Python se decidió continuar este proyecto con la utilización de este lenguaje para facilitar la compatibilidad del proceso. Se ha hecho uso de los clusters que nos ofrecen MySQL y MongoDB. Para la base de datos de MySQL, se hizo uso del "MySQL Workbench" para la visualización de las tablas y la posible modificación de ellas además de para poder proceder a la conexión de dicha base de datos.

En cambio, en el caso del clúster de MongoDB se hizo uso principalmente de dos herramientas: MongoDB Atlas UI [5] y MongoDB Compass [6]. Ambas son útiles para mostrar el contenido de las diversas colecciones que se pueden almacenar en las bases de datos de MongoDB, aunque MongoDB Atlas UI es más completa ya que permite llevar a cabo el control de las conexiones con el clúster además de las configuraciones más personales de tu cuenta de MongoDB. Para acciones más sencillas como la simple visualización de datos es más cómodo usar MongoDB Compass, pero aun así ambas han sido utilizadas para el desarrollo de este proyecto, aunque solo es necesario el uso de una de ellas sin preferencia ninguna.

Otras herramientas que ofrece MongoDB que se hicieron uso, más concretamente para el almacenamiento de los diferentes casos de prueba usados para la ejecución de PALADIN, son `mongodump` y `mongorestore`, que forman parte del pack de sus Database Tools [6]:

- `mongodump`: exporta una base de datos en MongoDB, aunque también puede exportar específicamente una colección concreta e incluso una base de datos sin una colección especifica. Como resultado se obtiene en una carpeta por cada colección que contiene la base de datos un fichero BSON con el contenido de las colecciones que se han querido extraer, aunque no en un formato legible, y un fichero JSON que contiene los datos de la estructura y los índices de la misma colección.
- `mongorestore`: realiza la función contraria, importa una base de datos en formato de una carpeta con un conjunto de ficheros BSON y JSON en un clúster de MongoDB.

Por otra parte, y haciendo hincapié en la parte más técnica del proyecto, a la hora de programar se hicieron uso de diferentes tecnologías dependiendo de la parte del proyecto a la que nos refiramos.

3.1 PALADIN

Para la parte del proyecto principal que es la ampliación del lenguaje de validación PALADIN. Entre la parte que ya se encontraba desarrollada y la parte actualizada que se ha llevado a cabo, se hace uso de múltiples librerías (Figura 3) de las cuales las más esenciales son:

- MySQLConnection: perteneciente a myqsl.connector [7]. Se usa para realizar la conexión con la base de datos de un servidor MySQL en la que se comprobarán las validaciones. Gracias a ellos se permite el desarrollo de aplicaciones de bases de datos.

- SPARQLWrapper y JSON: pertenecientes ambas a SPARQLWrapper [8]. Se usa para realizar la conexión con los grafos de conocimientos en los que también se pueden comprobar las validaciones. Su aplicación principal es ser usado para poder ejecutar de una manera remota consultas y devolver de una manera más cómoda y organizada los resultados obtenidos para luego ser utilizados o almacenados. La parte de JSON de esta librería se usa para, en el caso de los grafos de conocimiento, tras la ejecución dar el resultado en este tipo de formato.

- MongoClient y PyMongoError: pertenecen a pymongo [9] y a pymongo.errors respectivamente. Su uso se basa en realizar la conexión con el clúster de MongoDB y para que en caso de que surja algún problema se emita una excepción de error. El uso principal de esta librería permite tanto la conexión con el clúster como la ejecución correcta de las consultas que se desean aplicar en las bases de datos que se encuentran en dicho clúster. Aparte, en caso de que haya algún error en la conexión a MongoDB, permite el envío de una señal de error que nos mantendrá informado en este caso de la razón por la cual se ha dado dicho error, ya sea porque no se ha realizado bien la conexión o porque la conexión se ha realizado durante un tiempo demasiado largo y ha tenido que ser detenida, entre otros errores posibles.

- json[10]: las funciones que tiene esta librería se usan para obtener los datos que se encuentran en el fichero JSON necesarios para realizar tanto las conexiones para obtener los datos como para obtener las consultas de validación. La librería json permite la lectura de las líneas de un fichero JSON del que se obtienen el formato de la fuente de datos que se va a querer analizar, los datos necesarios para la conexión a dicha fuente como su nombre, el nombre de usuario y contraseña, el número de puerto o el link uri de conexión; también, permite la lectura de las condiciones que se tienen como target y como validaciones para realizar las limpiezas de datos en estas fuentes.

Una de las partes de la implementación de PALADIN, se encarga de calcular el tiempo que tarda en llevarse a cabo la validación. En ellas, al ser solicitada este tipo de configuración, se generan, en caso de que no existan anteriormente, un conjunto de ficheros en los cuales se almacenan los datos sobre el rendimiento de la ejecución de las consultas. En un primer fichero, se encuentran las métricas de las diferentes consultas pedidas continuadas con su nombre, la configuración que se usó en dicha consulta y los tiempos que se tomaron en la ejecución. Por otro lado, en el segundo fichero, se muestran los datos de rendimiento de las validaciones que se han llevado a cabo en la última ejecución realizada. Para ello se hizo uso de algunas librerías a destacar, de las cuales las más importantes son:

- os.path[11]: se usa para diferenciar si alguno de los ficheros resultados anteriormente comentados se encuentra o no ya creados y, en caso afirmativo, simplemente abrir el fichero en vez de crear uno nuevo.

- time[12]: perteneciente a la librería time. Se encarga de controlar en un valor tipo float el tiempo que tarda PALADIN en ejecutar todas las validaciones solicitadas en su ejecución. Para ello, se almacena la hora exacta en la que se ha solicitado la traza y, cuando finaliza la ejecución, se vuelve a solicitar y, tras restarse ambos valores, da como resultado el tiempo que ha tardado el prototipo en ejecutarse.

```
import abc
import argparse
import json
import os.path
from enum import Enum
from functools import partial
from time import time

import mysql.connector
from SPARQLWrapper import SPARQLWrapper, JSON


from pymongo import MongoClient
from pymongo.errors import PyMongoError
```

Figura 3: Librerías importadas pertenecientes al fichero PALADIN.py. Fuente: [19].

3.2 SDG.py

Para la óptima ejecución de este programa, y que ya se encontraba implementado con estas utilidades, se requería de otras herramientas y librerías (indicadas en la Figura 4) entre las que se encuentra numpy[13], pandas[13] y rdfizer[14]. Estos son implementados para realizar la generación aleatoria de los datos a formar y para almacenar la información necesaria para más tarde ser usada por un Docker externo. Aparte, se utilizó junto a estas herramientas la librería de MySQL myqsl.connector[7] que también se usaba en la parte de PALADIN del proyecto.

Entre las opciones de ejecución, este generador también puede requerir del uso de un Docker externo o, en caso de necesitar un Docker, pero no tenerlo, se puede hacer uso de un programa generate.sh que se encarga de la creación de un Docker, la ejecución de SDG y, tras esto, de la eliminación del mismo Docker.

Para la parte del proyecto dedicada a la ampliación del generador de datos se hizo uso de una de las librerías que también se usaron en PALADIN.py, más

concretamente, la librería MongoClient[9] la cual se usaba para realizar la conexión al clúster de Mongo. Para esta parte no se requería el uso de un Docker.

```
import argparse
import csv
import datetime
import os
import time

import mysql.connector
import numpy as np
import pandas as pd
from mysql.connector.connection import MySQLConnection
from mysql.connector.cursor import MySQLCursor
from rdfizer import semantify

from pymongo import MongoClient
```

Figura 4: Librerías importadas perteneciente al fichero SDG.py. Fuente: [18].

4 PALADIN

Como se ha introducido en capítulos anteriores, en cualquier tipo de ámbito se hace uso del almacenamiento de datos, ya sea en casos médicos, como económicos, como informáticos, por ejemplo. Su finalidad principal es guardarlos hasta que en algún momento esos datos sean necesitados para cualquier tipo de utilidad, el problema más conflictivo es que en el momento de realizar búsquedas, en múltiples condiciones, dichos datos pueden ir variando por culpa a factores o fases de un procedimiento que se esté llevando a cabo con el tiempo, un ejemplo claro es en una situación médica en la que un paciente haya sido tratado con medicinas o haya sido operado anteriormente y requiera de atención médica, los encargados requerirán seguir un proceso determinado dependiendo del historial médico de dicho paciente. A este tipo de datos se les denomina datos basados en procesos, ya que las elecciones posteriores dependen de los resultados de un proceso que se haya realizado anteriormente por eso resulta más complicado llevar a cabo el análisis y comprobación específicos de ellos. Como solución se hacen consultas mediante la búsqueda de criterios comunes que se vayan confirmando en dichos procesos, para ello están lo que se conoce como lenguajes de validación.

La funcionalidad de los lenguajes de validación es encontrar una calidad y una fiabilidad en el conjunto de datos que se toma como objeto, pero en este caso el proceso de validación se llevaría a cabo mediante el uso de búsquedas con restricciones determinadas para el contexto que se dé. Entre los lenguajes de validación más destacados se encuentran: el Shapes Constraint Language o SHACL[15], el lenguaje Shape Expressions o Shex [4] y el SPARQL Protocol and RDF Query Language o solo SPARQL[16] pero estos se encuentran enfocados a la validación mediante restricciones de grafos de conocimiento y, los dos primeros, a la representación de esquemas y de un lenguaje concreto de grafos de conocimiento RDF. El claro problema es que aparte de este tipo de representación de datos, se pueden encontrar muchos otros tipos como los más comunes que son bases de datos relacionales o no relacionales entre otros. Para llevar a cabo una solución a este problema se creó PALADIN [1], un lenguaje de validación el cual recopila un conjunto de diferentes formatos aceptables a la hora de realizar una validación mediante restricciones, pero, a diferencia de otros lenguajes, consigue ofrecer como resultado un conjunto de los identificadores que cumplen las condiciones tras dichas validaciones además de la capacidad de que devuelva en número de identificadores que no las cumplen por cada consulta.

El prototipo PALADIN actualmente se encuentra funcional para datos almacenados como grafos de conocimiento en RDF y para bases de datos relacionales en MySQL que funcionan de manera completamente eficiente y con una ejecución rápida, pero no tienen por qué ser los únicos formatos que pueda aceptar. Para que PALADIN pueda mejorar se pueden llevar a cabo proyectos de ampliación para que este lenguaje de validación acepte más formatos de manera moderada con el tiempo, como es el caso de este TFG. Para el desarrollo de este trabajo se ha llevado a cabo la ampliación de PALADIN para bases de datos no relacionales orientada a documentos como es MongoDB[5]. Este tipo de base de

datos en concreto almacena datos en un formato similar al dado por ficheros JSON y BSON en los cuales los datos se basan en un grupo y un valor. Estos datos agrupados forman lo que se conocen como colecciones que un conjunto de ellas forma las bases de datos en MongoDB. La diferencia de MongoDB con otras bases es que dentro de uno de los datos de una colección se puede encontrar un documento o un array que haga referencia a documentos. Esto permite la representación de referencias a agrupaciones de mas datos que puedan ser necesarios para identificar y aumentar la información sobre una entidad almacenada.

4.1 Arquitectura y Configuraciones

El diseño de PALADIN está basado en el uso de árboles binarios, almacenados en dicho fichero JSON, para la comprobación de las consultas que se necesiten para validar ya que, en caso de necesitar realizar este análisis solo se dan como resultado entidades que la cumplan y entidades que no lo hagan. Estos árboles se encuentran organizados en una arquitectura en la cual en el nodo se encuentran la población o conjunto de todas las entidades en las que se quiere realizar la primera consulta con la restricción correspondiente a dicho proceso, tras esto, las ramas siguientes se convertirán en nuevas poblaciones que serán de nuevo sometidas a la siguiente restricción que se necesite y así se realizará sucesivamente hasta que se llegue a las últimas restricciones que se requieran para que se finalice la validación. Como resultado de este procedimiento, se devuelve el conjunto de entidades que cumplen o no con estas restricciones.

En este procedimiento, al comienzo de cada nodo siempre se tienen dos componentes; un target, que es la población objetivo a la cual se quiere aplicar las condiciones del segundo componente; y la validation, la condición que se aplicará a la población consultada en ese mismo nodo. Generalmente, los nodos posteriores al primer nodo u nodo raíz vienen condicionados de los resultados obtenidos por el primero, de esta manera, el número de instancias de las poblaciones posteriores a la primera se verán condicionadas por las validation que se vayan realizando. Tras esto y desde una visión más general del árbol, las instancias que cumplan la validación que se da en un nodo continuaran como población para el subárbol izquierdo que se genera y, por la misma razón, las instancias que no cumplan las validaciones continuaran como población para el subárbol derecho. Por ende, en el siguiente nodo se tomará como población la unión entre los resultados obtenidos tras la validación del nodo anterior y el target que se ha asignado para este nuevo nodo. Dando lugar así, a un árbol en el cual las entidades que se hayan obtenido como resultado tras las validaciones realizadas por todo sub-árboles izquierdos, implicarán que hayan cumplido todos los requisitos y que serán las entidades más correctas y óptimas para el seguimiento que se esté realizando. Por esta misma razón, aquellas entidades que se hayan obtenido todo por subárboles derechos serán aquellas que no cumplan ninguna de las restricciones a las que han sido condicionadas; aparte en cada validation se realiza esta división entre instancias que la cumplen e instancias que no dando así lugar a otros subárboles izquierdos y derechos

intermedios. Un ejemplo de representación simple de un esquema de validaciones es el que se puede observar en la Figura 5.

