

**UNIVERSIDAD POLITÉCNICA DE MADRID**

**ESCUELA TÉCNICA SUPERIOR  
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA DE  
TECNOLOGÍAS Y SERVICIOS DE  
TELECOMUNICACIÓN  
TRABAJO FIN DE GRADO**

**GENERACIÓN AUTOMÁTICA DE UN  
MODELO PARA CREAR UNA BASE DE  
DATOS**

**JAVIER GARCÍA CÉSPEDES  
2022**



# GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

## TRABAJO FIN DE GRADO

**Título:** Generación automática de un modelo para crear una base de datos  
**Autor:** Javier García Céspedes  
**Tutor:** D. ....  
**Ponente:** D. ....  
**Departamento:** .....

## MIEMBROS DEL TRIBUNAL

**Presidente:** D. ....  
  
**Vocal:** D. ....  
  
**Secretario:** D. ....  
  
**Suplente:** D. ....

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de:  
.....

Madrid, a                      de                      de 20...

**UNIVERSIDAD POLITÉCNICA DE MADRID**

**ESCUELA TÉCNICA SUPERIOR  
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA DE TECNOLOGÍAS Y  
SERVICIOS DE TELECOMUNICACIÓN  
TRABAJO FIN DE GRADO**

**GENERACIÓN AUTOMÁTICA DE UN  
MODELO PARA CREAR UNA BASE DE  
DATOS**

**JAVIER GARCÍA CÉSPEDES  
2022**



## RESUMEN

Se parte de un esquema XSD del que se extraerán sus campos para formar tablas que se almacenarán en la base de datos Apache Cassandra, y de una serie de ficheros XML correspondiente a la telemetría del satélite UPMSat-2 con mediciones externas tales como: la temperatura o la posición de este, o internas: como su nivel de batería. El objetivo es automatizar la generación de estas tablas en Apache Cassandra, y automatizar la extracción de los ficheros XML que contienen la telemetría. Para ello, se ha diseñado una herramienta escrita en Python que automatice este proceso. Para ello, se cuenta con ciertas bibliotecas que facilitarán el proceso de creación de la herramienta y que convierten los ficheros XML y esquemas XSD en árboles que se recorrerán para filtrar aquellos parámetros que sean de utilidad. Es por ello, que tras recibir el archivo XML con las mediciones recopiladas, se procesarán para extraer sus datos. Por otro lado, del esquema XSD se extraerán los campos que compondrán la tabla que se debe generar. Finalmente, a partir de los datos de telemetría extraídos, así como las columnas de la tabla a la que pertenecerán, se deberán insertar en un nodo de Apache Cassandra. Esta operación se deberá llevar a cabo independientemente del número de ficheros de telemetría que se quieran procesar y tras terminar de procesar dichos ficheros, estos deberán ser eliminados. Esta herramienta, aunque inicialmente está desarrollada para UPMSat-2, es también aplicable a cualquier tipo de sistemas que requiera el uso de una generación de tablas automáticas en Cassandra a partir de un esquema XSD. Solamente se debe cambiar el array con los datos que se quieran extraer.

## SUMMARY

It is based on an XSD schema from which its fields will be extracted to form tables that will be stored in the Apache Cassandra database, and a series of XML files corresponding to the telemetry of the UPMSat-2 satellite with external measurements such as: temperature or position of this, or internal: such as its battery level. The goal is to automate the generation of these tables in Apache Cassandra and automate the extraction of XML files containing telemetry. For this, a tool written in Python has been designed to automate this process. For this, there are certain libraries that will facilitate the process of creating the tool and that convert the XML files and XSD schemas into trees that will be traversed to filter those parameters that are useful. That is why, after receiving the XML file with the collected measurements, they will be processed to extract their data. On the other hand, the fields that will make up the table to be generated will be extracted from the XSD scheme. Finally, from the extracted telemetry data, as well as the columns of the table to which they will belong, they must be inserted into an Apache Cassandra node. This operation must be carried out regardless of the number of telemetry files to be processed and after finishing processing these files, they must be deleted. This tool, although initially developed for UPMSat-2, is also applicable to any type of system that requires the use of automatic table generation in Cassandra from an XSD schema. You only must change the array with the data you want to extract.

## PALABRAS CLAVE

Nodo: espacio de memoria donde se almacenan los datos.

Keyspace: espacio que define la recopilación de datos en los nodos.

Cluster: colección de nodos.

Factor de Replicación: número de nodos de dentro del cluster que almacenan una copia del mismo dato.

Shell: intérprete de comandos del sistema.

CQL: Cassandra Query Language.

Cqlsh: Shell en la que se puede ejecutar CQL.

Primary Key: columna que se usa para identificar un valor de manera única y para poder extraer ese valor. Se divide en Partition Key y Clustering Key.

Clave de Partición: se encarga de la distribución de los datos a lo largo de los nodos.

Clave de Cluster: se encarga de ordenar los datos dentro de los nodos.

ID: valor de identificación

SQL: Un lenguaje de computación para trabajar con conjuntos de datos y las relaciones entre ellos. (Structured Query Language)

NoSQL: Not Only SQL

P2P: Peer to Peer

Rack: Base que almacena sistemas informáticos

# ÍNDICE DEL CONTENIDO

<b>1. INTRODUCCIÓN Y OBJETIVOS .....</b>	<b>1</b>
1.1.    Introducción.....	1
1.2.    Objetivos.....	2
1.3.    Plan de trabajo .....	2
<b>2. ESTADO DE LA TÉCNICA .....</b>	<b>3</b>
2.1.    Breve bibliográfica de las bases técnicas del trabajo.....	3
2.2.    Descripción de las herramientas que se van a usar.....	4
2.2.1.    Bases de Datos.....	4
2.2.2.    Lenguajes de marcado .....	12
2.3.    Justificar por qué es su interés. ....	15
<b>3. REQUISITOS DEL PROYECTO.....</b>	<b>16</b>
3.1.    Requisitos funcionales .....	16
3.2.    Requisitos No Funcionales .....	16
<b>4. DISEÑO DEL SISTEMA/APLICACIÓN .....</b>	<b>18</b>
4.1.    Estructuración en módulos o paquetes .....	18
4.2.    Desarrollo de algoritmos no triviales.....	21
4.2.1.    Algoritmo diseñado para obtener las columnas.....	21
4.2.2.    Algoritmo para obtener los datos de un fichero XML.....	22
4.2.3.    Generación de tablas.....	22
<b>5. IMPLEMENTACIÓN.....</b>	<b>23</b>
5.1.    Instalación del entorno de desarrollo.....	23
5.1.    Código.....	23
5.2.    Uso.....	25
<b>6. PRUEBAS .....</b>	<b>25</b>
6.1.    Pruebas Unitarias .....	25
6.1.1.    Generación de tablas de manera automática.....	25
6.1.2.    Inserción de datos de manera automática. ....	26
6.1.3.    Selección de ficheros XML: .....	26
6.2.    Prueba de Sistema.....	27
<b>7. CONCLUSIONES Y LÍNEAS FUTURAS.....</b>	<b>28</b>
7.1.    Conclusiones.....	28
7.2.    Líneas futuras.....	29
<b>8. BIBLIOGRAFÍA.....</b>	<b>29</b>

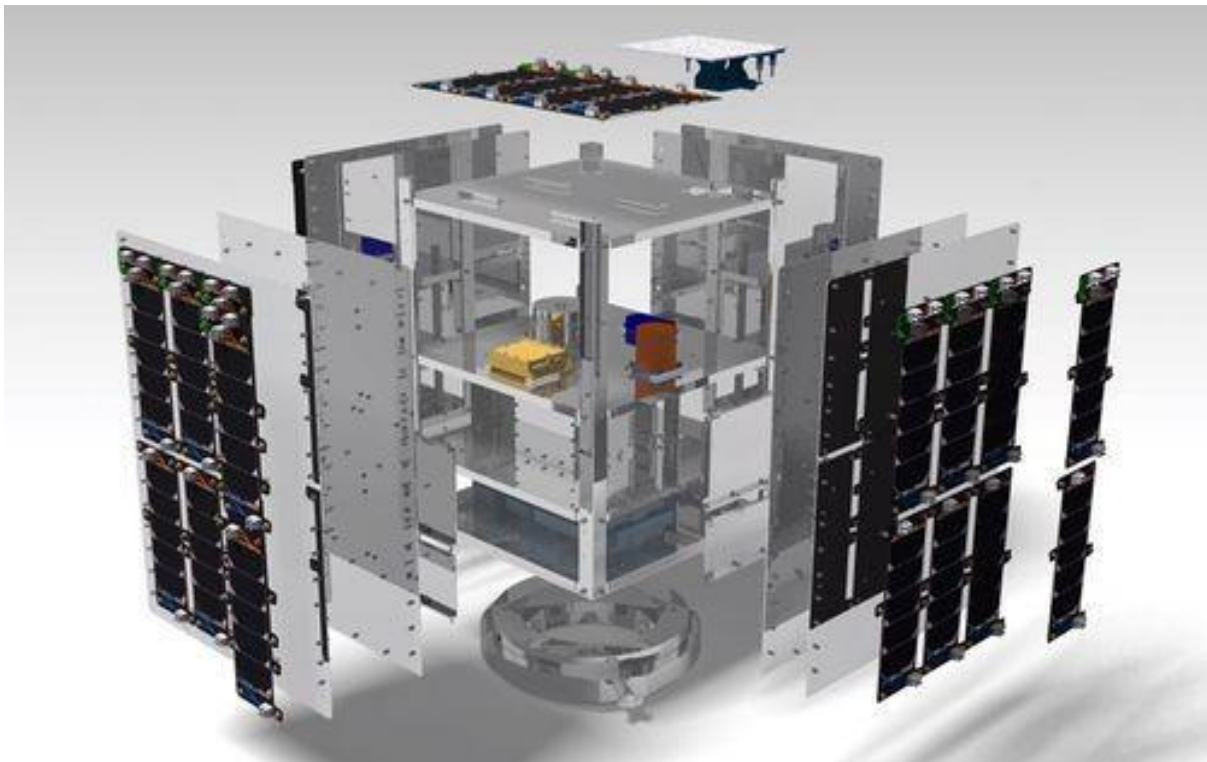


<b>ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES.....</b>	<b>32</b>
A.1 INTRODUCCIÓN .....	32
A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO .....	32
8.1.1.    aspectos éticos .....	32
8.1.2.    aspectos económicos.....	32
8.1.3.    aspectos sociales .....	32
8.1.4.    aspectos ambientales.....	32
A.3 CONCLUSIONES .....	33
<b>ANEXO B: PRESUPUESTO ECONÓMICO .....</b>	<b>34</b>
8.2.    Costes DE Personal.....	34
8.3.    Costes Materiales.....	35
8.4.    Costes Totales .....	35
<b>ANEXO C: PRUEBAS.....</b>	<b>36</b>
8.5.    Anexo 1. Generación de tablas de manera automática. ....	36
8.6.    Anexo 2. Inserción de datos de manera automática: .....	39
8.7.    Anexo 3. Selección de ficheros XML: .....	41
8.8.    Anexo 4. Prueba de Sistema .....	42

# 1. INTRODUCCIÓN Y OBJETIVOS

## 1.1.INTRODUCCIÓN

Este trabajo se ha desarrollado en el ámbito del proyecto OAPES-CM (Operación Avanzada de Pequeños Satélites), en relación a la estación de la tierra. UPMSat-2 es un proyecto universitario que continúa el trabajo de UPMSat-1, su misión consiste en la puesta en órbita de un mini-satélite llamado UPMSat-2, un microsatélite con 50 kg de masa, cuya envolvente geométrica es un paralelepípedo de  $0,5 \times 0,5 \times 0,6$  m [15]. Su lanzamiento se realizó en septiembre de 2020, sobre una órbita polar de unos 600 km de altitud con un periodo de aproximadamente 07 minutos, con una vida útil estimada de 2 años. Cada 24 horas hay dos períodos de visibilidad del satélite desde la estación de tierra, cada uno de ellos de un máximo de 10 minutos de duración. Durante los períodos de visibilidad, las comunicaciones con el satélite se llevan a cabo mediante un enlace de radio dual en la banda VHF de 400 MHz, con una tasa de transferencia de 9600 bit/s. Durante el resto de la órbita se emiten periódicamente mensajes de telemetría básica en una frecuencia de aficionados en la misma banda de VHF.



[1] Imagen del satélite UPMSat-2

En un periodo de visibilidad se intercambian dos tipos de mensajes:

- Telecomandos (TC), enviados desde tierra al satélite. Estos mensajes se usan para controlar el comportamiento del satélite, su modo de funcionamiento, la configuración de los sensores, la configuración del algoritmo de control de actitud, y el inicio y fin de los experimentos.
- Telemetría (TM), mensajes enviados desde satélite a la estación de tierra. Los más relevantes son:

- Hello: Información sobre el estado actual del sistema.
- Event: Información de sucesos o errores relevantes.
- Housekeeping: Datos completos de todos los sensores del sistema.
- Experiment: Datos de los sensores activos en el experimento.

Este Trabajo de Fin de Grado, se va a centrar en automatizar la recepción de la telemetría, concretamente en los mensajes de categoría “Housekeeping”, los cuales se procesarán en la estación Tierra y se almacenarán en una base de datos.

## 1.2.OBJETIVOS

El objetivo básico de este Trabajo de Fin Grado es automatizar la generación de tablas en una base de datos. Para ello, se parte de una herramienta desarrollada en Python que deberá primero reconocer cuales son los campos que necesitamos extraer de un esquema XSD para que sirvan como modelo de la tabla a representar. Y posteriormente, ser capaz de generar la tabla y poder insertar datos de telemetría en ella. Por otro lado, debemos ser capaces de extraer la información de telemetría de un fichero XML para poder almacenar su contenido en la base de datos. Una vez procesados, estos deben ser eliminados.

