Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Bases de datos para series temporales: estudio

Autor: Eduardo Garzo Jiménez

Tutor: Antonio Moratilla Ocaña

2023

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Bases de datos para series temporales: estudio

Autor: Eduardo Garzo Jiménez

Tutor: Antonio Moratilla Ocaña

TRIBUNAL:

Presidente: Dr. Eugenio Fernández Vicente

Vocal 1º: Ing. Ángel Javier Álvarez Miguel

Vocal 2º: Dr. Antonio Moratilla Ocaña

FECHA: 26/09/2023

Resumen

En este Trabajo de Fin de Grado se desarrolla un estudio teórico sobre las series temporales, los tipos que hay y componentes, así como en las bases de datos orientadas a series temporales, presentando una revisión de los sistemas de bases de datos disponibles, viendo qué características tienen, y posteriormente profundizar en los casos de InfluxDB y TimescaleDB, en este último se hará una comparativa de funcionalidades que se extienden de una base de datos relacional, como es PostgreSQL. Estos ejemplos se escogen para comparar la forma de tratar los datos de una base de datos SQL y otra NoSQL.

Palabras clave: series temporales, bases de datos, TimescaleDB, InfluxDB.

Abstract

This bachelor's thesis develops a theoretical study on time series, the types and components that exist, as well as on time series-oriented databases, presenting a review of the available database systems, examining their characteristics, and subsequently delving into the cases of InfluxDB and TimescaleDB, the latter of which will be compared in terms of functionalities that extend from a relational database, such as PostgreSQL. These examples are chosen to compare the way data is treated in an SQL database and a NoSQL one.

Keywords: time series, databases, TimescaleDB, InfluxDB.

Índice general

Índice de figuras

Capítulo 2

Capítulo 4

Índice de tablas

Capítulo 1

Introducción

1.1 Presentación

Las series temporales son conjuntos de datos representados normalmente como una secuencia de valores de una variable, que se toman a lo largo de un periodo de tiempo determinado. Actualmente se utilizan en diversos campos, como economía para predecir posibles cambios en el mercado, medicina para saber cómo evoluciona una enfermedad a lo largo del tiempo en una población y cuando se dan picos de contagio en ciertos momentos del año, o climatología para predecir si va a llover con mayor exactitud cierto día del año, en base a datos anteriores, entre otros.

Estos conjuntos de datos no son de cualquier tipo y han de ser necesariamente datos que van cambiando a lo largo del tiempo, y por ello se dispone de herramientas para guardar estos datos según ciclos de tiempo que nosotros indiquemos y así almacenarlos en las bases de datos que usaremos para después visualizar estos datos e intentar extraer conocimiento de los mismos, usando herramientas predictivas, o de visualización de patrones que puedan suceder a lo largo del tiempo.

1.2 Objetivos

En la elaboración del presente Trabajo de Fin de Grado se pretenden conseguir los siguientes objetivos:

- El primer objetivo consiste en describir el estado del arte de las bases de datos orientadas a series temporales. Para ello se busca cumplir con los siguientes puntos:
    - Entender qué son las series temporales y que componentes tienen.
    - Explicar el concepto de lo que son las bases de datos.
    - Y así entender qué son las bases de datos orientadas a series temporales.
- Como segundo objetivo se pretende conocer algunos de los principales ejemplos de sistemas de bases de datos de series temporales, y describir algunas ventajas y desventajas que tienen estas bases de datos.
- El tercer objetivo de este trabajo es hacer un estudio sobre las características y los métodos de procesamiento, acceso, modificación, almacenamiento, y otros tipos de formas de tratar los datos, dentro de estas bases de datos orientadas a series temporales, y ver las diferencias que hay con la forma de hacerlo con el modelo más común de las bases de datos, que es el relacional, para comprender mejor el papel de este tipo de bases de datos de series de tiempo.
- Y finalmente se pretende poner a prueba un caso práctico de bases de datos orientadas a series temporales, viendo cómo se puede utilizar, las formas de visualización de una serie temporal y hacer un ejercicio práctico comparativo sobre el rendimiento entre una base de datos que utiliza SQL y otra NoSQL.

1.3 Estructura de la memoria

Se hará un breve resumen de los puntos a tratar en cada capítulo de la presente memoria:

- En el Capítulo 2 se analizarán las series temporales, los tipos que encontramos y sus componentes, así como una explicación para entender qué son las bases de datos. Y por último ejemplos que encontramos de diferentes bases de datos orientadas a series temporales, un análisis de estas, así como ventajas e inconvenientes que estos sistemas presentan.
- En el Capítulo 3 se analizarán en profundidad los sistemas de bases de datos elegidos InfluxDB y TimescaleDB, haciendo una presentación sobre las características, estructura de datos y motivos por los que se escogen estas herramientas, así como una comparativa en el funcionamiento, y

formas de acceder, modificar y leer los datos. Observando la diferencia entre usar una base de datos SQL, como es TimescaleDB, y una NoSQL como InfluxDB.

- En el Capítulo 4 se explicará cómo se pueden instalar estas bases de datos en un sistema virtual, se expondrán las formas de visualización de las series temporales por parte de ambas bases de datos, y se realizará un estudio comparativo de rendimiento en diferentes operaciones de las dos bases de datos haciendo uso del mismo tipo de dataset.
- En el Capítulo 5 se presentan las conclusiones obtenidas sobre el uso de este tipo de bases de datos, que importancia tienen, así como se presenta una posible línea de investigación futura del trabajo para ampliar conocimientos, y comparar con más resultados.
- Finalmente se presenta la Bibliografía, y anexos donde se exponen los códigos utilizados para poner a prueba las técnicas de análisis.