Un ejemplo claro es el que se puede observar en el artículo sobre PALADIN[1] en el cual se toma como referencia el cargo de un Profesor catedrático de universidad del cual se han almacenado su historial de cargos en los que ha sido participe como, por ejemplo, que perteneció al grupo de profesores asistentes y que fue profesor titular. Aparte de ellos se tuvo que guardar el historial de posiciones y cargos anteriores en los que este profesor estuvo participe, de manera que para realizar una búsqueda de si este profesor era un buen candidato a ese ascenso se realizaría un esquema de validaciones en la cual, si todas se cumpliesen, el profesor sería uno de los mejores candidatos a elegir. La diferencia entre otros lenguajes de validación basados en procesos y PALADIN es que en lenguajes de validación como por ejemplo SHACL[15], teniendo en cuenta que la información dada anteriormente se encuentra representada en grafos de conocimiento, esta requiere del almacenamiento y del registro de todo el historial de posiciones y cargos anteriores para analizar el supuesto ascenso, pero, mediante el uso del prototipo PALADIN no es necesario esto ya que en el target del nodo raíz ya se realiza la búsqueda por igual de todos los empleados que hayan pertenecido a los grupos requeridos para este ascenso. De esta manera, según un esquema PALADIN, de todos dichos empleados que se toman como población del nodo raíz se pondrá como validación que anteriormente se haya sido profesor asociado o asistente y, en caso afirmativo, se continuará por la rama izquierda del árbol implicando que los empleados restantes son profesores catedráticos y, en caso contrario, se continuará por la rama derecha. Continuando con la rama de incumplimiento de la condición (la rama derecha), se tendrán como target los empleados que no son profesores catedráticos y de validación si anteriormente han sido profesores asistentes; en este caso, los que cumplen con la validación implicará que son profesores asociados y continuarán por el subárbol izquierdo y, en caso contrario, si incumplen la validación implicará que las entidades restantes son profesores asistentes y continuarían por el subárbol derecho. Se puede ver un esquema de validaciones PALADIN correspondiente a este árbol de ejemplo en la Figura 6.

Según este ejemplo, en un caso de validación mediante el uso de PALADIN, una de las validaciones en la cual se comprueba si un profesor ha sido anteriormente asistente sería una consulta en la cual se comprobaría si la fecha de comienzo del cargo de profesor asociado de los empleados es posterior a la fecha correspondiente al cargo de asistente. De esta manera, mediante consultas con PALADIN relacionadas entre sí como la anterior se puede hacer una comprobación de que los datos almacenados dentro de una base de datos correspondan con las condiciones realizadas a modo de validación dando lugar así a una limpieza de aquellas bases que no las cumplan.

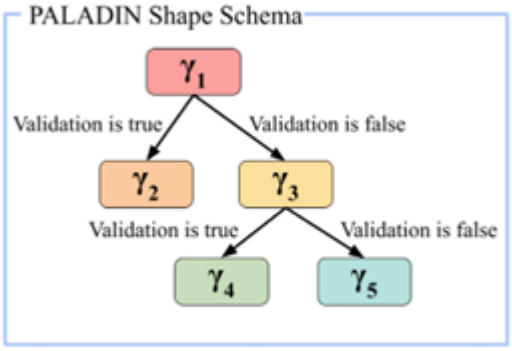

Figura 5: Representación de un ejemplo de esquema simple de un árbol de validaciones PALADIN. Fuente: [1].

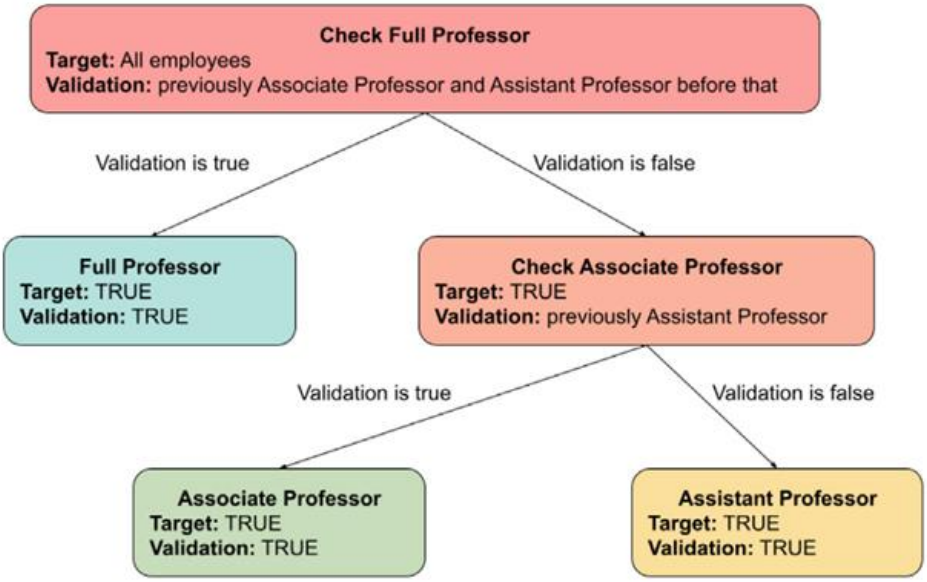

Figura 6: Esquema de árbol según validaciones con PALADIN. Fuente: [1].

Para el correcto funcionamiento de PALADIN, se quiere un conjunto de datos que se encuentre almacenado en alguna base determinada, ya sea en algún cliente de MySQL o en un clúster de MongoDB, ya que la primera acción que llevará a cabo PALADIN será la conexión a una base de datos donde se almacena la población que se desea tomar como entrada. Por otro lado, se necesita un fichero en JSON en el cual se encuentre la estructura completa del árbol con el conjunto de validaciones y targets que tomar como población que se deseen aplicar en cada nodo y las configuraciones necesarias para la conexión anteriormente comentada. Tras esto, se deberá seleccionar uno de los tres tipos de algoritmo de búsqueda que se desea realizar en el árbol recibido, dichos algoritmos son BFS, DFS y REC. En los algoritmos BFS, se realiza la búsqueda de las validaciones del primer nivel del árbol y, después de finalizar con las restricciones de ese nivel, realiza todas las restricciones que se lleven a cabo en la siguiente capa; este tipo de búsqueda suele ser las que menos tardan en

ejecutarse debió a que no tiene que realizar tantos movimientos hacia delante o hacia detrás en el propio árbol si no que poco a poco va avanzando. En el otro caso de algoritmo DFS, se van comprobando si todas las entidades van cumpliendo todas las consultas solo de una rama, después de acabar con la rama por completo, retrocede en el árbol al nodo contrario más antiguo que se haya comprobado de dicha rama y continúa con la rama que impliquen las validaciones de esa nueva rama; y así sucesivamente hasta que se no queden más ramas. En la Figura 7, se pueden observar dos representaciones diferentes en las cuales se ve el recorrido que se realiza tanto para búsquedas con algoritmos BFS y con algoritmos DFS.

Teniendo en cuenta el rendimiento entre estos dos algoritmos, la opción de DFS suele tardar más en ser ejecutada ya que se suele utilizar para encontrar todas las validaciones posibles del árbol. Por último, se encuentra la opción de búsqueda de recursividad (REC) que es una búsqueda similar solo que a la hora de ser implementado se hace mediante el proceso de recursividad; por ello, sus resultados de ejecución suelen tener poca diferencia en comparación con los DFS.

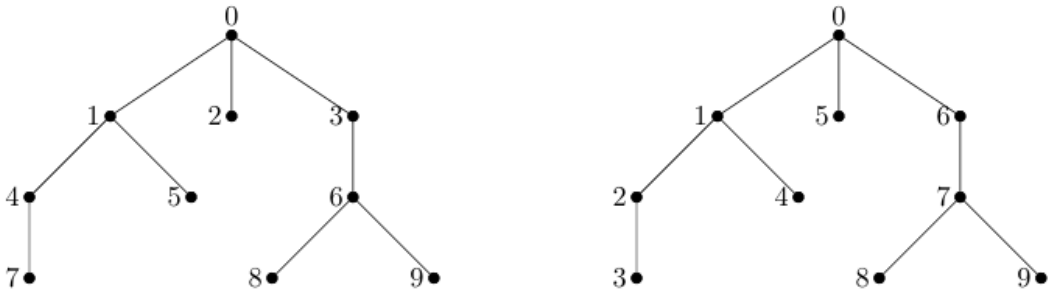

Figura 7: Diferencias entre BFS (izquierda) y DFS (derecha) en árboles binarios. Fuente: [17].

4.2 Casos para Uso de Prueba

Para probar el comportamiento y la correcta ejecución de PALADIN, se puede trabajar con una agrupación de diferentes conjuntos de bases de datos en los cuales la información almacenada se basa en un estudio realizado entorno a los tratamientos realizados a pacientes con cáncer de mama que presentan un gen HER2 amplificado[2]. Entre los datos que se almacenaban, se incluyen diagnósticos, tratamientos de quimioterapia, antecedentes familiares, datos demográficos e información sobre pautas de quimioterapia y datos sobre medicinas. Como se puede observar en la Figura 8, en este caso de prueba todos los datos que se generan dependen del historial del paciente que es la entidad central del diagrama. Los pacientes se encuentran relacionados de manera directa con los esquemas de quimioterapia del cual el dato más importante que se tiene en cuenta es la fecha en la cual fue aplicada dicho tratamiento, ya que de este se obtendrán las medicinas que van asociadas a él. De manera más secundaria pero también importante, se encuentran datos sobre el paciente como el número de operaciones y cuales a las que ha sido sometido; si en su familia tiene antecedentes de pacientes con enfermedades que puedan ayudar

en la futura pauta médica que se deba llevar, en caso de que algún familiar haya tenido algún tipo de tumor y se pueda prevenir su aparición antes de que se llegue a desarrollar; también se almacenan datos como la fecha en la que comenzó y en la que dejó de ser sometido a radioterapia, aparte de la dosis que se le dio y el número de estas sesiones acabó recibiendo; otro dato que se produce es si presencia dos o más enfermedades al mismo tiempo aparte de cuales; otro dato relacionado con el paciente es si toma medicinas orales y, en caso afirmativo, cuales y que tipo de medicina es; por, otro lado, otros datos generados son aquellos relacionados a los tumores que se pueden llegar a tener, entre los cuales los que más influencian son el tipo de tumor, el número de tumores y el tamaño del tumor. Siendo más específicos, el tamaño del tumor puede afectar a la pauta médica a aplicar ya que, antes de realizarse una operación, en ocasiones, se somete al paciente a un tratamiento de una medicina determinada cuya función es la reducción del tamaño del tumor; por esto, se realiza también un seguimiento del tamaño tanto antes de la operación como después. En caso de que se le haya aplicado esta medicina se le dice que el paciente ha recibido tratamiento neoadyuvante. Muchos de estos datos, también son diagnósticos más corrientes como la edad a la que se tuvo la primera regla y la menopausia; si el paciente ha tenido algún embarazado y, en caso afirmativo, si sucedió un aborto, si se llevó a cabo el parto o si el parto fue por cesárea; la fecha en la que se le fue diagnosticado el cáncer de mama, junto con la edad que tenía cuando fue diagnosticado y el fecha de su primer tratamiento; y datos más generales como el número identificador de su historia clínica electrónica (ehr) y su fecha de nacimiento. Estos datos no solo se encuentran generados basados en pacientes vivos en la fecha de su generación, sino que también se pueden encontrar casos de pacientes que fallecieron debido a complicaciones con esta enfermedad de manera que, también se almacenan en caso de fallecimiento, la fecha de muerte y la edad del paciente a la que falleció.

Con el conjunto de todos estos datos, se generan las bases de datos que se podían usar como caso de prueba en este proyecto y que, gracias a anteriores trabajos, dieron lugar a un conjuntos de bases de diferentes tamaños y de una probabilidad de contener errores en ellos diferente, diferenciándose así las dirty, aquellas que contienen mayoritariamente datos erróneos y que no cumplen las pautas necesarias para este estudio; las clean, aquellas que contienen plenamente datos correctos en cuanto a este contexto: y las mid, que son dato intermedios entre las dirty y las clean.