Por ello, los objetivos son los siguientes:

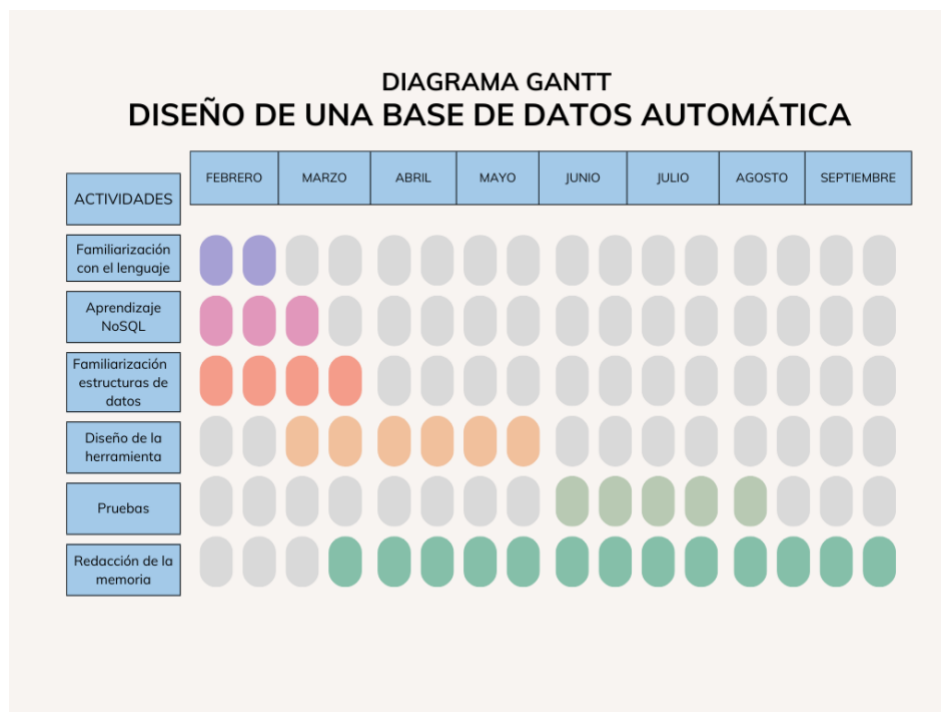
1. Generar un modelo de tablas que responda a la telemetría
2. Implementar un algoritmo capaz de extraer la información necesaria de un esquema XSD
3. Desarrollar una herramienta capaz de almacenar la telemetría en una base de datos
4. Realizar una serie de pruebas para su correcto funcionamiento

## 1.3. PLAN DE TRABAJO

A la hora de la planificación del trabajo se siguieron las siguientes directrices:

1. Familiarización con el lenguaje: Lo primero fue aprender Python mediante cursos online pues la herramienta se iba a desarrollar en este lenguaje.
2. Aprendizaje sobre bases de datos NoSQL: Posteriormente, se investigó sobre bases de datos NoSQL y la utilidad en este proyecto, haciendo especial énfasis en Cassandra.
3. Familiarización con las estructuras de datos: Tras conocer Python y Cassandra, se investigó en el lenguaje de marcado XML y los esquemas de validación XSD Schema, y como se están utilizando en el desarrollo de este proyecto.
4. Diseño de la herramienta: Se trabajó en hacer funcionar la herramienta mediante la prueba de distintas bibliotecas que nos facilitaban la interacción entre todos los sistemas.
5. Pruebas: Una vez desarrollada la herramienta, se probó su funcionamiento
6. Redacción de la memoria: Donde se describen todos los procesos seguidos en el desarrollo de la herramienta, los resultados que se han obtenido, y una serie de conclusiones y futuras mejoras.

Se resume este proceso en un diagrama de Gantt:



## 2. ESTADO DE LA TÉCNICA

### 2.1. BREVE BIBLIOGRÁFICA DE LAS BASES TÉCNICAS DEL TRABAJO

Para el desarrollo de la herramienta, se utilizará la base de datos Apache Cassandra, que destaca en el manejo de series temporales debido a la gran capacidad de velocidad de lectura/escritura de los datos que tiene. Como lenguaje de programación, usaremos Python, el cual está muy extendido hoy en día y destaca por su sencillez y facilidad de aprender. Finalmente, los ficheros donde se almacena la telemetría están escritos en XML y las tablas que se querrán crear vienen dentro de su esquema XSD, uno de los lenguajes de marcado más extendidos y que permite extraer los datos deseados de forma sencilla.

A la hora de desarrollar código contamos con la ventaja de que existen bibliotecas, cuyas funciones facilitan el trabajo pues recogen funcionalidades enteras que podríamos usar en una sentencia como puede ser una función, o que se encargan de recorrer el documento de ciertas maneras que pueden ser útiles para nosotros. Gracias a las bibliotecas podemos utilizar todas estas funcionalidades para implementarlas en proyectos de mayor tamaño.

La empresa encargada de dar soporte a Cassandra, Datastax, proporciona una biblioteca para gestionar su uso a través de un script de Python: cqlengine. [13] Esta biblioteca permite plantear un keyspace como si fuese un modelo de datos de Python. Para su uso, primero se debe instalar con el comando en la shell: “pip install cassandra-driver”. Una vez instalado se deben importar en el código lo que el controlador interpreta como columnas y como modelo

de datos, mediante las líneas: `from cqlengine import columns` y `from cqlengine import Model`. Ahora es posible generar una tabla a través de un modelo Python definiendo una clase pasando como parámetro `Model`.

Para poder trabajar con esquemas XSD en Python se usará la librería `XMLSchema`. [16] Esta permite, entre otras funcionalidades: Construir a partir de archivos XML sus plantillas XSD, validar archivos XML, o la funcionalidad que se usará en este proyecto, el transformar una plantilla XSD en un árbol que podremos recorrer y del se pueden extraer sus elementos. Para instalar esta librería se usará la librería de Python Pip mediante y mediante el comando: `"pip install xmlschema"` y para su uso se tendrá que importar la librería en nuestra herramienta mediante la instrucción: `"import xmlschema"`.

Para recorrer un esquema XSD y extraer su información también se planteó el uso de `GenerateDS` una librería que dado un esquema XSD te genera una clase de Python con que contiene también funciones "parser" que cargan un documento XML en la clase. [18] Cada clase generada contiene:

- Un constructor (`__init__`)
- Métodos "getter" y "setter" para cada dato
- Un método "export" que escribe la clase en formato XML
- Un método "exportLiteral" que escribirá la instancia (y cualquier subinstancia anidada) en un objeto de archivo como texto

Sin embargo, aunque en primera instancia puede parecer que cumple con los requisitos para el desarrollo de la herramienta, se encontró como desventaja que a la hora de manipular los datos era más complejo acceder a ellos de manera automática, por lo que se descartó esta opción.

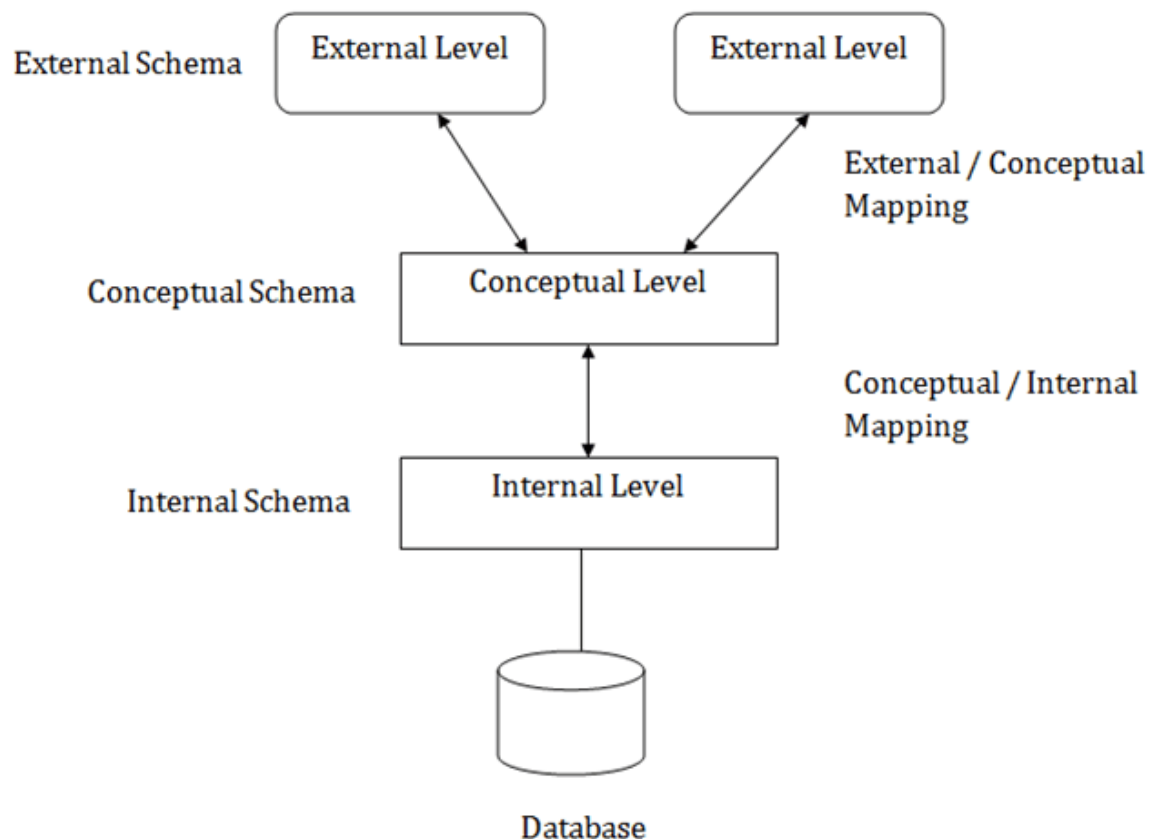
Es por ello se escogió la opción de a partir de las bibliotecas mencionadas anteriormente, desarrollar un algoritmo que cumpla con la funcionalidad deseada, pues las librerías escogidas nos ayudan con sus iterables y métodos a extraer la información independientemente de cómo sea esta y no tener que depender de entre otras cosas el tipo de variable o el nombre con el que se define. Ahora podremos almacenar todos los datos extraídos en listas y recorrerlos para formar un ejecutable auxiliar para proceder a la creación de las tablas y la inserción de datos.

## 2.2. DESCRIPCIÓN DE LAS HERRAMIENTAS QUE SE VAN A USAR

### 2.2.1. BASES DE DATOS

Una base de datos se encarga de recopilar datos para después organizarlos y relacionarlos para que puedan ser más accesibles a la hora de recuperarlos. Son sistemas que implementan un aislamiento entre capas físicas, lógicas, y de aplicaciones externas y se pueden representar mediante tres capas: la capa física, donde se almacenan los bytes de datos, la capa lógica,

donde se tienen los modelos de datos y sus relaciones, y la capa externa, donde se visualizan los datos. [19]



[20] Capas de una base de datos

Dentro de los tipos de bases de datos existentes se pueden distinguir dos tipos: las bases de datos relacionales o SQL, y las bases de datos no relaciones o NoSQL.

## SQL

Las bases de datos SQL destacan por el almacenamiento de los datos manteniendo una relación entre estos. Las bases de datos SQL más comunes son aquellas en las que la información se almacena en tablas, donde los parámetros de los datos que se manejan se organizan en columnas, y los datos que se insertan se organizan por filas. Cada fila contiene un ID único para evitar que los datos puedan estar duplicados, y a su vez garantizar la integridad referencial: Si se elimina uno de los registros, la integridad de los registros restantes no será afectada.

Otras de sus ventajas son el ancho soporte del que se dispone al ser estas las más utilizadas actualmente, la Atomicidad de la información: Al realizar cualquier operación en la base de datos, si surge algún problema, la operación no se realiza y el uso de SQL como estandarización, los datos deben ser consistentes, las transacciones concurrentes deben ser independientes, y ante un fallo del sistema, una transacción de información debe ser recuperable. Esto se corresponde con el modelo ACID que rigen las bases de datos relacionales para que esta funcione de manera fiable. [23]

Para el uso de estas bases de datos, se utiliza el estándar SQL: Un lenguaje de computación para trabajar con conjuntos de datos y las relaciones entre ellos. Aparte de manipular datos, se puede crear y modificar el diseño de objetos de base de datos. La sintaxis SQL se basa en la sintaxis del idioma inglés. Las cláusulas para la manipulación de los datos son las siguientes: [21]

- INSERT INTO: Agrega datos a una tabla. Posteriormente se define la tabla en la que se van a insertar los datos
- VALUES: Selecciona los valores a insertar dentro de una tabla
- SELECT: Muestra los campos de una tabla. Es un campo obligatorio en una instrucción SQL
- FROM: Selecciona la tabla sobre la que queremos conocer los campos. Es un campo obligatorio en una instrucción SQL
- WHERE: Especifica los criterios de búsqueda de la tabla. No es un campo obligatorio dentro de una instrucción SQL
- ORDER BY: Especifica el orden de los resultados. No es un campo obligatorio dentro de una instrucción SQL
- GROUP BY: Contiene funciones de agregado. Muestra los campos que no se resumen en la cláusula SELECT
- HAVING: especifica las instrucciones que se aplican a los campos que se reúnen en una función SELECT.

Por otro lado, para la definición de los datos contamos con las siguientes cláusulas:

- CREATE: para crear nuevos objetos de datos o tablas
- ALTER: modifica la estructura de una tabla dentro de una base de datos
- DROP: elimina una tabla de una base de datos
- TRUNCATE: elimina los datos de una tabla dentro de una base de datos

## NOSQL

Asimismo, se presentan las bases de datos no relacionales que serán las que vamos a manejar. Estas se caracterizan por el uso de una estructura de datos algo más compleja que una tabla y no existe un término identificador que relacione los conjuntos de datos entre sí. Estas bases de datos no siguen el estándar SQL a la hora de trabajar con ellas, si no que se apoyan en él. Como grandes ventajas, aparecen las siguientes:

- Son fácilmente escalables ya que tienen la característica de ser escalables horizontalmente. Esto implica que pueden crecer en número de máquinas sin tener que aumentar en especificaciones.
- Al tener un carácter descentralizado mejora la latencia.
- Se pueden implementar en máquinas que dispongan de pocos recursos.
- Las peticiones para grandes cantidades de datos están optimizadas.
- Se pueden hacer cambios en los esquemas sin tener que parar las bases de datos.
- Los datos se replican, haciendo posible tener la información más protegida y aumentar la disponibilidad de los mismos.



Por otro lado, las desventajas a la hora de implementar una base de datos NoSQL frente una SQL son: [2]

- No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos. Soportan lo que se llama consistencia eventual.
- Problemas de compatibilidad entre instrucciones SQL. Las nuevas bases de datos utilizan sus propias características en el lenguaje de consulta y no son 100% compatibles con el SQL de las bases de datos relacionales. El soporte a problemas con las queries de trabajo en una base de datos NoSQL es más complicado.
- Falta de estandarización. Hay muchas bases de datos NoSQL y aún no hay un estándar como sí lo hay en las bases de datos relacionales. Se presume un futuro incierto en estas bases de datos.
- Poca usabilidad. Suelen tener herramientas de administración no muy usables o se accede por consola.

Al igual que las bases de datos SQL se rigen por el modelo “ACID”, las NOSQL se rigen por el modelo BASE: [24]

- Basic Availability: El enfoque de la base de datos NoSQL se centra en la disponibilidad de los datos incluso en presencia de múltiples errores.
- Soft State: la consistencia de los datos es un problema del desarrollador y no debe ser manejado por la base de datos.
- Eventual Consistency: En algún momento, los datos convergen a un estado consistente.

Las bases de datos NoSQL se pueden caracterizar dentro del Teorema CAP, presentado por Eric Brewer en el año 2000. Este teorema sugiere que todo sistema de almacenamiento de datos es vulnerable a fallos de conectividad de la red, por lo que, frente al nivel de tolerancia de la partición de los nodos, tendrá que realizar algún tipo de concesión entre el acceso a la información o a su versión más reciente. Recibe el nombre de CAP porque se caracteriza en los siguientes tres puntos: [22]

- C - Consistencia: Los datos se encuentran sincronizados y replicados en todos los nodos a la vez.
- A - Disponibilidad: Se debe respuesta correcta y rápida para todas las solicitudes, aunque existan nodos inactivos.
- P - Tolerancia a particiones: Se refiere a la capacidad del sistema para permanecer estable y continuar procesando solicitudes a pesar de ocurrir una interrupción de los nodos.

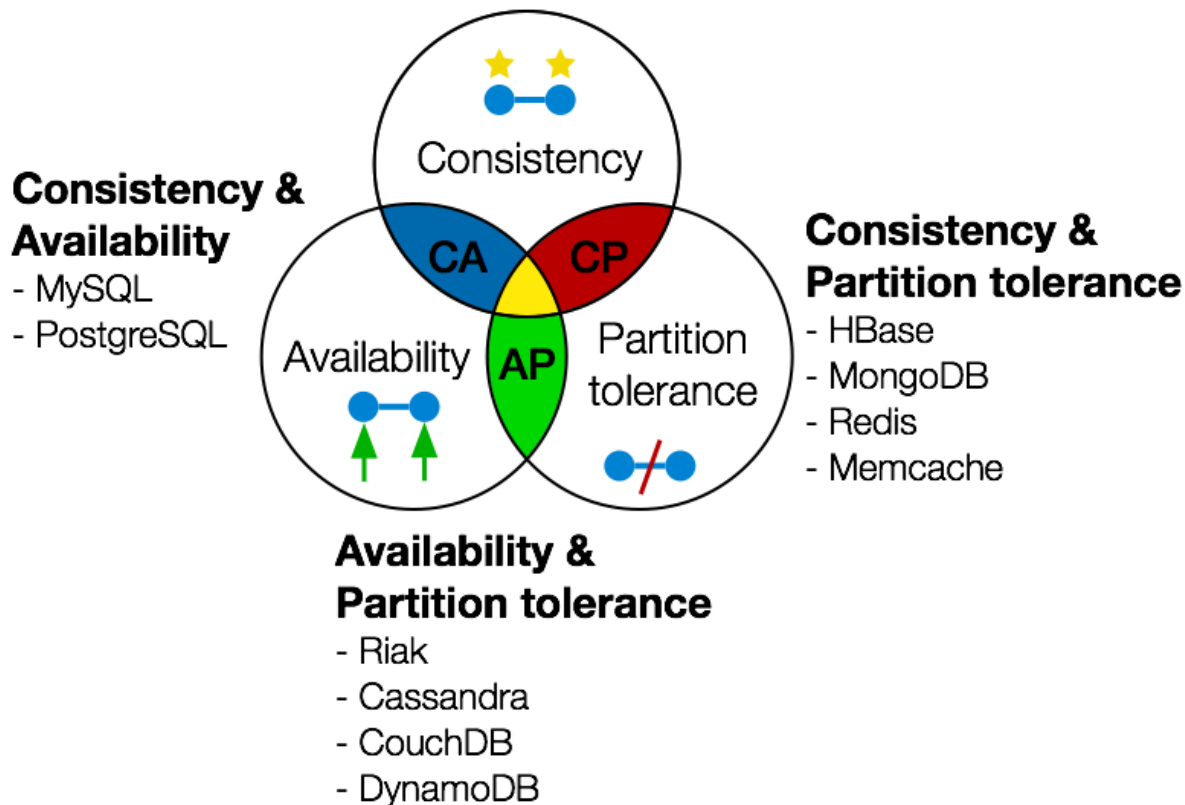
El teorema asegura que de entre las tres características presentadas, una base de datos puede garantizar dos de ellas. Las combinaciones son las siguientes [5]:

- CA: Consistencia y Disponibilidad - Se garantiza el acceso a la información y el valor del dato es consistente (igual) para todas las peticiones atendidas; de haber cambios, se mostrarán inmediatamente. Sin embargo, la partición de los nodos no es tolerada por el sistema de forma simultánea.
- AP: Disponibilidad y Tolerancia a la partición - Se garantiza el acceso a los datos y el sistema es capaz de tolerar (gestionar) la partición de los nodos, pero dejando en



segundo plano la consistencia de los datos, ya que no se conserva y el valor de dato no estará replicado en los diferentes nodos al instante.

- CP: Consistencia y Tolerancia a la partición - Se garantiza la consistencia de los datos entre los diferentes nodos y la partición de los nodos se tolera, pero sacrificando la disponibilidad de los datos, con lo cual, el sistema puede fallar o tardar en ofrecer una respuesta a la petición del usuario.



[3] Diagrama de CAP

Por otro lado, podemos clasificar las bases de datos según almacenen la información, destacando los siguientes modelos: [24]

- Orientadas a documentos: La información se almacena en un documento, típicamente un JSON o un archivo XML. Utilizan una clave única para cada registro. Destacan MongoDB y CouchDB
- Orientadas a grafos: La información se almacena en los nodos y las relaciones y propiedades se indican en los enlaces. Destaca Neo4J.
- Clave-Valor: Cada elemento se identifica con una clave única pudiendo ser recuperado de una manera muy rápida. Destaca Apache Cassandra.

## APACHE CASSANDRA

La realización de este trabajo se desarrollará a través de Apache Cassandra, una base de datos distribuida NoSQL clave-valor de código abierto desarrollada inicialmente por Facebook en 2008 y traspasada a la Fundación Apache haciendo que sea una herramienta de

software libre. Según el teorema CAP, podemos caracterizar a Apache Cassandra como una base de datos AP, ya que sacrifica su consistencia para ofrecernos una alta disponibilidad, y una gran tolerancia a particiones.

Las características al utilizar Apache Cassandra frente a otras bases de datos NoSQL son las siguientes:

- Es una base de datos distribuida
- Se encuadran dentro de las bases de datos no relacionales de tipo columnas
- Son fácilmente escalables, ya que son NoSQL
- Escala linealmente
- No sigue un patrón maestro-esclavo, sino que es P2P. Esto lo que conlleva es que, si se cae un nodo, el servicio puede seguir funcionando
- Permite la escalabilidad horizontal
- Tolerante a fallos, ya que posee replicación de datos
- Permite definir un nivel de consistencia
- Para su uso utiliza un lenguaje CQL, similar a SQL

La clave con la que se almacenan los datos dentro de un clúster se denomina Partition Key: [26] Todos los datos que compartan la misma Partition Key se van a almacenar dentro de un mismo nodo. De esta forma, agilizamos el proceso de búsqueda en una base de datos. El funcionamiento de la Partition Key dentro de Cassandra es el siguiente: Se transforma cada Partition Key mediante una función Hash para generar un ID único. De esta manera, Cassandra decide a qué nodo debe ir cada dato introducido. El resultado de aplicar una función Hash a una Partition Key se le conoce como Token: Y se implementan como un Integer de 64 bits con un rango de -263 hasta 263-1. Dentro de un nodo, los datos ordenados por su Partition Key, se ordenan posteriormente por su Clustering Key. Los clústeres de Cassandra tienen forma de anillo y cada nodo se encarga de almacenar aquellos datos cuyo valor de Token sea igual o inferior al del dato a almacenar. Cada nodo puede almacenar varios nodos virtuales, para que de esta forma sean los nodos con más capacidad aquellos que almacenen más Partition Keys diferentes.

El lenguaje que maneja Apache Cassandra es CQL, muy similar a SQL, pero con la gran diferencia de que al contrario que en SQL, en CQL no existen “joins”. Todos los datos se almacenan en nodos y estos a su vez se agrupan, formando un clúster. A lo largo de este clúster, se pueden tener uno o varios “keyspaces” o espacios de trabajo, los cuales almacenan las tablas. Cada keyspace almacena datos no relacionados entre sí. Para interactuar con Apache Cassandra mediante CQL, se usa “cqlsh” o la Shell de Cassandra. Para ello, antes debemos iniciar Cassandra en otra terminal mediante la instrucción “cassandra -f”. Una vez realizado este paso previo, se podrá iniciar la Shell de Cassandra.

Dentro de la shell de Cassandra, se debe manejar el uso de las keyspaces, para las cuales tenemos las siguientes instrucciones: [27]

- Crear un keyspace: `CREATE KEYSPACE nombre_keyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = 'true';` El campo class toma dos valores: “Simple Strategy”, para un clúster de un nodo, o “Network Topology” para un clúster de más de un nodo. Con el replication\_factor escogemos en cuantos nodos redundan nuestros datos. El campo “durable\_writes”

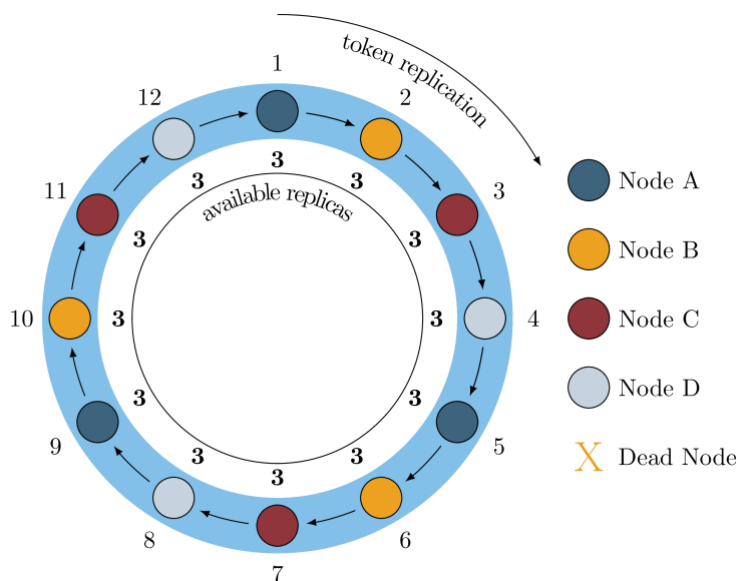
es un booleano que estando a true nos ofrece a nuestro keyspace una mayor seguridad siendo este su valor por defecto. Mientras marcando este campo como false, dotamos a nuestro keyspace de mayor velocidad, pero de un mayor riesgo de pérdidas de datos.

- Desplegar un listado con nuestros keyspace: `DESCRIBE KEYSPACES;`
- Eliminar un keyspace: `DROP KEYSPACE nombre_keyspace;`
- Activar un keyspace de trabajo: `USE nombre_keyspace;`

Tras activar un keyspace, se pueden tener una o varias tablas creadas. Las tablas se forman por filas de datos, y columnas de campos. Cada fila contiene una Primary Key, una columna que especifica el dato al que estamos accediendo. El proceso de crear tablas en CQL es muy similar al de SQL, debemos usar la palabra reservada `CREATE` y especificar el nombre de la tabla, las columnas de la tabla, la Primary Key, y los tipos de datos que va a almacenar cada columna, siendo algunos de estos tipos: Integer, Boolean o Text. Para ver una tabla, se utiliza la palabra reservada `DESCRIBE TABLE` seguida del nombre de la tabla, y para eliminar una tabla `DROP TABLE` y el nombre de la tabla. No se tienen por qué insertar datos en todas las columnas, solo se hace en las que se mencionan antes de la etiqueta `VALUES` y para introducir texto, debe hacerse entre comillas simples. [12]

Finalmente, se pueden insertar datos dentro de una tabla. Para ello, se hace uso de la palabra reservada `INSERT INTO` añadiendo a continuación el nombre de la tabla y sus campos, y la palabra reservada `VALUES` y los valores a introducir para cada columna. Para buscar valores dentro de alguna tabla, se utiliza la palabra reservada `SELECT` seguida de los campos que se quieran imprimir y la palabra reservada `FROM` más el nombre de la tabla sobre la que se busca. Es posible añadir cláusulas a la búsqueda con la palabra reservada `WHERE` seguida de las cláusulas para especificar los datos a imprimir. Cabe destacar, que solo se puede buscar por columnas que hayamos designado como Partition Key o como Clustering Key. Se pueden ordenar los datos extraídos de una búsqueda mediante la palabra reservada `ORDER BY`, más el nombre de la columna, que debe haber sido previamente nombrada como Clustering Key.

Por otro lado, Apache Cassandra cuenta con replicación de datos, de esta manera los datos almacenados en un centro de datos se pueden replicar en otro, para que, en caso de error, exista una copia a la que podamos acceder que se localice en otro centro de datos. Apache Cassandra también tiene replicación a nivel local, pudiendo replicar los datos no solo en otro centro de datos, si no en otro rack.



[28] Ejemplo de un anillo en Cassandra

Dentro del modelo, [12] para generar una columna nueva se debe declarar una variable, cuyo nombre será el nombre de la columna e igualarla a la sentencia `columns` e indicando el tipo del dato. Un ejemplo puede ser: `columna_ejemplo = columns.Text()`. En este caso, se ha generado una nueva columna de nombre “columna\_ejemplo” cuyos datos aceptados deben ser cadenas de texto. Si se quisiera que los datos en vez de ser un string fuesen un entero se debe sustituir `Text()` por `Integer()`. Dentro de los paréntesis, la función acepta ciertos argumentos, como por ejemplo el designar esta columna como Primary Key (`primary_key=True`), si se designa como Partition Key (`partition_key=True`) o como campo que no puede quedar vacío (`required=True`). El nombre de la tabla viene dado por el nombre del modelo, o bien podemos llamar al método `__table_name__`, e igualarlo al nombre que se quiera darle.