En concreto, se tomaron de ejemplo las 9 bases de datos en MySQL que se ofrecían (en el artículo [2]) de un total de 18 (las 9 bases restantes correspondían a grafos de conocimiento) cuya información fue traspasada a bases de datos no relacionales en MongoDB, aunque algunos tipos de los datos de la primera base no eran del todo compatibles para MongoDB, pero dichos errores fueron corregidos de manera legible para ambas bases. Estos conjuntos comprendían rangos de entidades de 1000, 10000 y 100000 formadas cada uno por separado de datos sucios, datos limpios y una mezcla entre datos limpios (a los que se les denomino mid), gracias a esto se pueden tomar como ejemplo también dichas bases de datos en un formato de base de datos más. Aparte de estos casos de prueba, también se ofrecía en este estudio los esquemas necesarios para la

comprobación y validación de una nueva implementación en la cual se encuentran los árboles en JSON con las restricciones hechas específicamente para este caso de prueba. Cada nodo comprobaba la validación de una parte del historial de los pacientes que se encontraban en las bases, de manera que cuan más adentrado en el árbol se encuentre el paciente, habrá salido validado por mayor número de restricciones siendo los mejores candidatos para el tratamiento que se quiera aplicar a dichos pacientes. Este JSON también fue traspasado a filtraciones aceptables para búsquedas en bases en MongoDB usando como ejemplo el esquema para datos sintéticos en MySQL. Aparte de este fichero JSON se encontraba otro esquema con filtraciones con la funcionalidad de comprobar la escalabilidad de PALADIN, entre todos ellos se encuentran un total de 7 ficheros más, en concreto aquellos en los que cuyos árboles tienen un total de 16, 32, 64, 128, 256, 512 y 1024 nodos. Como se puede, entender a menos nodos, menos tarda la ejecución de dicho prototipo.

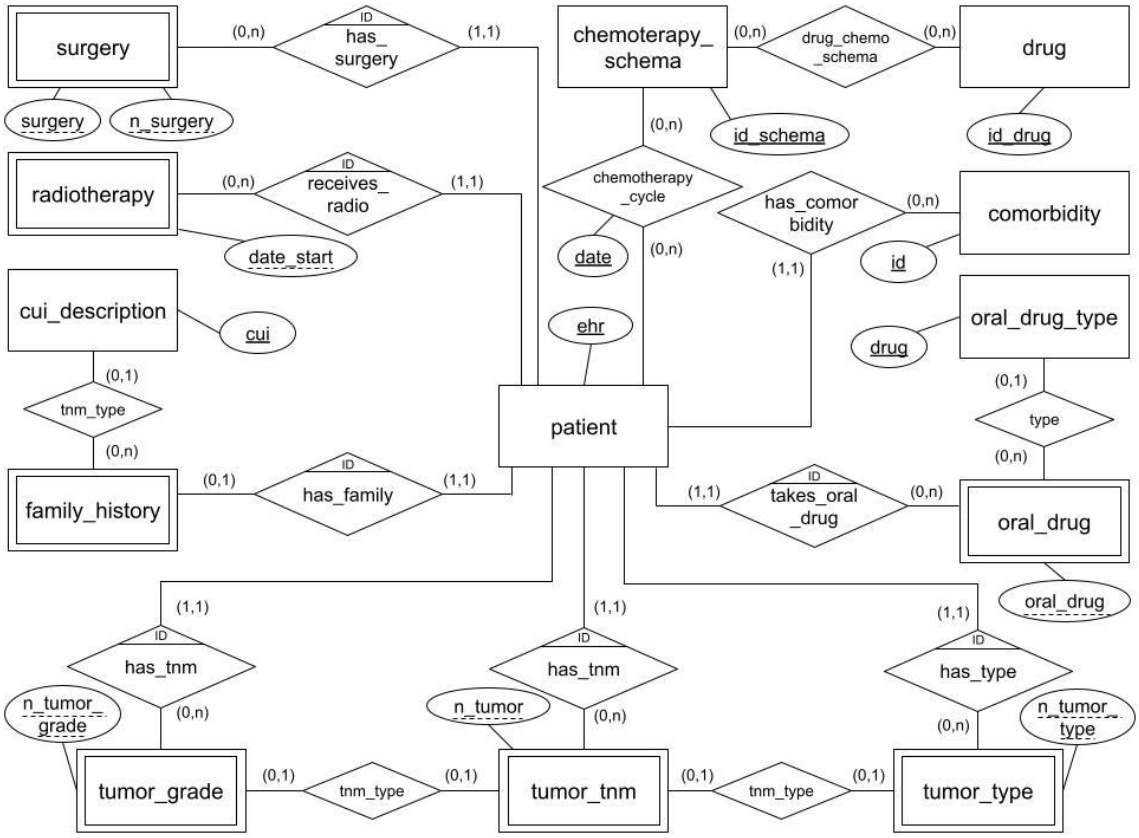

Figura 8: Diagrama Entidad/Relación de los datos generados por SDG. Fuente: [18].

4.3 PALADIN con MongoDB

Como se ha comentado en el subapartado anterior, se necesitó el traspaso a este formato de diversos ficheros para comenzar con el desarrollo de esta parte del trabajo. Para ella se requería desde un principio de un clúster para el almacenamiento de dichas bases que fue implementado utilizando MongoDB Atlas UI [5] aunque, para probar la correcta visualización de los datos y para comprobar en ocasiones que la conexión no estuviese dando error también se hizo uso de la herramienta MongoDB Compass[6]. Estas herramientas permitían conectar y desconectar el clúster, eliminar y añadir tanto colecciones como bases datos además de modificar datos concretos si se deseaba, hacer búsquedas basados en filtros y agregaciones o exportar bases de datos e importarlas de desde pares de ficheros JSON y BSON, aunque para realizar esto también se podía necesitar de otra adición externa denominada MongoDB Database Tools[6]. Esta última permitía la utilización desde la terminal del ordenador de comandos específicos para dichas exportaciones e importaciones de bases entre un directorio del ordenador y el clúster de MongoDB.

Tras esto, la ejecución de comprobaciones de MongoDB en PALADIN daría los resultados deseados para todo tipo de configuración que se pueda realizar respecto a los datos obtenido de los casos de prueba en MySQL.

4.4 Synthetic Data Generator

Otro proyecto aparte, pero ligado a PALADIN, era el Synthetic Data Generator o SDG[2], [18] que, como su propio nombre indica, consiste en un generador de datos sintéticos para uso de caso de prueba de PALADIN. Este se creó mediante distribuciones de datos reales de pacientes con cáncer de mama con el gen amplificado HER2 como se explicaba en el capítulo 4.2, en el cual se detalla de que trataba dicho estudio. Como los datos obtenidos en el ámbito de las medicinas son muy sensibles y debido a motivos éticos y legales, no se deben poner a disposición libre del público de esta manera con SDG se permite trabajar con datos sintéticos lo más similares a los reales.

Este generador actualmente ya ofrece 18 conjuntos de estos datos, 9 para bases relacionales y otros 9 para grafos de conocimiento RDF[21] de los cuales todos se encuentran en diferentes tamaños (1000, 10000, 100000 pacientes) y en diferentes probabilidades de mutación entre los datos (clean, mid y dirty; siendo mid una opción intermedia entre las otras dos opciones de probabilidades)

4.4.1 Arquitectura y funcionamiento

El SDG, como se determina en el siguiente artículo [2], se creó con el fin de ofrecer un conjunto de testbenchs con datos lo más similares posibles a la realidad para probar el prototipo PALADIN y así, poder comprender su funcionamiento y utilizarlo para estudios y búsquedas reales. Para ellos se almacenan datos aleatorios (cuyo número de entidades, en este caso pacientes, se selecciona a la hora de su ejecución) dependiendo de la probabilidad de mutación (la probabilidad de que los datos generados sean limpios o sucios siendo la probabilidad 0.0 correspondiente a datos completamente limpios y con sentido en los cuales se encontrarán pacientes cuyos características sean más cercanas para el uso del tratamiento que se investiga en este estudio) que se desee entre un rango de valores que dependen de la probabilidad en el que se puedan encontrar dichos datos, es decir, por ejemplo que una persona haya sido operada anteriormente de una mastectomía se encuentra indicado con una probabilidad del 50% ambos por lo que a la hora de generar estos datos será afectado por dicho porcentaje; así y teniendo en cuenta otros factores como los medicamentos a los que han sido sometidos o las fechas de sus primera y última operación se acaba generando una base de dados que principalmente se almacena en formato MySQL.

Desde la conexión de esta base se aprovecha para pasar a formato CSV y almacenarlo en una carpeta en la ubicación \data donde también se guarda un fichero comprimido ZIP con los datos en MySQL y; tras esto, se realiza una conexión a MongoDB para aprovechar la primera base e importar los datos en formato no relacional. Finalmente, se realiza un mapeo y se traspasan a formato de grafos de conocimiento RDF.

Por otra parte, SDG ofrece los esquemas en JSON necesarios para el uso de prueba en PALADIN. Entre ellos, se encuentran un total de 7 esquemas para cada tipo de formato, aunque como PALADIN todavía no se encuentra

implementado para el uso de datos en CSV no se ofrecen este tipo de esquemas para este caso. Uno de estos esquemas contiene las pautas y restricciones necesarias para el tratamiento de los pacientes con el gen amplificado HER2 y, por otro lado, los seis esquemas que quedan se hacen uso en caso de necesitar investigar la escalabilidad de PALADIN diferenciándose entre ellos por el número de nodos de los que se encuentran formados los árboles contenidos (16, 32, 64, 128, 256, 512 y 1024).

Finalmente, para la visualización de los resultados en PALADIN se ofrece un fichero comprimido con todos los conjuntos anteriormente especificados aparte de un script de ejecución que te permitirá hacer uso del prototipo y darte en una carpeta los ficheros con los resultados devueltos. Aunque PALADIN ofrece una opción en la cual se enseñan los resultados en modo de traza de la terminal, para usar el modo de ejecución anteriormente explicado se requiere de la instalación anterior de un Docker, en especial para la obtención de los datos desde el uso de conjuntos de grafos de conocimiento basados en RDF; además de que, en este caso, se requerirá también de la ejecución del fichero con permisos `sudo`.

5 Diseño e Implementación

Para la realización de este Trabajo de Fin de Grado se realizó la implementación de varios programas en lenguaje Python para hacer conexiones entre bases de datos y el paso de datos a otras bases de datos, además de la implementación del lenguaje de validación PALADIN asociado a bases de datos no relacionales. En este capítulo, se hablará de como resultó la implementación y el diseño de estos prototipos y su orden de funcionamiento.

Los programas implementados son SDG.py, el cual se encarga de realizar la trasferencia a diversos formatos desde la generación aleatoria de unos datos a diferentes tipos de bases de datos; y paladin.py, en el que se encuentra implementado los algoritmos de búsqueda y las configuraciones necesarias para realizar las validaciones deseadas en los diferentes formatos disponibles. A continuación, se especificarán los programas anteriormente introducidos aparte de otros programas que se implementaron externamente para facilitar la realización de este TFG.

5.1 Transformación de los Datos

Para el uso de prueba de PALADIN.py, gracias a trabajos previos en [2], ya se disponía de bases de datos con datos limpios, datos sucios y con datos medios. Estas bases de datos almacenan datos de tratamientos sobre pacientes de cáncer de mama basados en procesos de un estudio experimental realizado sobre datos reales de tratamientos de determinados pacientes de cáncer de mama con un gen HER2 amplificado.

Para comenzar con la implementación de la ampliación de PALADIN.py, se necesitaba una base de datos como las anteriormente dichas, pero en MongoDB para usarla como objeto de la validación. Como ya se encontraban en MySQL y grafos de conocimiento se decidió tomar las bases de datos en MySQL como base, ya que eran más similares a estas, y realizar un programa que realizase el paso a MongoDB. Para ello se implementó un programa auxiliar para facilitar el proceso.

La funcionalidad de este programa era transformar esa base de datos desde un servidor de MySQL a una colección de un clúster en MongoDB Atlas UI. En primer lugar, se realizará una conexión tanto a un servidor en MySQL como a un clúster de MongoDB. Para ello, requiere de los siguientes valores:

- Para la conexión al servidor MySQL necesaria para obtener la base de datos que queremos recibir como objeto, se necesita: en primer lugar, un “host” al cual conectarse; seguido el “user” o usuario correspondiente a dicho host y la “password” o contraseña de dicho usuario; y, por último, el nombre de la “database” que se quiere pasa a MongoDB. Se muestra un ejemplo de configuración para la conexión a una base de datos de nombre “clean_1000” para MySQL en la Figura 9.

```
"host": "localhost",
"user": "root",
"password": "contraseña",
"database": "clean_1000"
```

Figura 9: Ejemplo de configuración se conexión a la base de datos "clean_1000"

- Para la conexión al clúster de MongoDB se requiere de un link URI. Este se puede obtener, en caso de estar usando MongoDB Atlas UI en la opción de "*Connect*"- "*Drivers*" (la opción a seleccionar tras el botón de Connect es el que se puede observar en la Figura 10 y seleccionar el Driver *Python* para este caso y la versión que se tenga y, debajo de esto, se te dará el link URI listo para copiar y pegar en el programa.