Una vez se ha creado el modelo, se debe sincronizar con el keyspace, para ello se importa un nuevo controlador a través de la sentencia “`from cqlengine import connection`” y llamando a esta nueva variable con el método `setup`, incluyendo en la dirección IP en la que se encuentra el keyspace, y el nombre del keyspace al que se quiera conectar. Por ejemplo, si se quisiera conectar a un keyspace llamado “prueba” que se encuentra en almacenado localmente (dirección IP: 127.0.0.1), se debe escribir la sentencia: “`connection.setup(['127.0.0.1'], “prueba”)`”. Tras la conexión a el keyspace, hay que sincronizar la tabla con el mismo. Para ello, se utiliza un nuevo controlador a través de la sentencia: “`from cqlengine.management import sync_table`”. A través de esta sentencia, se habrá importado una función para sincronizar nuestra tabla con el keyspace al que no hayamos conectado con la sentencia anterior. Para hacerlo, solo hay que llamar a la función `sync_table` y pasar como parámetro el nombre del modelo a sincronizar.

Para insertar un dato en la tabla, se puede realizar de dos maneras: [12]

- La primera de ellas, hacer uso de la función “create” para ello, se debe crear un objeto y pasarle la función, que acepta como parámetros, todas las columnas de la tabla. Por

ejemplo, para insertar en una tabla llamada personas que almacena un nombre y una edad se escribe la siguiente sentencia:

```
p1 = personas.create(nombre= "Jaime", "21").
```

- La segunda de ella es muy similar a la primera, en vez de llamar a la función create se llama a la función save() tras crear un objeto. Por ejemplo, para insertar en una tabla llamada personas que almacena un nombre y una edad se escribe la siguiente sentencia:

```
p1 = personas(nombre= "Jaime", "21")
```

```
p1.save()
```

### 2.2.2. LENGUAJES DE MARCADO

Un lenguaje de marcado se define como un sistema de anotaciones organizado, dando al documento una estructura específica. Estos pueden ser clasificados según los siguientes tipos: [25]

- Lenguajes de presentación: Define la apariencia del texto.
- Lenguajes de procedimientos: Se interpretan las etiquetas del documento para la realización de diferentes acciones.
- Lenguajes descriptivos o semánticos: Definen el contenido, sin especificar su representación.

En este caso, nos centraremos en XML, un lenguaje descriptivo que permite definir otros lenguajes de marcas.

#### XML

XML (Extensible Markup Language) es un lenguaje de marcado similar a HTML. La principal diferencia es que XML es un lenguaje de marcado no predefinido, lo que significa que las etiquetas las generamos nosotros mismos. Su propósito general es la transmisión de información.

La estructura de un documento XML es en forma de árbol, donde cada documento tiene un elemento raíz del que cuelgan el resto de los elementos. Estos elementos están escritos en texto plano y se escriben entre etiquetas, indicadas por los caracteres menor que: "<", y mayor que: ">", donde almacenaremos el nombre del elemento. Un ejemplo de sintaxis sería: `<etiqueta>valor</etiqueta>`. Dentro de cada las etiquetas podemos incluir atributos, que vienen representados en formato clave, valor. El valor debe ir siempre entrecomillado.

#### XSD

Un esquema XSD tiene la función de validar un documento XML. Por tanto, a través de un esquema se puede definir la estructura de un XML: los elementos que lo componen, los tipos de datos, atributos o cuantas veces se repiten. En comparación con DTD (definición de tipo de documento), tienen las ventajas mencionadas anteriormente, pero son documentos más difíciles de interpretar.

A la hora de definir un esquema, se comienza definiendo la versión XML que vamos a utilizar y qué tipo de codificación de los caracteres usaremos. Posteriormente, existe la etiqueta `xs:schema`, que marca la raíz y presenta todos los atributos. Finalmente, a través de la etiqueta `xs:se` pueden definir los atributos (`xs:attribute`) del elemento raíz del documento XML. Por otro lado, el documento XML deberá referenciar al esquema a través de la etiqueta `xmlns:xsi`.

El componente `xs:element` declara los elementos que pertenecen al documento XML. Los principales atributos que se pueden utilizar en la declaración son los siguientes: [4]

- `name`: Indica el nombre del elemento. Obligatorio si el elemento padre es `<xs:schema>`.
- `ref`: Indica que la declaración del elemento se encuentra en otro lugar del esquema. No se puede usar si el elemento padre es `<xs:schema>`. No puede aparecer junto con `name`.
- `type`: Indica el tipo de dato que almacenará el elemento. No puede aparecer junto con `ref`.
- `default`: Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con algún tipo de dato textual.
- `fixed`: Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con algún tipo de dato textual.
- `minOccurs`: Indica el mínimo número de ocurrencias que deben aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es `<xs:schema>`. Va desde 0 hasta ilimitado (unbounded). Por defecto 1.
- `maxOccurs`: Indica el máximo número de ocurrencias que pueden aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es `<xs:schema>`. Va desde 0 hasta ilimitado (unbounded). Por defecto 1.

Un ejemplo puede ser el siguiente:

```
<xs:element name="nombre" type="xs:string" default="TicArte" minOccurs="1"
maxOccurs="unbounded" />
```

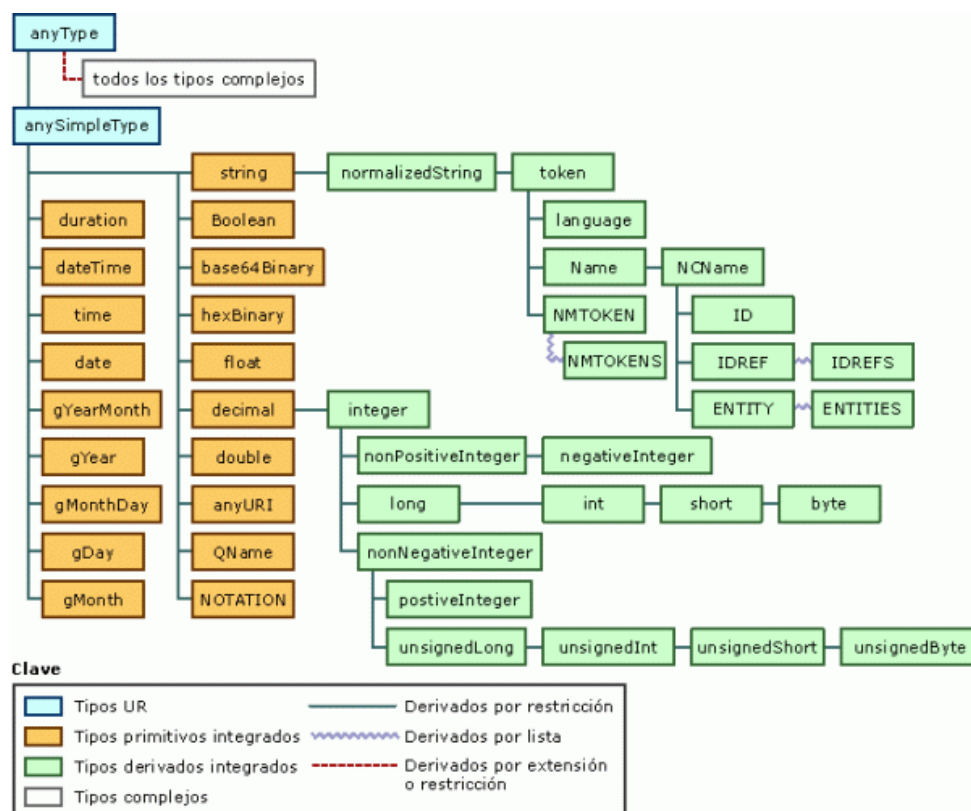
El componente `xs:attribute` permite declarar los atributos de los elementos del documento XML. Entre otros, los principales atributos que se pueden utilizar en la declaración son los siguientes: [4]

- `name`: Indica el nombre del atributo.
- `ref`: Indica que la declaración del atributo se encuentra en otro lugar del esquema. No puede aparecer junto con `name`.
- `type`: Indica el tipo de dato que almacenará el atributo. No puede aparecer junto con `ref`.
- `use`: Indica si la existencia del atributo es opcional (optional), obligatoria (required) o prohibida (prohibited). Por defecto opcional.
- `default`: Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con algún tipo de dato textual.
- `fixed`: Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con algún tipo de dato textual.

Un ejemplo puede ser el siguiente:

```
<xs:attribute name="moneda" type="xs:string" default="euro" use="required" />
```

En un esquema, podemos dividir los tipos de datos en dos grandes grupos: los complejos (xs:complexType), que se pueden clasificar en: elementos dentro de otros elementos, elementos que tienen atributos, elementos mixtos que tienen datos y otros elementos y en elementos vacíos, y los simples (xs:simpleType), que se dividen en predefinidos y contruidos. Los tipos de datos predefinidos son los 44 tipos distintos que posee la sintaxis XSD por defecto, y son los siguientes:



[4] Tipos de datos de un XSD Schema

Por otro lado, los tipos de datos contruidos son generados a partir de datos predefinidos y aplicándolos restricciones. Se pueden construir datos contruidos a partir de la definición de un elemento predefinido, asignándole un nombre o podemos extender un dato ya existente.

A estos tipos de datos simples se les pueden aplicar restricciones, como seleccionar un rango de números, rango de dígitos que pueda tener el elemento, longitud de caracteres o que solamente pueda tomar el valor de una lista de valores.

Dentro de los elementos complejos podemos distinguir dos tipos: el simple (xs:simpleContent) que se declara cuando el elemento no tiene otros elementos hijos, y el complejo (xs:complexContent) que se declara cuando sí tiene elementos hijos. Dentro de estos elementos, se pueden establecer características que los diferencian y los aproximan al modelaje que estamos realizando, se les llaman indicadores, y podemos distinguir los siguientes: [5]



Indicadores de orden: Asignan el orden de aparición de los elementos

- Secuencia (xs:sequence): Define el orden exacto de aparición de los elementos.
- Alternativa (xs:choice): Define una serie de elementos entre los cuales sólo se puede elegir uno de ellos.
- Todos (xs:all): Define una serie de elementos que pueden aparecer en cualquier orden.

Indicadores de ocurrencia: Asignan cuántas veces puede aparecer dicho elemento.

- Mínimo (minOccurs): Indica el mínimo número de ocurrencias que deben aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <xs:schema>. Va desde 0 hasta ilimitado (unbounded). Por defecto 1.
- Máximo (maxOccurs): Indica el máximo número de ocurrencias que pueden aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <xs:schema>. Va desde 0 hasta ilimitado (unbounded). Por defecto 1.

Indicadores de grupo: Agrupan un conjunto de elementos o de atributos.

- Grupo de elementos (xs:group): Sirve para agrupar un conjunto de declaraciones de elementos relacionados.
- Grupo de atributos (xs:attributeGroup): sirve para agrupar un conjunto de declaraciones de atributos relacionados.

## 2.3.JUSTIFICAR POR QUÉ ES SU INTERÉS.

Actualmente, la cantidad de datos que se genera está aumentando de manera considerable, y es por ello se debe investigar el cómo puede aumentar la velocidad a la que estos se procesan. Por otro lado, se debe demandar un proceso de abstracción lo suficientemente elevado para que independiente de su conocimiento, cualquier usuario sea capaz de realizar esta tarea. Es por ello por lo que el interés principal radica en ser capaz de procesar grandes cantidades de datos de una manera eficaz y sencilla. Resulta interesante el cómo en cuestión de pocos segundos se pueden procesar grandes cantidades de datos, y estos para un posterior uso. A partir de esto se pueden representar de una manera trivial.

Cabe destacar también el cómo se pueden procesar los datos independientemente del formato en el que vengan presentados. Poder pasar de una estructura de datos que una persona tenga dificultad para entender a un conjunto de valores perfectamente ordenados y legibles.

Para ello, haremos uso de bases de datos NoSQL, pues cumplen con los requisitos mencionados anteriormente. Dentro del conjunto de bases de datos NoSQL, se elige Cassandra, una base de datos similar a SQL ya que almacena los datos por columnas mediante un formato clave-valor y destaca por su tolerancia a particiones y disponibilidad. Esto es esencial debido a que la gran cantidad de datos que se van a procesar vendrán definidos en series de tiempo de conexión con el satélite y en un formato en el que el procesamiento de estos datos es ideal realizarlo por columnas en vez de por ficheros como proponen las bases de datos más extendidas NoSQL como MongoDB. Además, Apache Cassandra destaca por su alta capacidad de lectura/escritura, siendo ideal para procesar datos que provienen de un satélite, ya que se tratan de ráfagas de información que llegan de manera muy continuada y necesitamos procesos muy eficientes para su inserción en la base de datos.



En cuanto al uso del lenguaje de marcado XML, cabe destacar su facilidad de comprensión a la hora de que una persona sea capaz de leer un documento, pudiendo asimilar de una forma el tipo de contenido. Además, XML cuenta con la posibilidad de poder transformar el documento en un árbol, de tal manera que a través de Python se pueden procesar estos ficheros de una manera sencilla.

## 3. REQUISITOS DEL PROYECTO

### 3.1. REQUISITOS FUNCIONALES

#### RF-1. Extracción de datos XSD

La herramienta realizada debe ser capaz extraer los datos de telemetría de un esquema XSD para poder representar la información almacenada.

##### RF-1.1.

Se deben extraer del esquema los campos que se refieran con “Housekeeping” de para establecer una correlación con la estructura de la tabla a la que debe pertenecer.

##### RF-1.2.

Se debe equiparar el tipo que se maneja en un documento XSD con el que se maneja en Cassandra y se debe designar de los datos extraídos, cuales deben tener un tratamiento especial: Si precisa que forme parte de la clave primaria, o si debe ser obligatorio que el campo no esté vacío.

#### RF-2. Extracción de datos XML

Dado un fichero XML debemos extraer la información que contiene, la cual, puede venir en función de valor o de atributo.

##### RF-2.1

Se debe ser capaz de establecer una equivalencia entre el tipo de dato que se maneja en XML con el que se maneja en Cassandra.

##### RF-2.2.

Se deben insertar estos datos en la columna que corresponda, para que estos se almacenen de una forma adecuada.

#### RF-3. Inserción automática en la base de datos

Se deben seleccionar los ficheros XML a insertar de manera automática, además del esquema XSD sobre el que nos estamos basando para generar una tabla.

##### RF-3.1

Solamente con ejecutar la herramienta se debe insertar la telemetría deseada.

### 3.2. REQUISITOS NO FUNCIONALES

**RNF-1.** Los datos pertenecientes al fichero XML y al esquema XSD deben ser íntegros, respetando las claves primarias definidas (integridad referencial) y los campos que no pueden tomar un valor de NULL (integridad de dominio).

**RNF-2.** Debe tratarse de una herramienta portable.

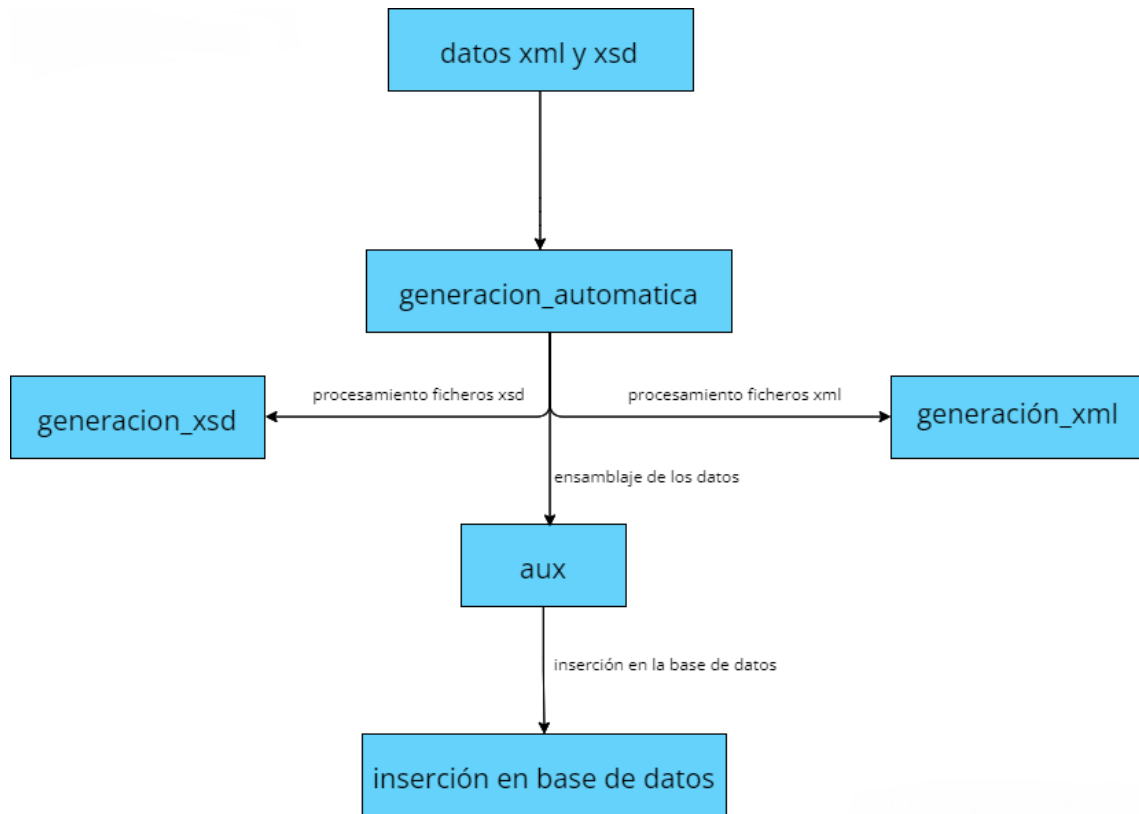
**RNF-3.** Debe ser escalable, dando tiempos razonables de uso para esquemas o ficheros de datos de mayor tamaño.

**RNF-4.** Debe ser reusable para posibles mejoras o ramificaciones partiendo de este código

## 4. DISEÑO DEL SISTEMA/APLICACIÓN

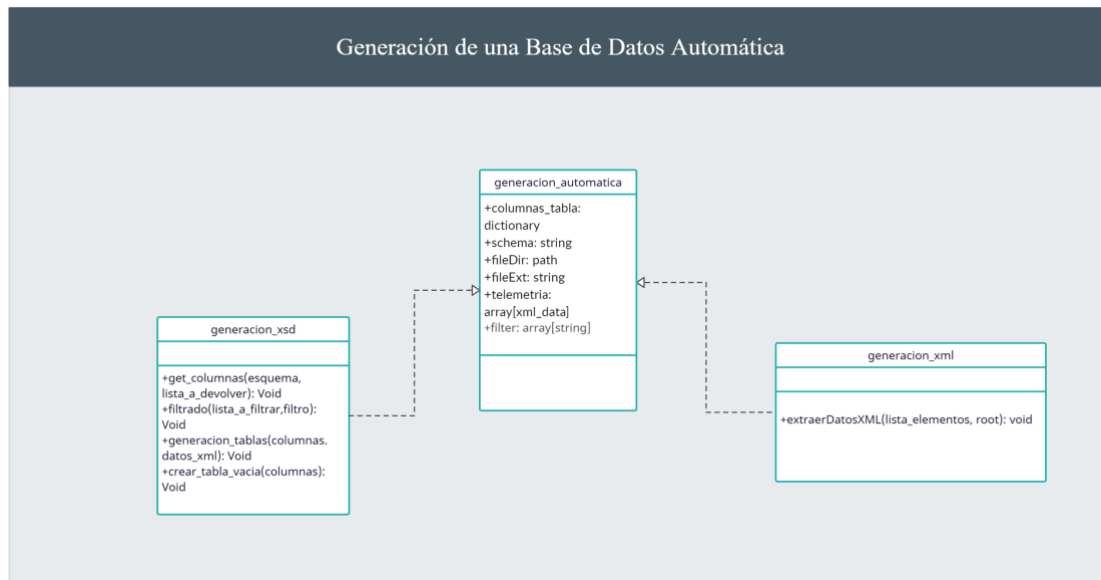
### 4.1. ESTRUCTURACIÓN EN MÓDULOS O PAQUETES

A continuación, se presenta el flujo de datos y como se procesan e interaccionan con los módulos:



*Flujos de los datos*

A través de la descripción del anterior flujo de datos, se realizó un diagrama de clases para poder implementar todas las funcionalidades anteriores en una herramienta. Los distintos módulos que la componen son las siguientes:

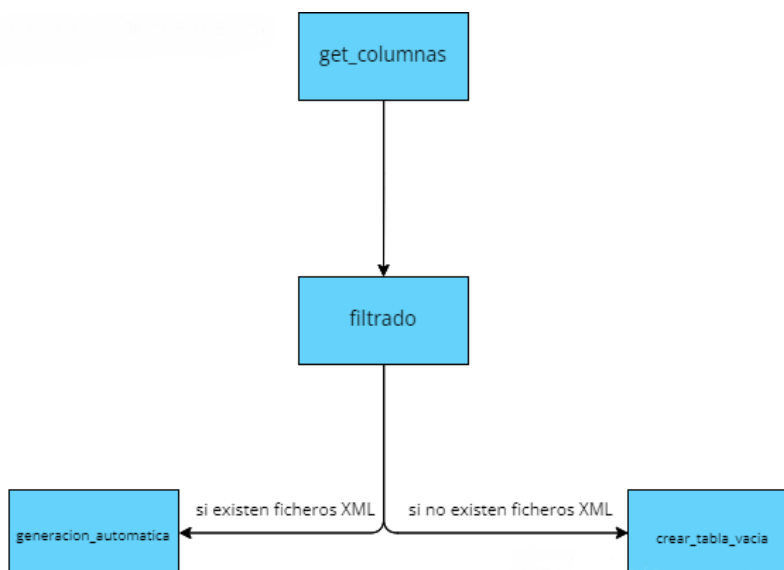


*Diagrama de clases*

Se aprecian los siguientes módulos:

- **generacion\_xsd**: se encarga de extraer las columnas del XSD Schema, ajustar los tipos de datos según lo precise la sintaxis de Cassandra, y crear las tablas ya sean vacías, o con datos de telemetría.
- **generacion\_xml**: se encarga de extraer los elementos de un fichero XML y ordenarlos según lo requiera la tabla, y ajustar los tipos de datos según lo precise la sintaxis de Cassandra.
- **generacion\_automatica**: se encarga de aplicar las funcionalidades de “generacion\_xml” y “generacion\_xsd”. Genera un fichero auxiliar llamado “aux.py” que se encarga de insertar la telemetría en la base de datos.

A través de la herramienta, se genera un fichero auxiliar que se ejecuta y se borra de manera automática. En él se integra toda la información extraída de los ficheros XML y XSD para poder insertar los datos en la tabla. Este fichero se encarga de insertar los datos en la base de datos en el keyspace correspondiente y sincronizar las tablas. En este caso nos estamos centrando en los datos relativos a “Housekeeping”, que contienen información correspondiente sobre su estado del sistema: nivel de batería, su posición relativa o tiempo actual. Para ello, en el archivo llamado “generación\_automatica” se procesan los ficheros XML y XSD. Esto se realiza por separado, en los correspondientes archivos “generacion\_xsd” para los ficheros con formato xsd, y en “generacion\_xml”, para los que tienen formato xml. Posteriormente, se genera un fichero auxiliar llamado “aux” para, a través de la información extraída previamente, proceder a su inserción en la base de datos.



*Funcionalidad de la herramienta*

El esquema XSD incluye varias estructuras de datos a parte de la correspondiente a “Housekeeping”. Teniendo en cuenta que recorreremos todo el fichero extrayendo todas las estructuras de datos, se deben seleccionar aquellos datos que pertenecen a “Housekeeping”. Por lo tanto, de toda la estructura se filtran los elementos deseados. A la hora de la selección de los elementos que componen la tabla, se debe tener en cuenta que aparte de su nombre se necesita también el tipo de elemento y para ello se debe almacenar el valor de este. Cabe destacar que la representación de los datos en un esquema XSD es distinta a la de Cassandra, por tanto, a la hora de la extracción de los datos se realiza una conversión para almacenar el tipo de formato que acepta Cassandra. Tras la realización de todas estas acciones se habrá procesado el esquema XSD y se estará listo para generar la tabla de Cassandra con los nombres de las columnas y su tipo.

Por otro lado, el fichero XML contiene toda la telemetría que debe ser almacenada. Es este caso, la relacionada con “Housekeeping”. Por lo tanto, debemos extraer estos datos y almacenarlos para después procesarlos. Una vez extraídos, para facilitar el trabajo debemos asegurarnos de que, a la hora de almacenar, debemos no solo tener correspondencia con el tipo de dato, sino que también debemos tener en cuenta la posición en la que lo almacenamos, ya que un error puede estar de que insertemos un valor de telemetría en una columna que no le corresponda, aunque el tipado sea el correcto. Para ello, debemos reorganizar los datos para asegurarnos que se vayan a insertar en su columna correspondiente, ya que estos pueden estar en posiciones que no se corresponden. Finalmente, se debe tener en cuenta que podemos procesar más de un fichero XML distinto. Partiendo de que la herramienta que almacena los datos debe estar en un mismo directorio que los ficheros XML y el esquema XSD, procesaremos cada fichero XML en una misma ejecución y posteriormente los borraremos, todo ello de manera automática.

## 4.2.DESARROLLO DE ALGORITMOS NO TRIVIALES.

### 4.2.1. ALGORITMO DISEÑADO PARA OBTENER LAS COLUMNAS

Dentro de un árbol XSD se pueden distinguir dos tipos de datos: los simples (Simplex Type) y los complejos (Complex Type). Los datos del esquema XSD están formados por datos complejos, estos los pueden formar otros datos complejos o datos simples. Los datos simples son los que se van a tener que representar. Es por ello, que se recorrerán todos los datos complejos hasta encontrar los simples. Una vez se haya encontrado este dato simple, se debe comprobar qué tipo de dato es, y establecer una correspondencia entre la representación de este dato en XSD y Apache Cassandra. Finalmente, almacenamos el resultado en un diccionario: Siendo la clave el nombre de la columna, y el valor el tipo de dato representado según Apache Cassandra. Un ejemplo visual de un esquema XSD puede ser este:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://tempuri.org/PurchaseOrderSchema.xsd"
  targetNamespace="http://tempuri.org/PurchaseOrderSchema.xsd"
  elementFormDefault="qualified">
  <xsd:element name="PurchaseOrder" type="tns:PurchaseOrderType"/>
  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="ShipTo" type="tns:USAddress" maxOccurs="2"/>
      <xsd:element name="BillTo" type="tns:USAddress"/>
    </xsd:sequence>
    <xsd:attribute name="OrderDate" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:integer"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
  </xsd:complexType>
</xsd:schema> [6]
```

Donde el árbol formado sería un elemento raíz llamado “PurchaseOrder” de tipo “PurchaseOrderType”. De este nodo cuelgan dos hojas llamadas “ShipTo” y “BillTo” de tipo “USAddress”. Este a su vez formado por elementos simples: “name”, “street”, “city”, “state” y “zip”. Los cuatro primeros son “strings” y el último un “integer”.

En este caso, el algoritmo desarrollado es el siguiente:

```
def get_columnas(esquema, lista_a_devolver):
    nodos_a_recorrer = []
    for n in esquema.iterchildren():
        nodos_a_recorrer.append(n)
    while len(nodos_a_recorrer) > 0:
        nodo_actual = nodos_a_recorrer[0]
        nodos_a_recorrer = nodos_a_recorrer[1:]
        if nodo_actual.type.is_complex() or nodo_actual.type.is_list() or nodo_actual:
            for n in nodo_actual.iterchildren():
                nodos_a_recorrer.append(n)
        else:
            if nodo_actual.type.base_type:
                if str(nodo_actual.type.base_type.local_name) == 'unsignedInt' or str(nodo_actual.type.base_type.local_name) == 'integer' or str(nodo_actual.type.base_type.local_name) == 'unsignedLong' or str(nodo_actual.type.base_type.local_name) == 'integer':
                    lista_a_devolver[nodo_actual.local_name] = 'Integer'
                elif str(nodo_actual.type.base_type.local_name) == 'string':
                    lista_a_devolver[nodo_actual.local_name] = 'Text'
                elif str(nodo_actual.type.base_type.local_name) == 'boolean':
                    lista_a_devolver[nodo_actual.local_name] = 'Boolean'
            else:
                if str(nodo_actual.type.local_name) == 'unsignedInt' or str(nodo_actual.type.local_name) == 'integer' or str(nodo_actual.type.local_name) == 'unsignedLong' or str(nodo_actual.type.local_name) == 'float':
                    lista_a_devolver[nodo_actual.local_name] = 'Integer'
                elif str(nodo_actual.type.local_name) == 'string':
                    lista_a_devolver[nodo_actual.local_name] = 'Text'
                elif str(nodo_actual.type.local_name) == 'boolean':
                    lista_a_devolver[nodo_actual.local_name] = 'Boolean'
```

*Captura del algoritmo*

#### 4.2.2. ALGORITMO PARA OBTENER LOS DATOS DE UN FICHERO XML

Para extraer los datos de telemetría de los ficheros XML se debe saber cómo está almacenada: Estos irán almacenados como valor o como atributo. Se recorre el árbol eliminando todos los elementos que lo componen: como “\n” o “None” y se van almacenando los datos que se quieren. Por otro lado, se debe tener en cuenta que la posición de la lista en la que se va a almacenar el valor debe corresponder con la posición del nombre de la columna en el diccionario. Para ello, se reorganizan los datos para que se asemeje a su posición en el diccionario.