  Además de este link URI, se necesitará crear una colección en tu clúster donde se guardará la base de datos tras el proceso de transcripción. Es recomendable que el nombre de esta colección se asemeje al nombre de la base de datos en MySQL para evitar confusiones más adelante, por ejemplo, "clean_1000".

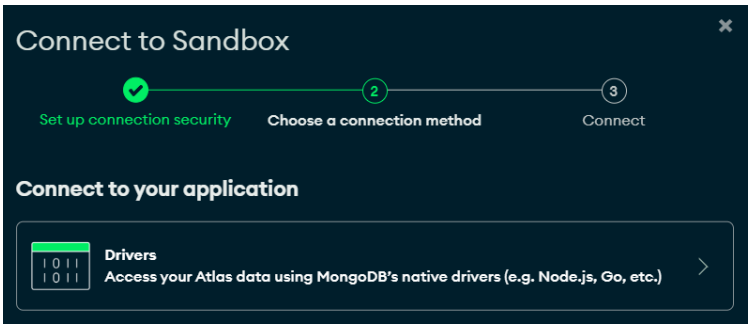

Figura 10: Opción que se debe seleccionar para obtener el "connection string" de un clúster de MongoDB[6].

Tras hacer la conexión con el servidor de MySQL, posteriormente, recoge los datos de las tablas de dicha base de datos especificada en el programa y se pasarán los datos a formato de MongoDB haciendo la conexión a un clúster de MongoDB en el que se encuentre la colección en la que se quieran añadir estos datos (también se especificará en el script). Tras coger los datos de las tablas de MySQL, se hace una búsqueda entre los datos para ver si alguno de ellos son de tipo fechas o binario en bit ya que, estos datos se encuentran implementados de una manera diferente en MongoDB y algunos de ellos pueden dar error.

Para realizar las transformaciones de los datos, se hizo uso de un programa externo que se creó con el objetivo de poder pasar bases de datos relacionales a no relacionales en MongoDB. Para comenzar, se tienen que especificar los datos de conexión para ambos formatos de fuente de datos que se ha especificado anteriormente y, tras estar conectados correctamente a ambas bases, se lleva acaba las transformaciones. En este proceso, se van listando las columnas de las tablas de la base relacional para luego seleccionar de esas columnas todos

los datos que se encuentran en ellas y así almacenarlas en una variable. Tras el traspaso de todas las variables tal cual se reciben, se realiza un filtro de algunos tipos de datos que pueden causar error al ser traspasados así tras el paso directo de una base a otra.

En el caso del tipo fechas, estas en MySQL se encuentran en el formato tipo texto YYYY-MM-DD, por ejemplo, 2025-05-16; en cambio, en MongoDB el formato se encuentra en tipo Date como objeto BSON tipo fecha, es decir, YYYY-MM-DDT00:00:00Z como se puede ver, aunque no se especifique la hora deja incluido el T00:00:00Z como predeterminado. Para traspasar este tipo de datos se tuvo que realizar una función auxiliar específica para el paso de este tipo de datos al formato correcto para MongoDB.

En el caso del tipo binario en bit en MySQL[20], al pasarlo a MongoDB[5] se pasa en formato "{campo: Binary.createFromBase64('AA==', 01)}" lo que en MySQL sería un 1 en caso de true. Para traspasar este tipo de datos se tuvo que realizar una función auxiliar específica para el paso de este tipo de datos a un formato en booleano para MongoDB. De esta manera, un dato tipo bit en MySQL b'\x01'/b'\x00' sería en MongoDB un dato tipo booleano true/false respectivamente. Ya que se intentó pasar a tipo binario de 1 y 0 pero el documento ocupaba demasiado y no podía mantener la conexión con el clúster para la transferencia completa de la base de datos.

Finalmente, tras traspasar todos los datos de las tablas de la base de datos en MySQL a la colección de MongoDB, se obtendrá en el URI especificado la base de datos SQL deseada en formato de una colección en MongoDB.

También se considera necesario especificar que, para la correcta transformación de los datos, se requería utilizar una configuración especifica al cursor en la cual este devuelve las columnas recibidas de la fuente de datos como clase diccionario. De esta manera, se almacenan como claves de cada columna el nombre de las tablas que, en el caso de MongoDB, corresponden con los nombres de las colecciones; además, dentro de ellas los datos serán contenidos tras su selección y corrección de errores de formato con valores de tipo tuplas, es decir, en pares clave-valor siendo la clave el nombre de la variable que se va a almacenar y, junto a ella, su valor correspondiente. Un ejemplo de tupla que se puede almacenar es el siguiente:

comorbidity: "cardiac insufficiency"

En él se puede observar que para la clave comorbidity (padecimiento de dos o más enfermedades simultáneamente) se le almacena el valor en string "cardiac insufficiency" (insuficiencia cardíaca). De esta manera varios de estas tuplas formarían un dato concreto sobre un paciente y, el conjunto de estos, una colección sobre una métrica determinada del estudio.

Parte de este funcionamiento se tomó como referencia para luego la implementación final del generador SDG en el cual también se lleva a cabo parte de este proceso de transformación de datos.

5.2 PALADIN.py

Este programa, como se ha explicado en capítulos anteriores, se encontraba pre-implementado para bases de datos relacionales y grafos de conocimiento y se tiene disponible en un repositorio de GitHub[19]. Su funcionalidad es obtener desde el fichero JSON los datos necesarios para realizar la conexión en la base de datos deseada entre las cuales, dependiendo de a qué tipo de base de datos se esté conectando, recibirá unos datos de inicio u otros, por ejemplo, en el caso de una base de datos en MySQL se usarían los datos dados en la Figura 11 y en el caso de una base de datos en MongoDB se usarían los datos dados en la Figura 12.


```
"process_id":"synthetic_mysql_dirty_small",
"data_source":"mysql",
"url":"localhost",
"port":3306,
"user":"root",
"password":"password",
"database":"dirty_1000",
```


Figura 11: Datos de ejemplo del fichero JSON para una base de datos en MySQL


```
"process_id":"synthetic_mongodb_clean_small",
"data_source":"mongodb",
"url":"mongodb+srv://<user_name>:<db_password>@sandbox.<cluster>.mongodb.net/",
"database":"clean_1000",
```


Figura 12: Datos de ejemplo del fichero JSON para una base de datos en MongoDB

Dependiendo del valor que se encuentre en la variable “data_source” del fichero JSON se dirigirá a la función correspondiente al tipo de base de datos que es haya seleccionado siendo el valor “mysql” para bases relacionales en MySQL, “sparql-endpoint” para grafos de conocimiento y “mongodb” para bases no relacionales en MongoDB. Tras esto y generalizando para las tres clases, como se puede ver representado en la Figura 13, se reciben los datos principales del JSON para realizar la conexión a la base de datos que se haya determinado y, tras esto, se realiza la primera consulta que determina el target u objetivo de las validaciones que se harán a continuación. A continuación, se realiza la primera validación del árbol, los datos que son validados se llevarán a una rama del árbol y los datos que no cumplen la validación se llevarán a la rama contraria y así sucesivamente hasta realizar todas las ramas. Los datos que se encuentren validados en las ramas más lejanas serán aquellos que cumplen la validación completamente y serán, según el ejemplo del estudio en pacientes con cáncer de mama con el gen HER2 amplificado, los óptimos para realizar el tratamiento determinado ya que son aquellos que cumplen el mayor número de condiciones en su historial médico.

Para el caso de la amplificación de PALADIN.py para bases de datos no relacionales en MongoDB, que era la objetivo en este TFG, se realizó una nueva clase en el programa en el cual se recibían los datos desde el JSON como la URI, el nombre de la base de datos el nombre de la colección de MongoDB que se va a usar como base y la population desde la que se va a hacer la primera consulta. Tras esto realiza la conexión a la base de datos de MongoDB desde la que

realizará las querys que se piden a modo de validaciones desde el documento JSON añadiendo cada respuesta en una lista y eliminando los datos repetidos.

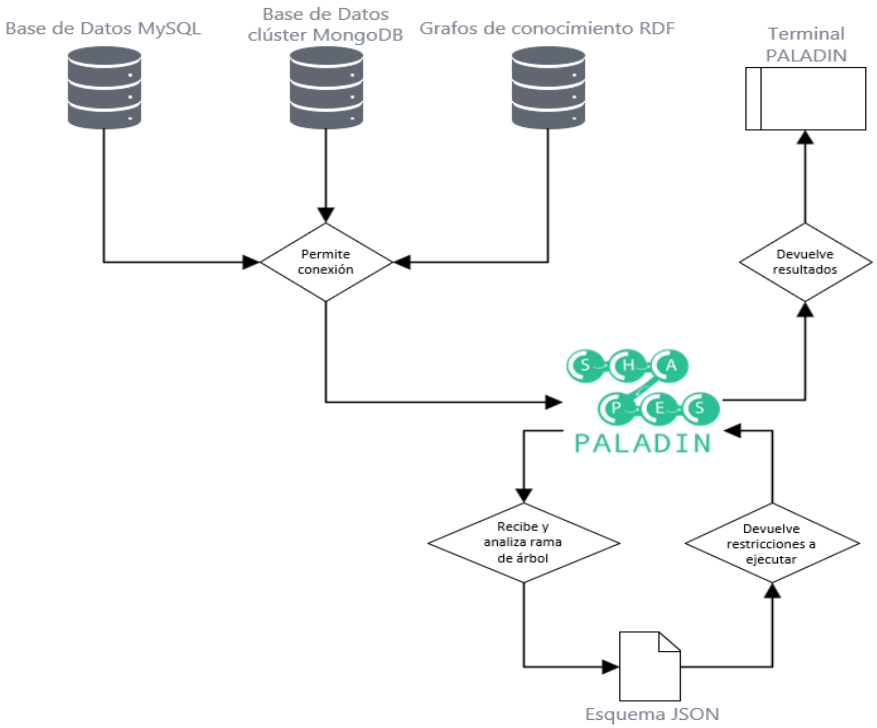

Figura 13: Diagrama de funcionamiento de PALADIN.

5.3 SDG.py

Tras la implementación de PALADIN y la confirmación de que todos los resultados que daban eran correctos, se tomó la decisión de realizar también la amplificación del Synthetic Data Generator para bases en MongoDB. La función principal de este programa es la generación de datos basados en el estudio anteriormente comentado y pasarlo a bases de datos en MySQL. Tras esto, recibe dicha base y la pasa a formato CSV y grafos de conocimiento basados en RDF. Y, tras esta ampliación, ahora a MongoDB.

Para ello, se tomó de referencia el programa comentado anteriormente y que fue desarrollado al comienzo del proyecto. De esta manera, solo se necesitó crear una función externa que realizase la conexión a la base en MongoDB y exportase los datos de la base en MySQL a la base en MongoDB; aparte, se necesitó añadir otras funciones auxiliares para el correcto traspaso de ciertos tipos que se usaban entre los datos de los pacientes (como ya se explicó en el Capítulo 6.1).

Esta implementación requería la adición de una librería necesaria especialmente para la conexión con el clúster de MongoDB, y ya explicada

anteriormente en el capítulo 3.2 referente a las Tecnologías Usadas, MongoClient.

Como se puede observar en la Figura 14, el funcionamiento del SDG es bastante lineal. En el todo depende de la generación correcta de los datos y su almacenamiento en la base de datos en MySQL, de la cual se tomará como base para el tras paso de diferentes formatos tras hacer conexiones con algunos de ellos. Finalmente, se obtienen como resultado múltiples fuentes de datos en diferentes formatos, pero todas con los mismos datos almacenados.

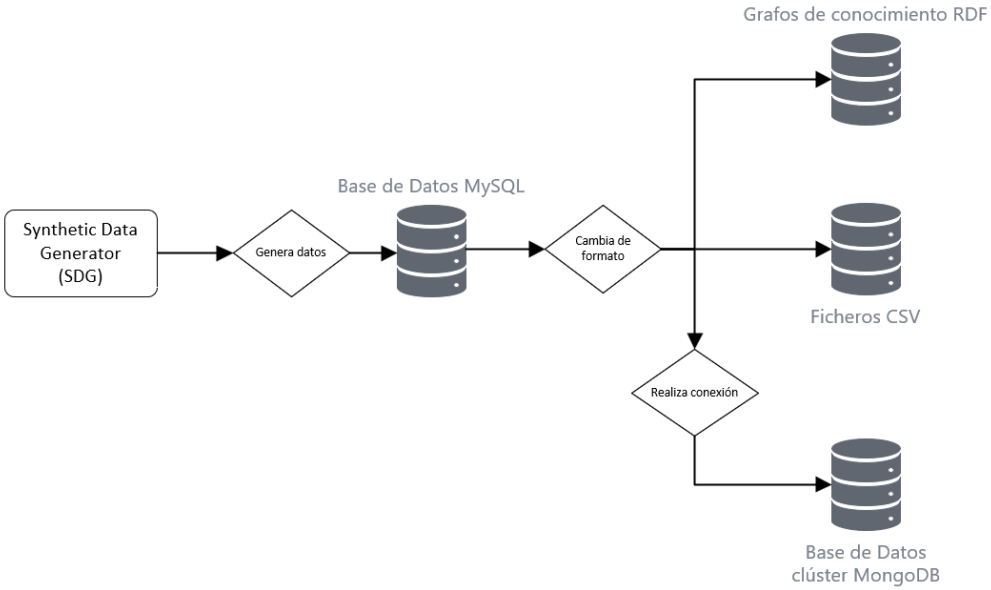

Figura 14: Diagrama de funcionamiento de generador de Datos sintéticos (SDG).

6 Resultados y Evaluación

Tras la ejecución y diseño de los prototipos comentados en capítulos anteriores de este trabajo, se realizaron diversos análisis tanto de los resultados que se imprimían por pantalla, como de los resultados de tiempo de ejecución comparados con otras configuraciones disponibles. En esta parte se realizará la comparación del diseño en MongoDB de estos prototipos con las otras opciones de ejecución que ya se encontraban disponibles, así como se comentará sus diferencias de resultados con otros lenguajes similares.

6.1 Resultados PALADIN

Como ya se ha explicado en el capítulo PALADIN4, para poder ser ejecutado y obtener los resultados de validación, se requiere de un fichero JSON y especificar el algoritmo de búsqueda que quiere que se realice en el árbol de consultas almacenado en dicho fichero. Un ejemplo de comando de ejecución sería:

```
!python paladin.py -p synthetic_mongodb_clean_small.json DFS
```

En este ejemplo, synthetic_mongodb_clean_small.json es el nombre del fichero JSON con los datos necesarios para la conexión a una base de datos en MongoDB con datos limpios y el árbol de consultas que se quiere realizar según el estudio, DFS (Depth-First Search).o será una de las opciones de algoritmos de búsqueda que se ofrecen para realizar la búsqueda (también se puede usar BFS (Breadth-First Search) y REC (recursivo) y, por otra parte, se encuentran varias opciones con las que ejecutar PALADIN: “-h” que muestra ayuda sobre como ejecutar el programa; “-p” que permite imprimir los resultado de validaciones por la terminal; y “-t” que permite mostrar los resultados sobre la ejecución del programa dos ficheros aparte metrics.csv y trace.csv que lo que hacen es que el primero almacena los traces que se han llevado a cabo y los trace.csv almacena los datos de tiempo de ejecución de cada búsqueda que se pide.

Para la prueba de PALADIN[2] se hizo uso de un conjunto de bases de datos que se encontraban con un número de pacientes de 1000, de 10000 y de 100000, aparte de que cada uno de estos conjuntos se tenían con datos limpios, sucios y casos intermedios de limpios y sucios a los que se les llama clean, dirty y mid en los casos de prueba. Para cada uno de estos conjuntos aparte se podían obtener resultados de cada uno de los algoritmos de búsqueda (BFS, DFS y REC).

Para analizar los resultados correspondientes a PALADIN se estudiaron por separado tanto la funcionalidad por terminal de su ejecución como el tiempo que tarda en ser ejecutado por completo. Haciendo referencia al primer modo de análisis de los resultados, entre los diferentes tipos de datos que se encuentra en las bases de datos, se obtuvieron diferentes resultados tras su

ejecución de manera que, en los casos en los cuales todos los datos eran clean y la probabilidad de mutación por ello era 0.0, todas las entidades entregadas como objeto validaban las pautas que se enviaban; y, por el caso contrario, aquellas que almacenaban datos dirty pocas de ellas validaban las restricciones a las que eran sometidas. Aparte, dependiendo del modo de búsqueda que se desease aplicar al árbol, se mostraban diferentes resultados tras su ejecución. En el ejemplo de la Figura 15, se puede observar un caso claro de la opción de búsqueda por BFS con datos clean, en ellos no hay ningún paciente el cual no valide las restricciones comprobadas ya que todos sus datos contienen valores que los haría probables candidatos para el uso de las pautas de tratamiento para este estudio en concreto.

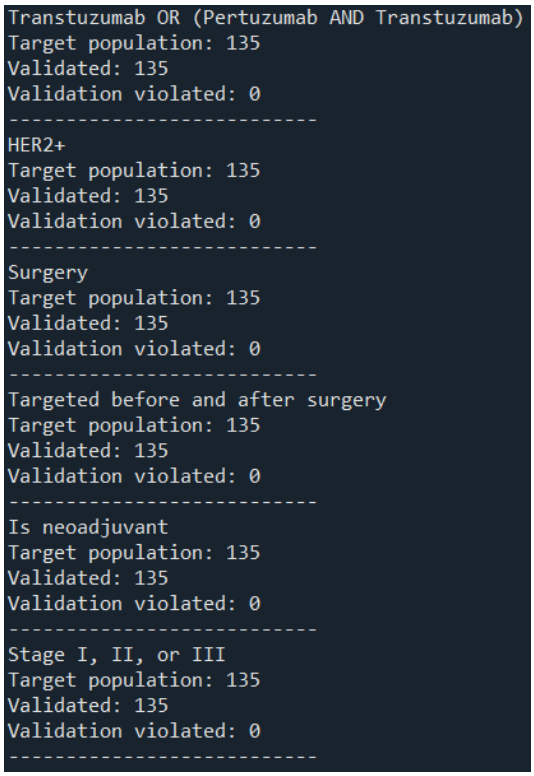

Figura 15: Resultado en terminal de paladin.py para bases de datos en MongoDB de 1000 con datos limpios con BFS.

Otro ejemplo que comparar con este, pero también con la opción de búsqueda por BFS, es el caso de datos dirty que se puede observar en la Figura 16. En él se puede observar que aparte de las restricciones que se observaron con los datos clean también se encuentran las comprobaciones con los nodos en las que salen datos no validados. De esta manera, se van recorriendo todos los nodos del árbol hasta llegar a los nodos finales en los cuales se observa que el número de pacientes que las valida son 1 en comparación con los 831 que se tenía la comienzo como población de entrada.

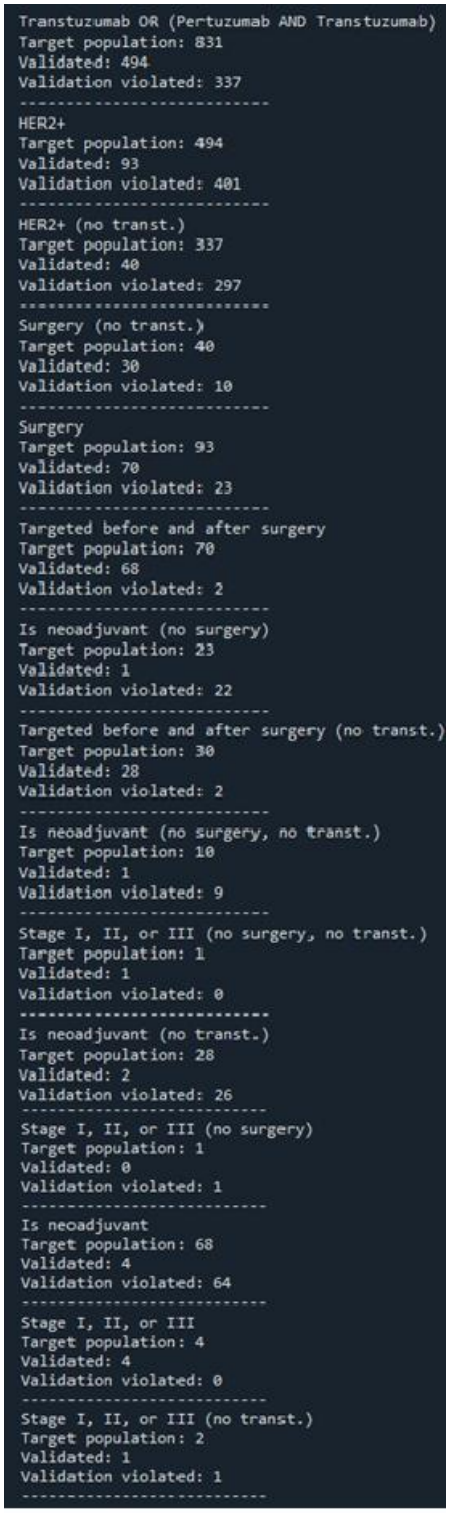

Figura 16: Resultado en terminal de paladin.py para bases de datos en MongoDB de 1000 con datos dirty con BFS.

En otro caso de ejemplo similar, se mostrará una búsqueda DFS de una base de datos mid_1000, es decir, con 1000 datos de pacientes registrados en los cuales los datos tienen una probabilidad intermedia de mutación. En este caso, se obtendría el resultado impreso por terminal que se puede observar en las Figura 17 y Figura 18. Como se puede observar, en este caso se muestran por pantalla solamente todos aquellos pacientes que van cumpliendo las validaciones, aunque también acaba recorriendo el árbol por completo siguiendo el orden de búsqueda de los algoritmos DFS.

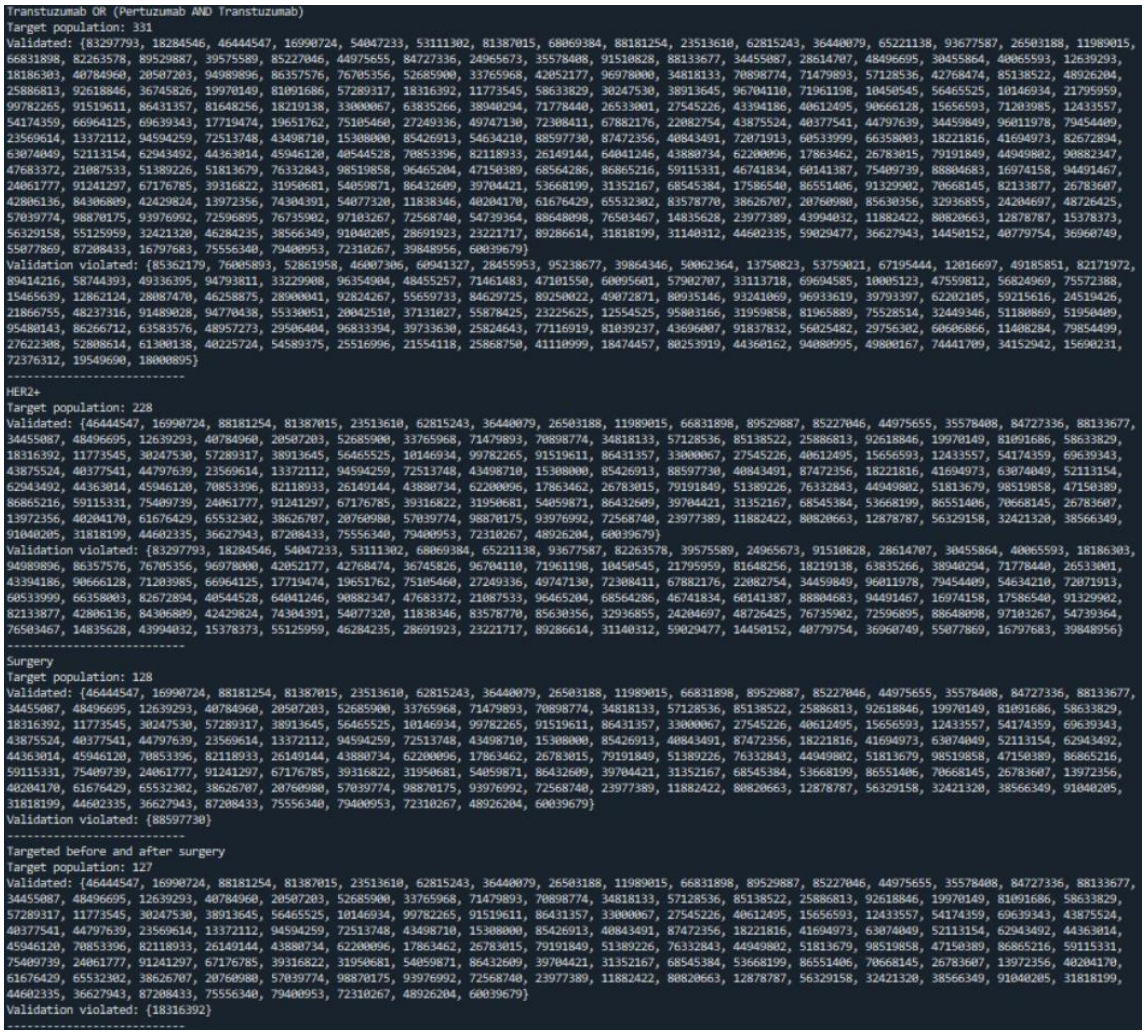

Figura 17: Resultado en terminal de paladin.py para bases de datos en MongoDB de 1000 con datos medio limpios medio sucios con DFS (Parte 1).