```
def extraerDatosXML(lista_elementos, root):
    for elem in root.findall('.//'):
        if "\n" not in str(elem.text) and "None" not in str(elem.text):
            if str(elem.text).isnumeric():
                lista_elementos.append(int(elem.text))
            elif str(elem.text) == "true":
                lista_elementos.append(True)
            elif str(elem.text) == "false":
                lista_elementos.append(False)
            else:
                lista_elementos.append(str(elem.text))
    cambio = lista_elementos.pop(60)
    lista_elementos.insert(2, cambio)
    lista_elementos.insert(0, int(root.find('.//tm/hk_tm_type').attrib["sequencecount"])) #Extraemos el número de secuencia (se pasa como atributo)
    lista_elementos.insert(0, "Housekeeping")
```

*Captura del algoritmo*

#### 4.2.3. GENERACIÓN DE TABLAS

Aquí se genera un fichero auxiliar para poder insertar los datos en Cassandra a través de Python de manera más cómoda. Para ello, primero se deben importar las librerías necesarias. Posteriormente se genera la tabla recorriendo el diccionario que almacena los nombres de las columnas y sus valores. Además, se abre el keyspace y se sincroniza la tabla para poder almacenar datos en ella. Finalmente, se genera la función que inserta los datos extraídos del XML en la tabla en Cassandra.