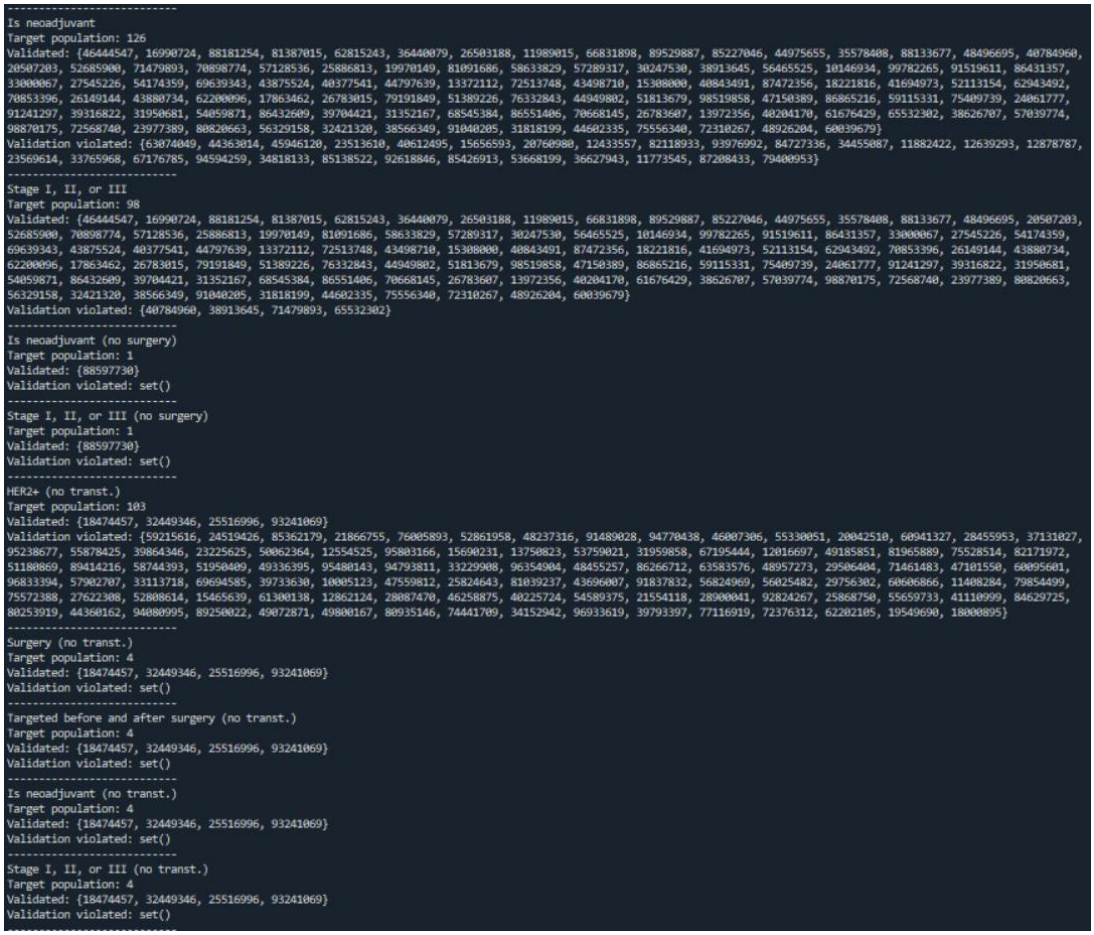

Figura 18: Resultado en terminal de paladin.py para bases de datos en MongoDB de 1000 con datos medio limpios medio sucios con DFS (Parte 2).

A parte de los esquemas con restricciones para datos sintéticos, también se podía comprobar los esquemas de escalabilidad en los cuales cada uno tenía un número de nodos. Su función principal era ir comparando sus tiempos de ejecución para ir analizando el rendimiento de estos dependiendo de la cantidad de validaciones que se van haciendo, como resolución se obtuvo que al ir aumentando el número de nodos más tardaba en realizarse la ejecución. Teniendo en cuenta esta métrica, se puede utilizar para compararlo con otras opciones disponibles de PALADIN[1] como con grafos de conocimiento[21] y con bases relacionales[20]. Como se puede observar en la Figura 19, la configuración con la que menos se tarda es en la cual se utilizan bases relacionales (Relational Data Bases o RDB) teniendo la opción en la cual se usan grafos de conocimiento (KG) como segunda configuración más rápida; por último, los resultados tras la ejecución para bases no relacionales indicaban que sería la tercera opción más rápida en cuanto tiempo de ejecución comparada con las anteriores. Esta opción daba un tiempo de ejecución aproximado de entre 60 y 80 segundos teniendo un tiempo medio aproximado de 75 segundos.

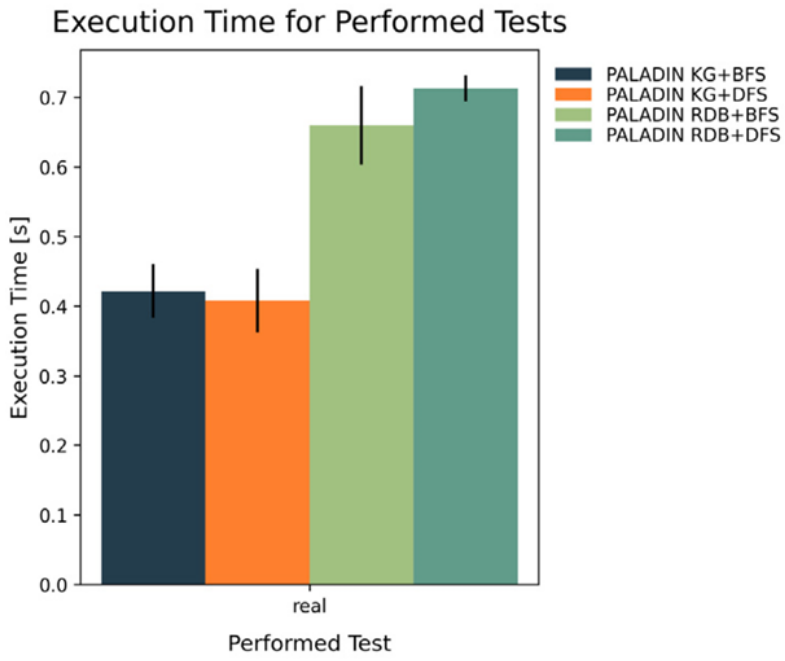

Figura 19: Tiempo de ejecución en tests realizados para grafos de conocimiento (KG) y para bases de datos relacionales (RDB). Fuente: [1].

En trabajos previos, también se realizaron comparaciones con otros lenguajes de validación, entre ellos se encontraban PyShex[4], Shex[4], SHACL[15] y Trav-SHACL[3], de los cuales la ejecución con PALADIN[19] era de las más rápidas en comparación con los demás lenguajes de validación por restricciones. En este caso, la opción en la cual se usa MongoDB, teniendo en cuenta la gráfica de Figura 20, se encontraría aproximadamente entre el caso de PALADIN para RDB en el caso de búsquedas con DFS y Trav-SHACL de manera que aun así seguiría siendo en cuanto a rendimiento, más eficiente que los demás validadores no involucrados con PALADIN.

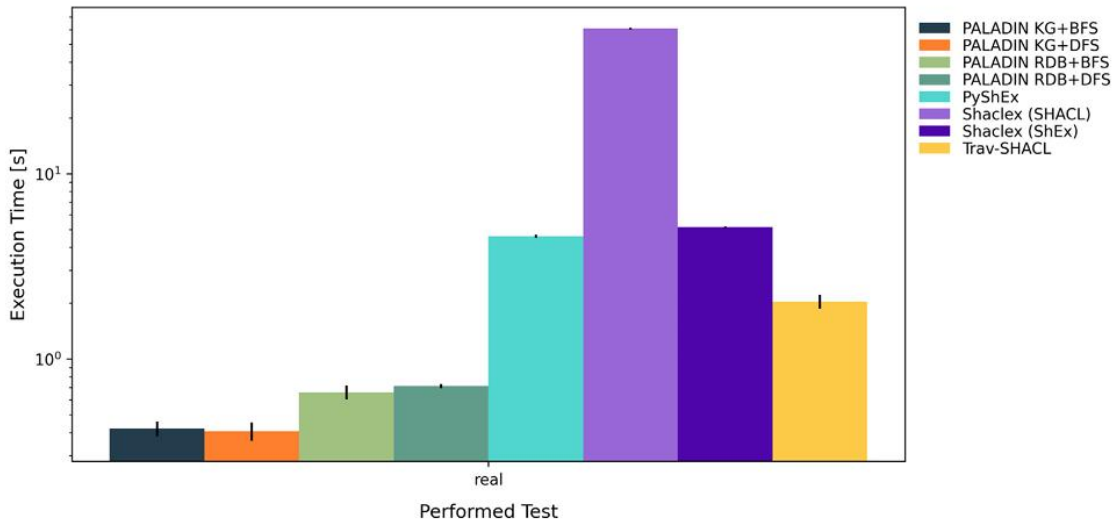

Figura 20: Comparación de validaciones con diferentes bases en PALADIN, con PyShex[4], Shex[4], SHACL[15] y Trav-SHACL[3]. Fuente: [1].

Todos estos resultados con los datos sobre tiempos de ejecución se podían obtener solicitando al ejecutarse la importación de unos ficheros con datos sobre las métricas de tiempo que se han llevado a cabo en total y, por otro lado, las trazas de tiempo que ha dado esa ejecución en concreto. Esto se muestra como resultado tras la ejecución con un esquema para datos sintéticos, cuando se presenta como esquema uno de escalabilidad de los que se ha comentado anteriormente, el fichero devuelto muestra el tiempo que se ha tardado en realizar cada una de las restricciones de manera que se puede llevar a cabo un estudio y análisis más específico que las anteriores. Gracias a esta opción se pudo hacer un análisis y una comparación entre el rendimiento de PALADIN[1] con bases relacionales y con bases no relacionales. En él, se pudo observar que la eficiencia con MongoDB no es del todo la mejor ya que tarda tiempos bastante altos comparados con los obtenidos como resultado en las relacionales. Teniendo en cuenta datos más exacto, varios de las métricas solicitadas con MongoDB no se lograron completar hasta el final debido a que el límite de tiempo de conexión en el link URI con el clúster de MongoDB limitaba esta ejecución de manera que no lograba llegar a la finalización de su ejecución. Pero en los casos en los cuales si se conseguía finalizar dicha ejecución como, por ejemplo, tomando como ejemplo el fichero JSON encargado del rendimiento para árboles con 16 nodos, para validaciones de bases de datos relacionales con BFS y con DFS se obtenían datos de 9 y 9,23 segundos respectivamente, por otro lado, para bases de datos no relacionales tomando el mismo fichero de ejemplo pero con solicitudes para MongoDB con BFS y con DFS se obtenían datos de 29,5 y 37 segundos respectivamente. De esta manera y, como se muestra en el gráfico de la Figura 21, se puede observar que la diferencia entre el rendimiento es levemente más amplia con una diferencia de entre 25 y 20 segundos aproximados entre ambas bases de datos. También se puede observar que entre los dos modos de búsqueda el DFS sigue siendo la búsqueda más lenta en comparación con el modo BFS. También se puede observar otro ejemplo en este gráfico, pero para 32 nodos, en él se muestra que, aunque el tiempo de ejecución aumenta con respecto al ejemplo con 16 nodos, las diferencias entre bases y configuraciones continúan siendo iguales siendo las bases relacionales más eficientes que las no relacionales dando como resultados en las relacionales de 18 segundos tanto para BFS como para DFS y como resultados en las no relacionales de 57 y 59 segundos respectivamente.

Por otra parte, teniendo en cuenta de nuevo la Figura 19, en comparación el uso de PALADIN para grafos de conocimientos en RDF el tiempo de ejecución también aumenta comparado con fuentes de datos relacionales, pero aun así sigue teniendo una ejecución más rápida en comparación con el aplicado a bases no relacionales con una diferencia en torno a 10 segundos aproximadamente. De esta manera, en orden de tiempo de ejecución más rápido a más lento, se encontraría en primer lugar la parte implementada para bases relacionales en MySQL, seguida de la parte realizada para grafos de conocimiento basados en RDF y, por último, la implementación bases de datos no relacionales para MongoDB.

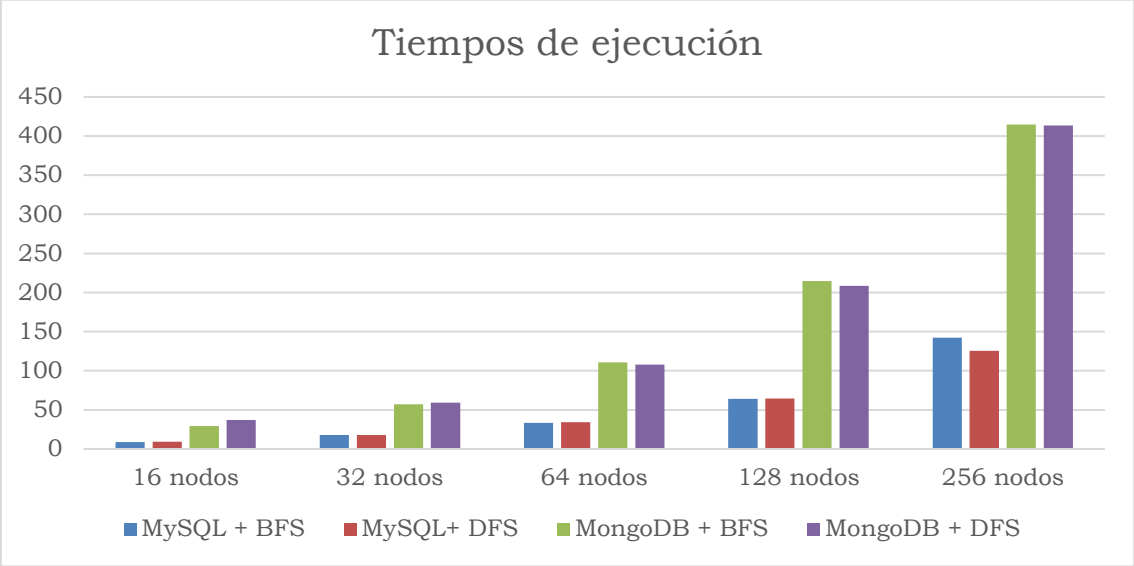

Figura 21: Gráfico de barras con comparación entre el rendimiento de PALADIN para bases relacionales y no relacionales con diferentes configuraciones.

6.2 Resultados SDG

Para la obtención de resultados en SDG[18] y la comprobación de su correcto funcionamiento, se hizo uso de este programa sin la parte de la exportación a grafos de conocimiento ya que el sistema operativo que se estaba utilizando no era compatible con las tecnologías que se requerían para ello. Como no disponía de un Docker solo se hizo con las comprobaciones de SDG para CSV, MySQL y MongoDB; aun así, el funcionamiento de la ampliación a MongoDB se realizó correctamente y con el resultado que se esperaba.

Para ejecutar este generador, se hace uso de un comando similar al siguiente:

```
!python sdg.py -n 1000 -p 0.0
```

En este comando, el número posterior a "-n" hace referencia al número de pacientes registrados que se desea que contenga las bases de datos finales y el número posterior a "-p" hace referencia al porcentaje de mutación en los datos siendo el valor 0.0 un conjunto con datos limpios y el valor 1.0 uno con datos sucios.

Como resultado de la ejecución de este comando se obtienen datos con valores concretos generados aleatoriamente teniendo en cuenta unos rangos y condiciones dependiendo de la variable y los rangos que varían dependiendo de ellas. Estos se almacenan al comienzo en una base de datos MySQL y, además, se guardan en una carpeta *\data*, anteriormente creadas tanto la base vacía como la carpeta, en formato comprimido; posteriormente, se pasa dicha base de datos a datos en CSV además de almacenar los documentos que se generan en la misma carpeta *\data* (como en el ejemplo que se puede ver en la Figura 23)

y, tras esto, se realiza el traspaso de los datos a un clúster en MongoDB como se puede observar en la Figura 22.

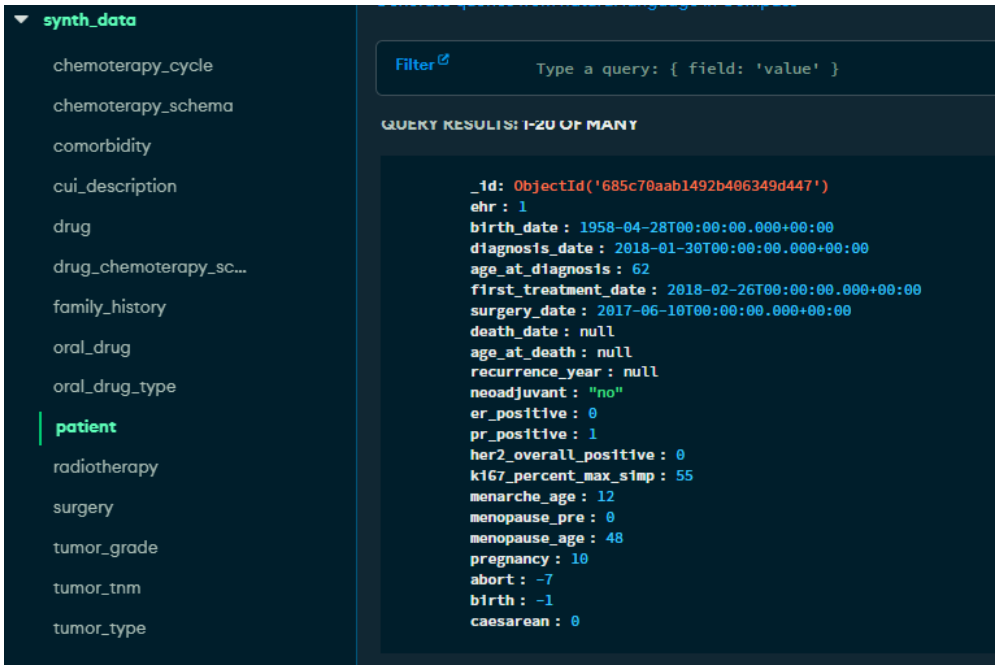

Figura 22: Base de Datos en MongoDB Atlas UI resultado de la ejecución del SDG

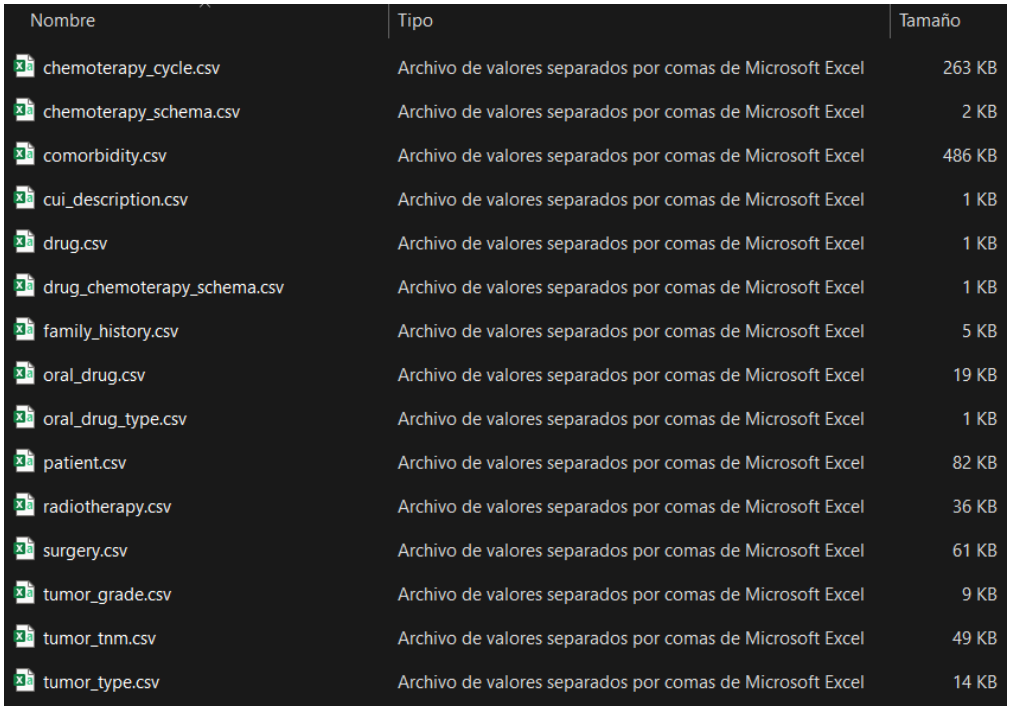

Figura 23: Documentos en formato CSV en una carpeta data/csv resultado de la ejecución del SDG.

A parte de estos conjuntos de datos, como resultado por pantalla en la terminal, se muestra cuanto se ha tardado en realizar diferentes funciones que se realizan en su ejecución, entre ellas se encuentran la generación de datos y la creación y almacenamiento de dichos datos en los diferentes formatos de bases de datos que se encuentran disponibles. También especifica el tiempo total que se ha realizado durante toda la ejecución. Un ejemplo de resultado por terminal es el que se puede observar en la Figura 24. En él se puede observar que la parte que más tiempo de ejecución gasta es la encargada de traspasar la base a MongoDB, seguido de la parte en la que se generan todos los datos antes de almacenarlos en MySQL, tras esto, pasarlo a formato CSV es la siguiente parte con más tiempo de ejecución. Por último, generar la base relacional y almacenar los datos en ella.

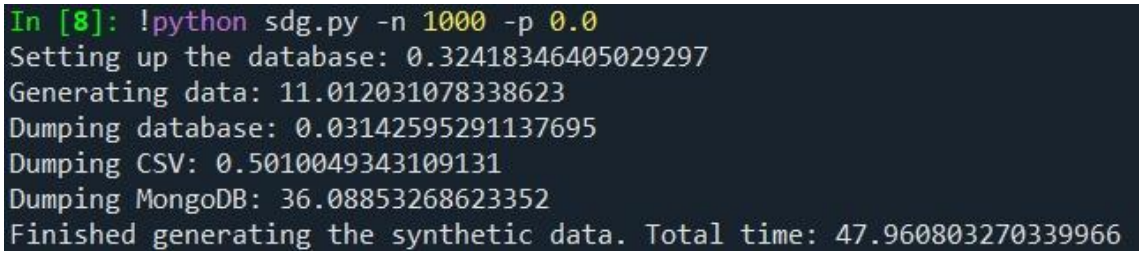

Figura 24: Ejemplo de resultado en terminal de la ejecución para 1000 pacientes de bases con datos limpios.

7 Conclusiones y Trabajos Futuros

En el transcurso de este proyecto, se consiguieron resolver los objetivos que se especificaron al comienzo de él que completaban las tareas planificadas para la organización de este Trabajo de Fin de Grado. Realizando este proyecto se desarrolló la implementación que se concretó como centro de este TFG aparte de otras implementaciones que venían ligadas a él, como lo es el SDG[2],[18]; además, de las adaptaciones de los conjuntos de bases de datos que se encontraban anteriormente en otros formatos y de los ficheros correspondientes a la correcta ejecución de estos prototipos. Aparte, se hizo el estudio anterior a estas implementaciones de las tecnologías que se requerían para llevar a cabo este trabajo ya que se comenzó con un conocimiento muy débil o nulo de todas ellas.

7.1 Conclusiones

Como ya se ha podido observar en los capítulos anteriores, dentro de este Trabajo de Ampliación del Lenguaje de Validación de Datos PALADIN para su Uso en Bases de Datos NoSQL se obtuvieron resultados funcionales en cuanto a la implementación para conjuntos de datos de 1000, 10000 y 100000 registrados aparte de que estos conjuntos podían contener datos limpios, sucios e intermedios siempre y cuando se encontrasen almacenados en MongoDB; además, todos los resultados obtenidos fueron comprobados de las tres maneras de búsquedas que se pueden utilizar para la consulta de validación (BFS, DFS y REC). Se hizo uso de MongoDB Atlas UI y Compass lo cual facilitó la visualización de los resultados obtenidos.

Por otro lado, también se decidió realizar la ampliación del generador de datos sintéticos SDG para su uso en bases de datos NoSQL. Para ello, se comprobó su funcionamiento realizando diferentes pruebas con valores diferentes tanto de número de pacientes como de probabilidad de mutación entre los datos producidos.

Esto resultó en una correcta implementación de ambos prototipos los cual solucionó el problema de limitación de formatos de dichos prototipos además de que se realizó la adaptación del testbench de conjuntos de datos a bases de datos NoSQL en MongoDB más específicamente que ya se ofrecía tanto en bases relacionales en MySQL como en grafos de conocimiento RDF.

Una de las partes más problemáticas del proyecto, aparte del estudio de las tecnologías que su tuvieron que utilizar y del tipo de bases de datos que se utiliza, fue el paso de los ficheros JSON que se necesitan para determinar las validaciones específicas para los casos de prueba con datos sobre el estudio referente al tratamiento con pacientes con cáncer de mama con el gen HER2 amplificado, esto se debía a que al ser de mis primeros proyectos con este tipo de documentos en relación a bases de datos no relacionales basadas en documentos como lo es MongoDB se hizo costosa el traspaso de dichas consultas ya que, en comparación con por ejemplo las consultas en MySQL se

hacen basados en filtros y agregaciones mediante consultas en un lenguaje similar al JSON y no como en MySQL que se realiza con comandos más lineales y menos aparatosos.