```
for xml in os.listdir(fileDir):
    if xml.endswith(fileExt):
        tree = ET.parse(xml)
        root = tree.getroot()
        generacion_xml.extraerDatosXML(telemetria,root)
        generacion_xsd.generacion_tablas(columnas_tabla,telemetria)
        exec(open("aux.py").read())
        call(['rm','aux.py'])
        call(['rm',xml])
        telemetria = []
    else:
        generacion_xsd.crear_tabla_vacia(columnas_tabla)
        call(['rm','aux.py'])
print("Telemetría insertada de manera correcta.")
```

*Captura del algoritmo*

## 5. IMPLEMENTACIÓN

Se detalla la realización de cada uno de los elementos que componen la herramienta, así como su uso.

### 5.1.INSTALACIÓN DEL ENTORNO DE DESARROLLO

En primera instancia, se deben instalar las dependencias de Python para poder ejecutar la herramienta, también se debe instalar Apache Cassandra:

- *sudo apt update*
- *sudo apt install Python*
- *sudo apt install -y Cassandra*

Posteriormente, dentro de los distintos ficheros de la herramienta se utilizan las siguientes bibliotecas:

- El módulo *subprocess*, que lo debemos declarar con la sentencia: *from subprocess import call*, para poder procesar comandos de terminal en fichero
- La biblioteca *os*, para poder trabajar con rutas de ficheros
- La biblioteca *etree*, mediante la sentencia *import xml.etree.ElementTree as ET*, para poder recorrer los ficheros XML y XSD.

Una vez instalado Cassandra, podemos acceder a su consola a través del comando “*cqlsh*” en la terminal. Sin embargo, en otra terminal se debe arrancar Cassandra con el comando “*cassandra -f*”. Con estas sentencias, ya se tiene un acceso completo a un nodo local de Cassandra, donde se almacenarán los datos procesados por la herramienta.

### 5.1.CÓDIGO

La ejecución del código se basa en tres parámetros: la primera llamada telemetría. Una lista que almacenará la telemetría que se extrae del XML y que se actualizará con cada fichero de telemetría disponible. La segunda recibe el nombre de “*columnas\_tabla*”. Esta variable es un diccionario que recibe como clave el nombre de la columna de la tabla, y como valor el tipo de dato que se debe almacenar en la columna. Este dato irá almacenado según se



define en Cassandra y no en XSD, y para ello se deberá realizar una conversión. Finalmente, se tiene una variable que almacenará el esquema XSD que se usa llamada “schema”.

El primer paso es extraer el nombre del esquema, para ello hay que moverse a la ruta en la que se encuentre la herramienta y buscar de entre todos los archivos cual es el que tiene una extensión XSD y almacenar este nombre en la variable “schema”.

Posteriormente, se debe extraer de este la información correspondiente a las columnas que formarán la tabla “Housekeeping” y sus tipos de datos. Para ello, se llama a la función `get_columns()` que recibe como parámetros el diccionario “columns\_tabla” y la variable “schema”. Esta función se encarga de recorrer todo el esquema como si fuese un árbol, extrayendo todos los parámetros simples que lo componen, así como su tipo y almacenándolos en la variable “columns\_tabla”. El tipo viene por defecto según se expresa en XSD, y se quiere expresar según lo vamos a definir en Cassandra. Es por ello por lo que en este punto se realiza la conversión de formatos pasando de una definición del tipo de variable de XSD a Cassandra.

En este caso, el esquema que se tiene contiene la información de varias tablas distintas y se está trabajando con la de Housekeeping, por lo tanto, de entre todas las columnas extraídas se deben seleccionar las que pertenezcan a “Housekeeping” y a través de una función llamada filtrado y pasando como parámetros todas las claves del diccionario columns\_tabla (que se corresponde con los nombres de todas las columnas obtenidas) y un array llamado filter que contiene las columnas que pertenecen a Housekeeping. El resultado es un diccionario que contiene las columnas y el tipo de dato que formarán la tabla Housekeeping. Con esto se tiene todo lo necesario para generar la tabla.

A continuación, se debe extraer la telemetría de los ficheros XML. Se hará de uno en uno y para ello se procede de manera similar a como se ha obtenido el nombre del esquema: se debe mover a el directorio en el que se encuentra la herramienta y se recorren todos los archivos con extensión XML, almacenando su nombre. Una vez obtenido el nombre del archivo XML se genera su árbol y se extrae su raíz. Posteriormente, se llama a la función `extraerDatosXML()` que recibe dos parámetros: el array telemetría que almacenará la información correspondiente a la telemetría extraída del fichero XML, y la raíz del árbol del fichero XML que se esté procesando. Se recorre el árbol almacenando toda la telemetría y finalmente, se debe reordenar el array para que se puedan insertar de manera secuencial los elementos. Por último, se extrae el número de secuencia, que viene almacenado en forma de atributo y se inserta en su posición correspondiente tomando siempre como referencia el orden en el que se ha extraído la tabla.

Una vez habiendo rellenado las variables que almacenan la tabla y la telemetría se llama a la función `generacion_tablas()`, que se encarga de generar un fichero auxiliar llamado aux.py. Esta herramienta importa las librerías de cqlengine que facilitan el uso de Cassandra con Python y generamos un modelo a partir del diccionario que contiene las columnas. Las columnas que precisan de algún parámetro especial se deben generar personalmente, pero el resto de los datos se generan recorriendo el diccionario y representando su nombre y su tipo de dato. Una vez hecho se establece conexión con el keyspace y se sincroniza la tabla. Finalmente, se recorre el array que almacena la telemetría y se almacenan los datos recorriendo a la vez los nombres de las columnas y generando el método “create” de manera automática. Cabe destacar que, al extraer datos

numéricos de un XML a través de un árbol, se hace siempre en formato string. Por lo tanto, para que Cassandra pueda almacenar los datos numéricos, hay que transformar los strings que contienen números a tipo entero mediante la función “Type” que viene por defecto en Python.

Una vez generado el fichero auxiliar, volvemos al fichero original y solamente queda continuar el flujo del programa para insertar la telemetría en la base de datos y borrarlo, para poder volverlo a generar con los datos del siguiente fichero XML. Los ficheros XML se borrarán después de haber sido implementados dentro de la base de datos y tendremos que volver a poner el array que almacena la telemetría en blanco, para que almacene la información del siguiente fichero XML a procesar. Si no existen ficheros XML a procesar, solamente se genera la tabla “Housekeeping” con los datos correspondientes.

## 5.2.USO

Para el uso de esta herramienta se debe tener en un mismo directorio el fichero `generacion_automatica.py`, el esquema XSD y los ficheros de telemetría en formato XML que se quieren insertar.

Una vez hecho esto, debemos ejecutar la herramienta de Python escribiendo por terminal: “python3 `geracion_automatica.py`”. Esta se ejecutará generando la tabla a partir del archivo XSD, extrayendo el contenido de los XML, e insertando la telemetría en la base de datos. Los ficheros XML se eliminarán tras la ejecución del código.

# 6. PRUEBAS

Para las pruebas del código se han desarrollado una serie de pruebas unitarias que probarán las funcionalidades por separado, y finalmente una prueba de sistema, la cual consiste en la prueba del código de forma general.

## 6.1.PRUEBAS UNITARIAS

### 6.1.1. GENERACIÓN DE TABLAS DE MANERA AUTOMÁTICA.

Verificamos que se generan tablas en nuestro keyspace local a través de la biblioteca que Python tiene para interactuar con Cassandra. La salida debe ser una tabla generada con los campos que se han detallado en el código.

En el anexo 1 se representa de manera manual la estructura de la tabla, definiendo todas las columnas que debe tener la tabla, con el tipo de dato que tiene cada columna y los campos especiales que puede tener cada una: si forma parte de la clave primaria, si tiene un valor por defecto y si el valor almacenado debe ser distinto de NULL. Todo esto se almacenará en un keyspace de nombre prueba situado en la dirección local de la máquina y previamente se necesita importar las librerías necesarias para su funcionamiento.

Para realizar esta prueba, se ejecuta el código mediante la sentencia “python3 creacion\_tabla.py” y el resultado esperado debe ser una tabla de nombre “Housekeeping” guardada en el keyspace. El resultado y la tabla se pueden ver en el anexo 1.

### 6.1.2. INSERCIÓN DE DATOS DE MANERA AUTOMÁTICA.

Verificamos que a partir de un fichero XML podemos extraer sus datos y almacenarlos en una base de datos. Si se parte de una tabla vacía, la prueba será correcta si tras ejecutar una instrucción para obtener la información de una tabla, aparecen los datos almacenados previamente.

Para esta prueba, se parte de una tabla de menores dimensiones para simplificar la prueba, puesto que, si se es capaz de insertar valores, se deberá poder insertarlos independientemente de la cantidad que se quiera. Es por ello, que se parte del código situado en el anexo 2.

En este caso, se está generando una tabla con tres columnas como en el caso anterior, y ahora se están insertando un valor de tipo UUID generado por el sistema y dos columnas que guardan un nombre y un apellido: Javier y García. Por lo tanto, ejecutamos el código “python3 prueba\_insertar\_datos.py” y se visualiza el resultado en el keyspace.

En el anexo 2 se aprecia como ahora se tienen dos tablas, la creada en la prueba anterior y la nueva tabla llamada personas y, al seleccionar los datos existentes, se ve que aparece lo esperado: un valor UUID y el nombre de Javier y el apellido García situados en sus respectivas columnas.

### 6.1.3. SELECCIÓN DE FICHEROS XML:

Esta prueba consiste en comprobar que se seleccionan todos los ficheros XML que contengan datos de telemetría para poder ser procesados a través de esta herramienta. Cabe recordar que la herramienta y los ficheros deben estar en el mismo directorio. Para ello, se presenta en el anexo 3 el extracto de código.

El funcionamiento es el siguiente: se debe estar en la ruta donde se encuentra la herramienta, y en la que deben estar también los archivos XML y se seleccionan para que se almacenen en una lista que se debe imprimir. El resultado será correcto si se consigue imprimir los nombres de todos los ficheros XML.

En el anexo 3 podemos ver que la herramienta de la prueba se encuentra en el mismo directorio que los archivos XML, y el resultado es el esperado, pues por la terminal podemos apreciar que la lista contiene el nombre de todos los archivos XML.

## 6.2.PRUEBA DE SISTEMA

En este caso, se va a probar el funcionamiento de la herramienta al completo. Esta prueba va a consistir en el ensamblaje de los componentes de nuestra herramienta procesando seis ficheros XML diferentes de forma simultánea. Previamente también deberá generar la tabla a partir del fichero XSD y finalmente debe eliminar los ficheros XML tras haberlos procesado. Para ello se parte del código original, situado en el anexo 4.

Se ejecuta la herramienta abriendo la terminal e insertando el comando “Python3 generacion\_automatica.py”. Se debe tener otra terminal abierta y para haber iniciado Cassandra con el comando “cassandra -f”. Al ejecutarlo, se genera un fichero auxiliar que se borra automáticamente cuando se inserta un archivo, este fichero se llama “aux.py”

Cuando acaba de procesar todos los ficheros XML se recibe un mensaje por terminal: “Telemetría insertada de manera correcta”.

Se abre una nueva terminal para acceder a Cassandra y observar si los datos se han insertado en la tabla “Housekeeping”. Para ello, se abre Cassandra con el comando “cqlsh” y se abre el keyspace donde se ejecutan las pruebas. Por ello, se inserta el comando “use prueba”. Una vez dentro del keyspace, se va a ver primero cuántas filas tenemos, para ello, se inserta el comando “SELECT COUNT (\*) FROM housekeeping;”.

El resultado se observa en el anexo 4. Se verá si los datos se han insertado en la tabla. Para ello, se muestran todos con el comando “SELECT \* FROM housekeeping;”. Finalmente, se revisa si se han eliminado los archivos XML que ya han sido procesados:

Se aprecia que se han insertado todos los elementos en la tabla “Housekeeping” y además se muestra también el número de filas, en este caso seis.

## 7. CONCLUSIONES Y LÍNEAS FUTURAS

Finalmente, se redactan unas conclusiones donde se exponen los resultados del Trabajo de Fin de Grado realizado, una valoración personal y posibles mejoras y líneas futuras.

### 7.1. CONCLUSIONES

Con este Trabajo de Fin de Grado, se ha logrado el objetivo de automatizar la inserción de la telemetría enviada del satélite UPM-Sat2 a una base de datos en Apache Cassandra. Para ello, se hace uso de una herramienta desarrollada en Python que extrae el contenido de los ficheros que contiene la telemetría, en formato XML, y gracias a su propio esquema XSD, se es capaz de almacenarlos en la base de datos. Cabe destacar que la herramienta no tiene un uso exclusivo para un satélite en particular, si no para cualquier sistema que precise de la funcionalidad de generar una tabla en Cassandra a partir de un esquema XSD.

Para ello, previamente, se ha estudiado como será el flujo de los datos del problema presentado, para en consonancia, plantear un modelo que sea capaz de cumplir con los requisitos expuestos.

En cuanto al desarrollo personal, la realización de este Trabajo de Fin de Grado me ha servido en primera instancia para aprender a utilizar Python, lenguaje que hasta entonces no había utilizado, y que se utiliza en el mundo laboral. Por otro lado, me he introducido en el mundo de las bases de datos NoSQL. He podido aprender sobre sus usos y ventajas con respecto a las bases de datos SQL. Además, he ampliado mi conocimiento sobre los lenguajes de marcado y he aprendido a utilizar otras tecnologías como son los esquemas para validar documentos. Me ha servido para mejorar mi capacidad de resolver problemas mediante algoritmos al haberseme planteado un problema desde cero y teniendo total libertad para poder resolverlo. Es por ello, que la realización de este Trabajo de Fin de Grado es una experiencia de aprendizaje plenamente positiva.

Los principales problemas a la hora de la realizar este trabajo han sido los siguientes: En primer lugar, se requerían unos conocimientos que yo no tenía. Como he mencionado anteriormente, tuve que aprender Python y a manejar bases de datos NoSQL. Además, se partió de una primera idea para realizar el proyecto, como fue utilizar una biblioteca llamada “GenerateDS” para generar todas las estructuras de datos, para posteriormente, optar por desarrollar la herramienta formando dos árboles que se recorren para extraer los datos requeridos: el del esquema XSD, y el del fichero XML que se desee procesar. Finalmente, a la hora de desarrollar el algoritmo capaz de satisfacer los requisitos de uso existieron problemas a la hora de desarrollarlo, que se solucionaron a base de pruebas de las correspondientes funcionalidades que daban problemas.

Las pruebas finales del software se han realizado con los campos de telemetría “Housekeeping”, que contienen información sobre el estado del satélite. Todos los requisitos presentados han sido concluidos de manera favorable: Los datos con los que se trabajan son íntegros y se ha probado la escalabilidad procesando distintas cantidades de archivos a procesar. Por otro lado, se ha probado la herramienta en diferentes sistemas operativos para asegurar su portabilidad.

## 7.2. LÍNEAS FUTURAS

Se presentan una serie de líneas a seguir para la mejora de la herramienta y el desarrollo de la extracción de la automatización de los procesos del satélite UPMSat2.

En primer lugar, estudiar la posible mejora de la herramienta en términos de eficiencia, puesto que en un ordenador portátil no es capaz de procesar de manera efectiva todos los ficheros de telemetría que se manejan de media en un día, unos 13.000 ficheros. Sería recomendable sustituir el lenguaje en el que la herramienta ha sido desarrollado por uno que se pudiera paralelizar, como pudiera ser Scala, un lenguaje de programación funcional. De esta manera, podemos repartir la carga de trabajo entre varios nodos, haciendo más eficiente el procesamiento de la telemetría.

Se puede sustituir el algoritmo utilizado para recorrer árboles por algunos que sean menos complejos y más eficientes. Por otro lado, se deben hacer ajustes manuales para asignar a cada elemento de la tabla generada en Apache Cassandra con su correspondiente dato correcto de telemetría. Una posible línea de estudio podría ser que la herramienta fuese capaz de reconocer los valores que le deben pertenecer a cada columna de manera automática.

Finalmente, una última mejora es la de desarrollar una interfaz para poder seleccionar que elementos del esquema XSD deseamos extraer, en vez de hacerlo de manera manual. De esta forma, se puede decidir de una forma más elegante que datos de telemetría deseamos insertar, y en donde se quieren almacenar.

## 8. BIBLIOGRAFÍA

- [1] *El proyecto UPMSat-2*. (s. f.). Inicio. <https://www.idr.upm.es/index.php/es/el-proyecto-upm-sat-2>
- [2] PandoraFMS. (2022, 18 de febrero). NoSQL vs SQL: principales diferencias y cómo elegir. Pandora FMS - The Monitoring Blog. <https://pandorafms.com/blog/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/>
- [3] Gildder. (2018, 31 de enero). *Teorema CAP*. [https://gildder.medium.com/. https://gildder.medium.com/teorema-cap-e99d66fde6a0](https://gildder.medium.com/.https://gildder.medium.com/teorema-cap-e99d66fde6a0)
- [4] Morales, R. (2015, 14 de mayo). *Qué son los esquemas XSD*. TicArte. <https://www.ticarte.com/contenido/que-son-los-esquemas-xsd>
- [5] Microsoft XML Core Services. (s. f.). *XML Data Types Reference - MSXML 5.0 SDK Documentation*. Documentation & Help. [https://documentation.help/MSXML-5.0-SDK-2/xsd\\_ref\\_5bc5.htm](https://documentation.help/MSXML-5.0-SDK-2/xsd_ref_5bc5.htm)

- [6] *Ejemplo de archivo XSD: esquema simple - Visual Studio (Windows)*. (27 de septiembre de 2022). Microsoft Learn: Build skills that open doors in your career. <https://docs.microsoft.com/es-es/visualstudio/xml-tools/sample-xsd-file-simple-schema?view=vs-2022>
- [7] Access SQL: conceptos básicos, vocabulario y sintaxis. (s. f.). Microsoft Support.
- [8] Base de datos: SQL y NoSQL. (2020, 28 de abril). Cloud Computing y Servicios Gestionados IT. <https://www.ilimit.com/blog/base-de-datos-sql-nosql/>
- [9] IBM Cloud Education. (2019, 19 de noviembre). cap-theorem. IBM - Deutschland | IBM. <https://www.ibm.com/es-es/cloud/learn/cap-theorem>
- [10] Cupas, C. (2021, 25 de octubre). *Qué es el Teorema CAP y cómo afecta al elegir la BBDD*. OpenWebinars.net. <https://openwebinars.net/blog/que-es-el-teorema-cap-y-como-afecta-al-elegir-la-base-de-datos/>
- [11] Mesa, A. R. (2019, 17 de junio). *Qué es Apache Cassandra*. OpenWebinars.net. <https://openwebinars.net/blog/que-es-apache-cassandra/>
- [12] \_\_\_\_\_jumpstartCS. (2020, 31 de agosto). *Apache Cassandra Tutorial*. <https://www.youtube.com/playlist?list=PLalrWAGybpB-L1PGA-NfFu2uiWHEsdscD>
- [13] Datastax. (2012). *cqlengine documentation — cqlengine 0.21.0 documentation*. cqlengine documentation — cqlengine 0.21.0 documentation. <https://cqlengine.readthedocs.io/en/latest/>
- [14] La arquitectura de Cassandra. (2019, 3 de marzo). Emanuel Goette, alias Crespo. <https://emanuelpeg.blogspot.com/2019/03/la-arquitectura-de-cassandra.html>
- [15] M Criado, J. (2005, 27 de octubre). Componentes de un documento XML. Home de DesarrolloWeb.com. <https://desarrolloweb.com/articulos/2228.php>
- [16] Package API — xmlschema 1.10.0 documentation. (s. f.). xmlschema Documentation — xmlschema 1.10.0 documentation. <https://xmlschema.readthedocs.io/en/latest/api.html>
- [17] Alonso, A. (s. f.). El segmento de tierra del satélite UPMSat-2. dit.upm.es. <https://www.dit.upm.es/~str/papers/pdf/alonso&19.pdf>
- [18] Kuhlman, Dave. "GenerateDS -- Generate Data Structures from XML Schema." GenerateDS -- Generate Data Structures from XML Schema, 3 Oct. 2022, <https://www.davekuhlman.org/generateDS.html>.
- [19] Del Alamo, J.M. (2019) "Information Systems." Madrid: Jose M Del Alamo.
- [20] JavaTPoint DBMS Three Schema Architecture - javatpoint, [www.javatpoint.com](http://www.javatpoint.com). Available at: <https://www.javatpoint.com/dbms-three-schema-architecture> (Accessed: September 23, 2022).
- [21] MariaDB *Comandos SQL básicos*, *MariaDB KnowledgeBase*. Available at: <https://mariadb.com/kb/es/basic-sql-statements/> (Accessed: November 18, 2022).
- [22] Barra, E. (2019) "PDF." Madrid: Enrique Barra.
- [23] Piperlab, S.el A. (2020) *Acid*, *PiperLab*. Available at: <https://piperlab.es/glosario-de-big-data/acid/> (Accessed: November 25, 2022).



- [24] Acens (no date) *Bases de Datos NoSQL. Qué son y tipos que nos Podemos encontrar - acens, acens.com.* Available at: <https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf> (Accessed: September 18, 2022).
- [25] Morales, R. (2018) *Qué son Los Lenguajes de Marcas, Inicio.* Available at: <https://www.ticarte.com/contenido/que-son-los-lenguajes-de-marcas> (Accessed: November 25, 2022).
- [26] Baeldung (2022) *Cassandra Partition Key, composite key, and clustering key, Baeldung.* Available at: <https://www.baeldung.com/cassandra-keys> (Accessed: November 25, 2022).
- [27] Tutorialspoint (no date) *Cassandra - Shell commands, Tutorialspoint.* Available at: [https://www.tutorialspoint.com/cassandra/cassandra\\_shell\\_commands.htm](https://www.tutorialspoint.com/cassandra/cassandra_shell_commands.htm) (Accessed: November 25, 2022).
- [28] Lynch, J. (2016) *Yelp, Monitoring Cassandra at Scale.* Available at: <https://engineeringblog.yelp.com/2016/06/monitoring-cassandra-at-scale.html> (Accessed: November 25, 2022).
- [29] COIT. “Pleno Empleo Para Los Ingenieros De Telecomunicación.” *Pleno Empleo Para Los Ingenieros De Telecomunicación*, 29 Nov. 2017, <https://www.coit.es/noticias/pleno-empleo-para-los-ingenieros-de-telecomunicacion#:~:text=El%2042%25%20de%20encuestados%20trabaja%20en%20empresas%20privadas%20multinacionales.&text=El%20sueldo%20medio%20de%20los,situ%C3%A1ndose%2052.711%E2%82%AC%20brutos%20Fa%C3%B1o.>



## ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

### A.1 INTRODUCCIÓN

Se puede clasificar este proyecto en el ámbito de las ciencias, concretamente en el sector tecnológico, en el departamento de sistemas telemáticos. El objetivo del proyecto es automatizar la inserción de distintos datos de telemetría dentro de una base de datos. Para ello, primero se deben procesar los ficheros XML y XSD para obtener los datos de la telemetría y los nombres y tipos de las columnas de las tablas.

### A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Presentamos los principales impactos relevantes que han supuesto la realización de este proyecto:

---

#### 8.1.1. ASPECTOS ÉTICOS

Los datos que se tratan están relacionados con medidas tomadas por el satélite UPMSat2 de aspectos internos como puede ser su nivel de batería, o externos, como puede ser su orientación. Es por ello, que son datos que se pueden usar con fines tanto educativos como de investigación, no pudiendo afectar a la privacidad de las personas.

---

#### 8.1.2. ASPECTOS ECONÓMICOS

Como se ha automatizado el proceso de la inserción de los datos de telemetría en una base de datos reducimos el coste de personal, al haber sustituido un procedimiento manual, por uno automatizado.

---

#### 8.1.3. ASPECTOS SOCIALES

Con este Trabajo de Fin de Grado, se pretende mejorar la eficiencia en cuestión del almacenamiento de la telemetría. Se podrán procesar una gran cantidad de ficheros de telemetría de forma casi inmediata facilitando el trabajo a los compañeros del proyecto UPMSat2.

---

#### 8.1.4. ASPECTOS AMBIENTALES

Al tratarse de un proceso automático, que se puede ejecutar a lo largo del día, se puede alimentar el sistema que lo ejecuta mediante luz generada por placas solares, reduciendo el gasto eléctrico y aportando a la reducción de la huella de carbono.

### A.3 CONCLUSIONES

Valorar el proyecto desde un punto de vista ético, social, económico y medioambiental y justificar si el uso de criterios de sostenibilidad ha aportado o puede aportar valor añadido al proyecto.

Recapitulando todo lo mencionado en el apartado previo, se afirma que este Trabajo de Fin de Grado ayuda a:

- Facilitar las labores de educación e investigación
- Reducir costes de producción mejorando la eficiencia de los procesos de recopilación de datos
- Ayudar a reducir la huella de carbono

## ANEXO B: PRESUPUESTO ECONÓMICO

A continuación, se dividen y evalúan los distintos costes económicos que han supuesto la realización de este proyecto.

### 8.2. COSTES DE PERSONAL

Para comenzar, se indica el presupuesto que requiere contratar a un estudiante de ingeniería de telecomunicaciones para la realización de este trabajo en función de las horas requeridas para la realización del trabajo. Según el Colegio Internacional de Ingenieros de Telecomunicación (COIT), el salario medio de un ingeniero asciende a los 52.711€ brutos/año [29]. Si se realizan los cálculos, esto es una media de 26,35 €/hora. Aproximando el salario a los 17 €/hora debido a que se trata de un becario el que realizará esta tarea, y que las horas totales son las siguientes:

- Estudio Python (5 horas)
- Estudio Bases de Datos NoSQL (25 horas)
- Estudio Apache Cassandra (30 horas)
- Desarrollo de la herramienta (150 horas)
- Redacción de la memoria (90 horas)

Esto hace un total de 300 horas de trabajo. Los resultados obtenidos se reflejan en la siguiente tabla:

	Coste (€)	Horas (h)	Total (€)
<b>Becario</b>	17 €	300 h	5.100 €

### 8.3.COSTES MATERIALES

En cuanto a los costes materiales: Toda la documentación ha sido extraída de internet o de diapositivas de clases del Grado teniendo un coste nulo. Sin embargo, para la confección de la herramienta y la redacción de la memoria, se ha usado un ordenador MacBookPro de 2019 con un procesador i5 de cuatro núcleos y 8 GB de memoria RAM. Siendo su coste de 1.500€.

	Precio Inicial (€)	Uso (meses)	Amortización (años)	Total (€)
<b>MacBookPro 2019</b>	1.500€	8 meses	5 años	150 €

### 8.4.COSTES TOTALES

Finalmente, para el cálculo de los costes totales, tomamos los costes de material y personal y se les aplica el 21% correspondiente al IVA. El resultado se refleja en la siguiente tabla:

	Coste (€)
<b>Gasto en personal</b>	5.100 €
<b>Gasto en material</b>	150 €
<b>Subtotal</b>	5.250 €
<b>IVA</b>	1102,5 €
<b>Gasto Total</b>	<b>6202,5 €</b>

## 8.5.ANEXO 1. GENERACIÓN DE TABLAS DE MANERA AUTOMÁTICA.

*Tabla generada de manera manual*