En referencia a los objetivos que se especificaron al comienzo de este documento, en el capítulo 1.2, se han completado todos estos exitosamente, aunque de ellos se deben especificar los siguientes comentarios:

1. **Unificación y compatibilidad entre el prototipo PALADIN y las herramientas necesarias para su usabilidad:** Se hizo uso de diferentes implementaciones para la transformación y adaptación de las diferentes testbenchs que se ofrecían como casos de prueba gracias a trabajos previos. De esta manera hizo más eficiente la parte de interoperabilidad y compatibilidad entre PALADIN y las herramientas que se usaron como el clúster de MongoDB y la base de datos en MySQL. Por otra parte, el traspaso de los ficheros JSON a consultas para bases no relacionales no fue tan rápido como el que se esperaba ya que no se encontró una manera tan sencilla para su transformación desde consultas de bases relacionales; debido a esto, se tardó bastante en realizar este paso comparado con las transformaciones de los testbenchs.

2. **Ampliación del prototipo PALADIN:** Tras haber obtenido las herramientas necesarias para el correcto funcionamiento de PALADIN[1],[19], se consiguió llevar a cabo el prototipo PALADIN para bases no relacionales ya que, tomar de base el proyecto que ya se encontraba realizado y tomarlo como ejemplo resulto de mucha ayuda para su desarrollo a la hora de poder comparar los resultados y de realizar el seguimiento del proceso de ejecución. De esta manera, al poder ver cómo funciona para otro tipo de fuentes de datos era más sencillo comprender el funcionamiento que debería seguir para un nuevo formato.

3. **Ampliación del Generador de Datos Sintéticos:** Tras haber conseguido la ampliación del prototipo PALADIN, se consiguió realizar la ampliación del SDG[2],[18] para este mismo formato nuevo. Igual que se realizó en el desarrollo de PALADIN se tomó como base el proyecto que ya se encontraba disponible para formatos de bases relacionales, CSV y grafos de conocimientos basados en RDF en el cual se vio desde el principio como era el funcionamiento para estos datos, en especial para la transformación de los datos generados a CSV, para así poder comprender y así implementar esta nueva implementación para bases no relacionales al SDG.

4. **Análisis de los resultados obtenidos:** Se realizó un análisis en comparación con otros formatos para estos prototipos ofreciendo graficas de comparación con datos reales obtenidos de su ejecución. De esta manera se puedo observar que, aunque era el formato que más tardaba en comparación con otros tipos de fuentes de datos que se aceptan, sigue siendo más rápido y eficiente que otros lenguajes de validación como por ejemplo el SHACL.

7.2 Trabajos Futuros

Como posibles trabajos futuros, se podría realizar una implementación de este prototipo de alguna manera más eficiente ya que, aunque los trabajos realizados son funcionales correctamente, el tiempo de ejecución de este prototipo con la implementación comentada es más alto comparándolo con las versiones en MySQL, por ejemplo; por ello, un posible proyecto futuro sería una implementación con unos resultados menos tardíos y con un rendimiento más eficiente comparado con los otros formatos disponibles.

A parte, como se ha comentado en la transformación de los datos se podría realizar de una manera más eficiente y que ocupe menos espacio la adaptación de las bases de datos de caso de prueba en aquellos valores de tipo bit binario en MySQL a uno más similar que no fuese un booleano para que así se diesen casos de pruebas mas exactos a la realidad. Este cambio realmente no afectaría en el funcionamiento final de estos prototipos ya que los esquemas proporcionados para PALADIN son leídos y usados correctamente para la validación de estas bases y no afecta de ninguna manera con ese tipo de valores mientras se soliciten de la manera correcta.

Por otra parte, siempre se podría realizar la ampliación de PALADIN para otro formato de base de datos para así aumentar la capacidad de conjuntos aceptados por este lenguaje de validación.

Teniendo en cuenta estas dos opciones, finalmente se podría hacer también otra comparación de PALADIN con la nueva implementación junto con otros lenguajes de validación junto con otros futuros formatos de bases de datos para PALADIN.

8 Análisis de Impacto

Este Trabajo de Fin de Grado tiene como objetivo solucionar la limitación en el ámbito software del prototipo PALADIN como lenguaje de validación ya que, a la hora de ser utilizado en una investigación, este estaría limitado al uso de ciertas bases de datos que todavía se encontraban sin implementación. De esta manera y con su amplificación del rango de formatos aceptables en la validación que realiza este prototipo, se proporciona una solución funcional y óptima. Gracias al prototipo PALADIN, en las investigaciones en las cuales se haga uso de bases de datos basadas en procesos será de gran ayuda para la facilitación de la obtención de fuentes de datos coherentes, limpios y listos para ser utilizados en el estudio que se esté desarrollando.

Otro ámbito que podría ser influenciado gratamente gracias a la ampliación del prototipo PALADIN es el ámbito médico. Esto se debe a que, a la hora de llevarse a cabo pautas médicas, muchos de los datos del paciente pueden influenciar en el futuro tratamiento que se le vaya a administrar. Como generalmente, los datos de los pacientes suelen ir asociados a procesos, como las fechas de operaciones anteriores o si en algún momento ha sido tratado con un medicamento en especial, PALADIN solucionaría la limpieza y búsqueda de dichos datos a parte de la obtención de los mejores candidatos para ser estudiados en una investigación de alguna medicina o algún nuevo tipo de tratamiento que puedan ser útiles para futuros pacientes con similares diagnosis. Así, se haría más sencilla y rápida la búsqueda de estos datos entre conjuntos de datos que puedan llegar a parecer sucios.

Un ámbito que destacar, y al que más ligado está, sería el ámbito informático más en concreto el relacionado con las bases de datos e investigaciones cercanas a ellas. En este caso, la ampliación de un lenguaje de validación facilitaría las búsquedas y limpiezas de datos entre la gran cantidad que se llegan a generar en este ámbito. Además, el uso de estos lenguajes y su ampliación a las bases de datos no relacionales haría que, gracias al uso de consultas sencillas, se tengo un desarrollo más seguro y un aumento en la interoperabilidad sobre todo teniendo tantas opciones a la hora de su utilización.

La funcionalidad de PALADIN como lenguaje de validación ya implicaba una ayuda en la comprobación de la correcta validación de bases de datos en las cuales los datos sigan un proceso determinado como lo son por ejemplo unos pacientes en un estudio médico, de esta manera a la hora de realizar la ampliación del prototipo a la aceptación de bases de datos no relacionales en MongoDB se realiza un impacto positivo en las mejoras de entorno médico que cumple parcialmente con el objetivo 3.b perteneciente al objetivo 3 que hace referencia a la Salud y el Bienestar, además de cumplir con el objetivo 9.c perteneciente al objetivo 9 que hace referencia a la Industria, Innovación e Infraestructuras; estos objetivos se encuentran recogidos en los Objetivos de Desarrollo Sostenible (ODS)[22].

En el objetivo 3.b, obtenido del documento [22], se enfatiza que se realice un apoyo a las actividades de investigación y desarrollo de vacunas y medicamentos

para enfermedades de transmisión o no transmisión, de esta manera, al ser PALADIN especializado para bases cuyos datos requieran de un proceso especifico en su estudio; por otra parte, el generador SDG ya está basado en datos reales de un estudio hecho en pacientes con cáncer de mamá lo cual facilita y mejora la obtención de bases de datos de este tipo con datos lo más similares a los reales como casos de prueba para las investigaciones médicas en las cuales se pueda necesitar dicha información.

Además de ser útil en la mejora del ámbito médico, como se cumple en el 3.b, la ampliación de los prototipos presentados provoca un aumento en la investigación científica además de mejorar y ampliar significativamente el acceso y la capacidad tecnológica de la información para ser proporcionado a Internet permitiendo su acceso a un numero mayor de personas que quieran hacer uso de ellos. Estas condiciones implican que también se cumplan las condiciones del objetivo 9.c de los ODS[22].

Bibliografía

[1] A. J. Díaz-Honrubia, P. D. Rohde, E. Niazmand, E. Menasalvas, and M. E. Vidal, "PALADIN: A process-based constraint language for data validation," Information Fusion, vol. 112, Art. no. 102557, 2024

[2] Philipp D. Rohde, Antonio Jesus Diaz-Honrubia, Emetis Niazmand, Maria-Esther Vidal (2023). Dataset: PALADIN: Benchmarks, Experimental Settings, and Evaluation. https://doi.org/10.57702/kf5tc88r

[3] Figuera, M., Rohde, P. D., & Vidal, M. E. (2021, April). Trav-SHACL: Efficiently validating networks of SHACL constraints. In Proceedings of the Web Conference 2021 (pp. 3337-3348).

[4] Thornton, K., Solbrig, H., Stupp, G. S., Labra Gayo, J. E., Mietchen, D., Prud'Hommeaux, E., & Waagmeester, A. (2019). Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation. In The Semantic Web: 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2–6, 2019, Proceedings 16 (pp. 606-620). Springer International Publishing.

[5] Phaltankar, A., Ahsan, J., Harrison, M., & Nedov, L. (2020). MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world. Packt Publishing Ltd.

[6] Sharma, M. (2021). MongoDB Complete Guide: Develop Strong Understanding of Administering MongoDB, CRUD Operations, MongoDB Commands, MongoDB Compass, MongoDB Server, MongoDB Replication and MongoDB Sharding (English Edition. BPB Publications.

[7] Krogh, J. W., Krogh, G., & Gennick. (2018). MySQL Connector/Python Revealed. New York: Apress.

[8] Meroño-Peñuela, A., Lisena, P., & Martínez-Ortiz, C. (2021). Accessing Knowledge Graphs Programmatically. In Web Data APIs for Knowledge Graphs: Easing Access to Semantic Data for Application Developers (pp. 11-25). Cham: Springer International Publishing.

[9] MongoDB Inc., MongoDB PyMongo Documentation. [Online]. Available: https://www.mongodb.com/docs/languages/python/pymongo-driver/current/

[10] Dhalla, H. K. (2020, November). A performance analysis of native json parsers in java, python, ms. net core, javascript, and php. In 2020 16th International Conference on Network and Service Management (CNSM) (pp. 1-5). IEEE.

[11] Python Software Foundation, os.path — Common pathname manipulations. [Online]. Available: https://docs.python.org/3/library/os.path.html#module-os.path

[12] Python Software Foundation, time — Acceso a tiempo y conversiones. [Online]. Available: https://docs.python.org/es/3.10/library/time.html

[13] Fuhrer, C., Solem, J. E., & Verdier, O. (2021). Scientific Computing with Python: High-performance scientific computing with NumPy, SciPy, and pandas. Packt Publishing Ltd.

[14] Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., & Vidal, M. E. (2020, October). SDM-RDFizer: An RML interpreter for the efficient creation of RDF knowledge graphs. In Proceedings of the 29th ACM international conference on Information & Knowledge Management (pp. 3039-3046).

[15] Corman, J., Reutter, J.L., Savković, O. (2018). Semantics and Validation of Recursive SHACL. In: Vrandečić, D., et al. The Semantic Web – ISWC 2018. ISWC 2018. Lecture Notes in Computer Science(), vol 11136. Springer, Cham. https://doi.org/10.1007/978-3-030-00671-6_19

[16] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," ACM Transactions on Database Systems (TODS), vol. 34, no. 3, Art. no. 16, pp. 1–45, Aug. 2009. [Online]. Available: https://doi.org/10.1145/1567274.1567278

[17] Dimitrov, S., Minchev, M., & Zhuang, Y. (2024). BFS versus DFS for random targets in ordered trees. arXiv preprint arXiv:2404.05664.

[18] A. J. Diaz-Honrubia and P. D. Rohde, Synthetic Data Generator. [Online]. Available: https://github.com/SDM-TIB/Synthetic-Data-Generator

[19] SDM-TIB, PALADIN: A process-based constraint language for data validation. [Online]. Available: https://github.com/SDM-TIB/PALADIN/blob/main/README.md

[20] MySQL, A. B. (2006). MySQL administrator's guide and language reference. Sams Publishing.

[21] Pérez, J., Arenas, M., & Gutierrez, C. (2010). nSPARQL: A navigational language for RDF. Journal of Web Semantics, 8(4), 255-270.

[22] de Desarrollo Sostenible, O. (2015). Objetivos de desarrollo sostenible. Naciones Unidas. Recuperado de https://www. un. org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible.

Anexos

1% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto citado

Fuentes principales

0%	Fuentes de Internet
0%	Publicaciones
1%	Trabajos entregados (trabajos del estudiante)

turnitin

Recibo digital

Este recibo confirma quesu trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega:	CELIA MARTIN MUÑOZ
Título del ejercicio:	Turnitin Memoria Final
Título de la entrega:	TFG_Celia_Martin_Muñoz.pdf
Nombre del archivo:	26348_CELIA_MARTIN_MUÑOZ_TFG_Celia_Martin_Muñoz_8376...
Tamaño del archivo:	1.53M
Total páginas:	50
Total de palabras:	16,232
Total de caracteres:	83,951
Fecha de entrega:	02-jul-2025 11:30a. m. (UTC+0200)
Identificador de la entrega:	2709226712

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Ampliación del Lenguaje de Validación de Datos PALADIN para su Uso en Bases de Datos NoSQL

Autor: Celia Martin Muñoz
Tutor: Antonio Jesús Díaz Honrubia

Madrid, julio 2025

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 17:03:04 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)