```
→ ~ git:(master) ✗ cqlsh
/usr/local/Cellar/cassandra/4.0.3/libexec/bin/cqlsh.py:460: DeprecationWarning:
Legacy execution parameters will be removed in 4.0. Consider using execution p
rofiles.
/usr/local/Cellar/cassandra/4.0.3/libexec/bin/cqlsh.py:490: DeprecationWarning:
Setting the consistency level at the session level will be removed in 4.0. Con
sider using execution profiles and setting the desired consistency level to the
EXEC_PROFILE_DEFAULT profile.
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.0.0 | Cassandra 4.0.3 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
cqlsh> use prueba;
cqlsh:prueba> describe tables;

housekeeping

cqlsh:prueba> █
```

*Tabla vista en la terminal de Cassandra*

```

CREATE TABLE prueba.housekeeping (
  sequencecount int PRIMARY KEY,
  batt_t_ext_tm int,
  batt_t_int_tm int,
  batt_tbat1_tm int,
  batt_tbat2_tm int,
  batt_tbat3_tm int,
  batt_vbat_tm int,
  batt_vbus_tm int,
  battery_status text,
  boom1_vbus boolean,
  boom2_vbus boolean,
  current_operating_mode text,
  current_time int,
  das_n15v boolean,
  das_p15v boolean,
  das_p3v boolean,
  das_p5v boolean,
  ebox_t_ext_tm int,
  ebox_t_int_tm int,
  mgm1_p5v boolean,
  mgm1_t_tm int,
  mgm1_x_tm int,
  mgm1_y_tm int,
  mgm1_z_tm int,
  mgm2_p5v boolean,
  mgm2_t_tm int,
  mgm2_x_tm int,
  mgm2_y_tm int,
  mgm2_z_tm int,
  mgm3_n15v boolean,
  mgm3_p15v boolean,
  mgm3_t_tm int,
  mgm3_x_tm int,
  mgm3_y_tm int,
  mgm3_z_tm int,
  mgt_tx_tm int,
  mgt_x_vbus boolean,
  modem_t_tr_tm int,
  modem_vbus boolean,
  mts_pitts1_tm int,
  mts_pitts2_tm int,
  mts_pitts3_tm int,
  mts_pitts4_tm int,
  mts_pitts5_tm int,
  mts_pitts6_tm int,
  mts_vbus boolean,
  n15v_tm int,
  obo_t_tm int,
  p15v_tm int,
  p3v3_tm int,
  p5v_tm int,
  pdu_lvbus_tm int,
  pdu_p3v3 boolean,
  pdu_p5v boolean,
  psu_n15v_tm int,
  psu_ip15v_tm int,
  psu_ip3v3_tm int,
  psu_ip5v_tm int,
  psu_t_tm int,
  pv_ispxn_tm int,
  pv_ispxp_tm int,
  pv_ispyr_tm int,
  pv_ispyr_tm int,
  pv_ispyp_tm int,
  pv_ispyp_tm int,
  pv_tpsxn_tm int,
  pv_tpsxp_tm int,
  pv_tpsyn_tm int,
  pv_tpsyp_tm int,
  pv_tpsyp_tm int,
  pv_tpsyp_tm int,
  rw1_t_tm int,
  rw2_t_tm int,
  rm_p5v boolean,
  rm_vbus boolean,
  sma_sb01 boolean,
  sma_sb02 boolean,
  ss6_xn_tm int,
  ss6_xp_tm int,
  ss6_yn_tm int,
  ss6_yp_tm int,
  ss6_zn_tm int,
  ss6_zp_tm int,
  temp_e_p5v boolean,
  temp_b_p5v boolean,
  tm_type text,
  ttc_stat boolean
) WITH additional_write_policy = '99p'

```

Tabla vista en la terminal de Cassandra

## 8.6. ANEXO 2. INSERCIÓN DE DATOS DE MANERA AUTOMÁTICA:

```
2/6/22, 17:45 prueba_insertar_datos.py
1 from cassandra.cqlengine.models import Model
2 from cassandra.cqlengine import columns
3 from cassandra.cqlengine.management import sync_table
4 from cassandra.cqlengine import connection
5 import uuid
6
7 ###Definición del Modelo de datos
8 class personas(Model):
9     __table_name__ = "personas" #Cambiar nombre de la tabla
10     id = columns.UUID(primary_key=True, default=uuid.uuid4)
11     first_name = columns.Text()
12     last_name = columns.Text()
13
14
15 connection.setup(['127.0.0.1'], "prueba", protocol_version 3) #127.0.0.1
16 localhost, "prueba" - nombre keyspace
17 sync_table(personas)
18
19 person = personas.create(first_name = "Javier", last_name = "Garcia")
20 print("todo OK")
```

*Tabla generada con una fila a insertar*

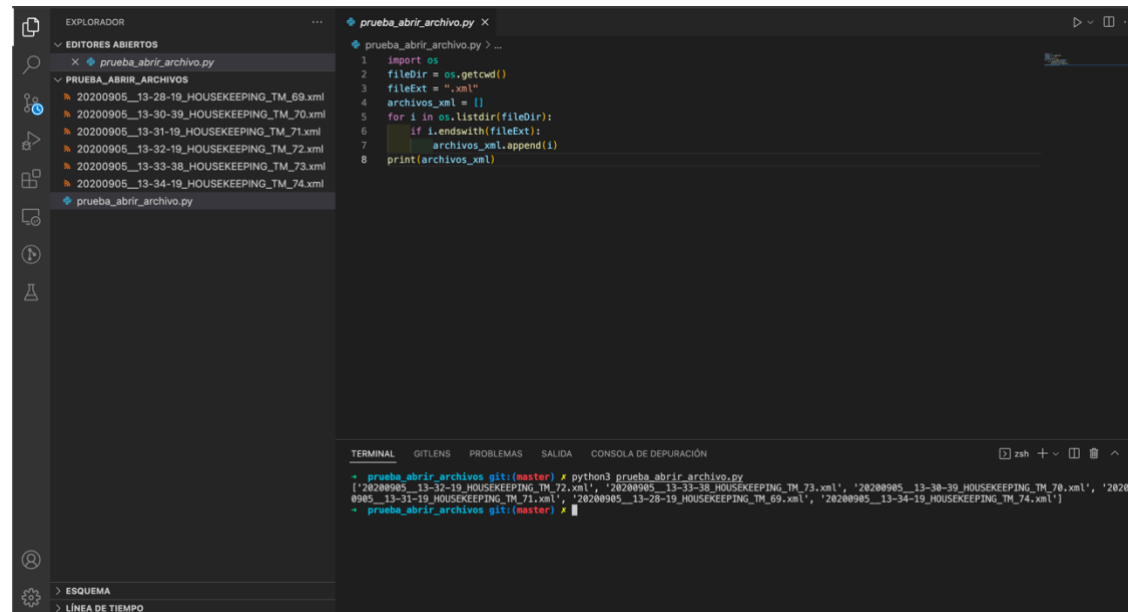


```
[cqlsh:prueba> describe tables;
housekeeping personas
[cqlsh:prueba> select * from personas;

id | first_name | last_name
---+-----+-----
f59b0de4-b4d6-4174-a4aa-129fd1e1e2f4 | Javier | García
```

*Fila de la table*

## 8.7. ANEXO 3. SELECCIÓN DE FICHEROS XML:



The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a directory named 'prueba\_abrir\_archivos' containing several XML files. The code editor displays a Python script named 'prueba\_abrir\_archivo.py' that uses the 'os' module to list files in the current directory and filter for those ending in '.xml'. The terminal shows the output of running the script, displaying a list of XML files.

```
EXPLORADOR
EDITORES ABIERTOS
prueba_abrir_archivo.py
PRUEBA_ABRIR_ARCHIVOS
20200905_13-28-19_HOUSEKEEPING_TM_69.xml
20200905_13-30-39_HOUSEKEEPING_TM_70.xml
20200905_13-31-19_HOUSEKEEPING_TM_71.xml
20200905_13-32-19_HOUSEKEEPING_TM_72.xml
20200905_13-33-38_HOUSEKEEPING_TM_73.xml
20200905_13-34-19_HOUSEKEEPING_TM_74.xml
prueba_abrir_archivo.py

prueba_abrir_archivo.py
1 import os
2 fileDir = os.getcwd()
3 fileExt = ".xml"
4 archivos_xml = []
5 for i in os.listdir(fileDir):
6     if i.endswith(fileExt):
7         archivos_xml.append(i)
8 print(archivos_xml)

TERMINAL
git:(master) / python3 prueba_abrir_archivo.py
['20200905_13-32-19_HOUSEKEEPING_TM_72.xml', '20200905_13-33-38_HOUSEKEEPING_TM_73.xml', '20200905_13-30-39_HOUSEKEEPING_TM_70.xml', '20200905_13-31-19_HOUSEKEEPING_TM_71.xml', '20200905_13-28-19_HOUSEKEEPING_TM_69.xml', '20200905_13-34-19_HOUSEKEEPING_TM_74.xml']
git:(master) /
```

*Código para recorrer los ficheros XML*

[illegible]

*Código de la herramienta*



```
cqlsh:prueba> SELECT COUNT(*) FROM housekeeping ;
```

count

6

(1 rows)

*Filas insertadas*

```
838188 | 72 | 1711 | 1710 | 1636 | 1628 | 1624 | 2524 | 8 | NONE | False | False | SAFE
1646 | True | True | True | 2496 | 2042 | 1715 | 1701 | True | 1740 | True | 1712 | True | 2296 | 807 | 2107
1647 | 1598 | 1139 | False | 2187 | 1583 | 3222 | 730 | 1091 | 1756 | True | True | 1223 | 493 | 703 | 712
1810 | 2449 | 1760 | 2915 | 1906 | 1587 | 1704 | 1694 | 1710 | 1707 | 1600 | 1696 | 1698 | False | True |
True | True | 856 | 54 | 739 | 161 | 161 | 50 | True | True | Housekeeping | False |
73 | 1705 | 1707 | 1633 | 1624 | 1627 | 2525 | 8 | NONE | False | False | SAFE
838433 | True | True | True | 2587 | 1820 | 1744 | 1701 | True | 2240 | 2497 | 1543 | 1753 | True | 2254 | 751 | 1857
1667 | True | True | 1711 | 2587 | 1820 | 1744 | 1701 | True | 2240 | 2497 | 1543 | 1753 | True | 2254 | 751 | 1857
1619 | 1502 | 1182 | False | 2186 | 1580 | 3223 | 730 | 1090 | 1839 | True | True | 1226 | 478 | 685 | 684
1807 | 2613 | 1768 | 2681 | 1915 | 1527 | 1701 | 1641 | 1708 | 1700 | 1565 | 1696 | 1694 | False | True |
True | True | 1811 | 68 | 591 | 186 | 198 | 50 | True | True | Housekeeping | False |
74 | 1712 | 1710 | 1635 | 1627 | 1624 | 2527 | 8 | NONE | False | False | SAFE
838678 | True | True | True | 2587 | 1711 | 1698 | 1624 | 2247 | 2433 | 1822 | 1764 | True | 2276 | 808 | 1637
1668 | True | True | 1711 | 2554 | 1580 | 1745 | 1702 | True | 1740 | True | 1621 | 1446 | 1619 | 785
1650 | 1599 | 1143 | False | 2184 | 1582 | 3221 | 730 | 1091 | 1756 | True | True | 1230 | 486 | 683 | 785
1810 | 2831 | 1784 | 2385 | 1907 | 1587 | 1705 | 1696 | 1714 | 1711 | 1603 | 1697 | 1698 | False | True |
True | True | 1152 | 182 | 413 | 182 | 211 | 37 | True | True | Housekeeping | False |
69 | 1708 | 1707 | 1635 | 1628 | 1623 | 2589 | 8 | NONE | False | False | SAFE
837208 | True | True | True | 1709 | 1695 | 1695 | True | 2255 | 1814 | 546 | 1689 | True | 2292 | 1458 | 2823
1709 | True | True | 1709 | 1795 | 2819 | 1685 | 1698 | True | 1742 | True | 1623 | 1444 | 618 | 676
1644 | 1598 | 1136 | False | 2188 | 1585 | 3220 | 730 | 1090 | 1754 | True | True | 1226 | 511 | 694 | 676
1812 | 1996 | 1754 | 3316 | 1946 | 1587 | 1698 | 1686 | 1784 | 1700 | 1593 | 1695 | 1698 | False | True |
True | True | 168 | 6 | 898 | 207 | 168 | 55 | True | True | Housekeeping | False |
71 | 1710 | 1709 | 1636 | 1628 | 1624 | 2528 | 8 | NONE | False | False | SAFE
837943 | True | True | True | 2324 | 2303 | 1700 | 1700 | True | 2256 | 2286 | 1834 | 1710 | True | 2296 | 922 | 2364
1650 | True | True | 1710 | 2324 | 2303 | 1700 | 1700 | True | 2256 | 2286 | 1834 | 1710 | True | 2296 | 922 | 2364
1646 | 1598 | 1139 | False | 2186 | 1583 | 3224 | 730 | 1089 | 1756 | True | True | 1223 | 490 | 701 | 464
1802 | 2293 | 1749 | 3060 | 1917 | 1597 | 1702 | 1691 | 1709 | 1705 | 1598 | 1696 | 1697 | False | True |
True | True | 673 | 44 | 805 | 180 | 148 | 54 | True | True | Housekeeping | False |
70 | 1709 | 1709 | 1636 | 1627 | 1624 | 2517 | 7 | NONE | False | False | SAFE
837698 | True | True | True | 1709 | 1695 | 1624 | 2517 | 2118 | 887 | 1710 | True | 2294 | 1086 | 2626
1680 | True | True | 1711 | 2124 | 2565 | 1705 | 1700 | True | 1741 | True | 1621 | 1444 | 1619 | 684
1646 | 1598 | 1136 | False | 2186 | 1584 | 3223 | 730 | 1091 | 1756 | True | True | 1229 | 495 | 631 | 684
1804 | 2155 | 1752 | 3166 | 1933 | 1582 | 1701 | 1690 | 1707 | 1703 | 1596 | 1695 | 1697 | False | True |
True | True | 460 | 16 | 844 | 203 | 156 | 57 | True | True | Housekeeping | False |
```

*Filas insertadas*

generacion\_automatica.py  
tm-schema.xsd

*Ficheros XML eliminados*