Capítulo 2

Estado del arte

2.1 Introducción a las series temporales

Una serie temporal es una sucesión de datos observados que tienen un valor con respecto a una variable, que podemos denominar X, a lo largo de un periodo de tiempo determinado, que podemos llamar T. La forma habitual de representar las series temporales es mediante un gráfico de la serie temporal, donde:

- En el eje x del gráfico se representan los valores de tiempo, sería una variable independiente.
- Y el eje y muestra los valores de la variable observada, objeto del estudio, que es una variable dependiente.

En este tipo de gráficos, se pueden observar patrones y tendencias en los datos a lo largo del tiempo, como por ejemplo estacionalidad, tendencia o ciclos.

Una característica importante de las series temporales que las distingue de otros tipos de datos es que los datos de la serie se recopilan en intervalos de tiempo regulares, en orden cronológico. Debido a esta naturaleza secuencial que tienen las series temporales, estas deben de ser tratadas de una manera específica en su análisis, ya que los datos de las series temporales pueden estar correlacionados entre sí, pudiendo presentar patrones estacionales, tendencias de largo plazo y ruido, que se deben de tener en consideración durante el procesamiento y la modelización de estos datos. Por ejemplo, podríamos hacer un análisis de la relación entre las ventas de un producto a lo largo del tiempo, y relacionarlo con una variable externa determinada, como podría ser el clima en el que nos encontremos. Supongamos que tenemos un conjunto de datos de las ventas del producto que queremos analizar y de la temperatura media diaria a lo largo de un año. La correlación entre las dos variables podría ayudarnos a optimizar nuestras campañas, estrategias de marketing y de producción, viendo si las ventas de un producto pueden estar influenciadas por la temperatura. Si por ejemplo, estamos analizando una bebida refrescante, veremos que seguramente durante los meses con una temperatura media mayor las ventas de este producto sean mayores, de forma que gracias a este análisis de las series temporales podemos así prever una mayor demanda y aumentar el nivel de inventario para evitar la escasez de productos. En este caso, el análisis de la correlación puede ayudar a comprender el comportamiento de la serie y a tomar decisiones sobre la gestión de la cadena de suministro y el marketing.

Los objetivos que podemos conseguir con el análisis de las series temporales son:

- Identificar los patrones estacionales de una variable y las tendencias, para así comprender cómo evoluciona a lo largo del tiempo.
- Predecir el comportamiento de una variable en el futuro, pudiendo prever futuras observaciones en la serie temporal en función de los datos históricos y las tendencias que hayamos podido identificar.
- Analizar la correlación sabiendo así la relación que pueden presentar las series temporales de la variable o variables que estemos estudiando, con respecto a otras variables que podamos considerar relevantes.

2.1.1 Tipos de series temporales

Dentro del estudio que hacemos de las series temporales podemos diferenciar varios tipos, de forma que dependiendo del tipo de serie que analicemos podamos ver de manera más sencilla algunas tendencias, identificar correlación entre otras variables, o incluso podemos diferenciar en el tipo de tratamiento previo que tenemos que hacer a los datos antes de comenzar su análisis.

Los tipos de series temporales que identificamos son:

- **Series temporales univariantes:** son aquellas donde cada observación corresponde a una sola variable estadística de interés, medida en diferentes momentos a lo largo del tiempo, es decir, es una secuencia de datos donde cada punto de datos tiene un único valor y está relacionado con un momento en el tiempo. Un ejemplo de este tipo de series sería la temperatura promedio diaria.
- **Series temporales multivariantes:** son series temporales donde cada observación corresponderá a varias variables estadísticas medidas a lo largo del tiempo. Por ejemplo, la temperatura y la humedad relativa.
- **Series temporales estacionarias:** son aquellas que tienen propiedades constantes a lo largo del tiempo que estamos midiendo, tales como la media y la varianza, es decir, que la serie tiene una frecuencia constante, y no se ve influenciada por la estacionalidad, tendencias u otros patrones que cambian a lo largo del tiempo. Este tipo de series son relevantes en el análisis de datos, ya que es más sencillo predecir y modelar el comportamiento de una variable específica, aunque no todas las series temporales son estacionarias, necesitando un cierto grado de transformación para poder utilizar modelos estadísticos más usados, como es el caso de ARIMA. Por ejemplo, una serie temporal de este tipo sería una donde el número de alumnos inscritos en una escuela se mantiene constante durante varios años sin cambios significativos.
- **Series temporales no estacionarias:** en estas a diferencia de las series estacionarias, se pueden observar variabilidad y patrones, que van cambiando a lo largo del tiempo o tendencias, de forma que aquí sí que tenemos propiedades como la media y la varianza que no son constantes. Estos patrones pueden dificultar el análisis y la predicción de la serie temporal, ya que los datos no cumplen con los supuestos necesarios para aplicar algunos modelos estadísticos, como los modelos ARIMA, buscando en algunos casos transformar la serie temporal mediante técnicas de diferenciación o de descomposición para hacerla estacionaria, antes de aplicar modelos estadísticos para su análisis o predicción. Un ejemplo es una serie temporal que muestra un aumento constante en las temperaturas medias a lo largo del tiempo.
- **Series temporales con tendencia:** son aquellas muestran un patrón a lo largo del tiempo, que no se repite en un ciclo regular. Con tendencia en este caso nos referimos a un cambio de forma gradual en el valor medio de la serie que estamos observando. Esta tendencia puede ser lineal o no lineal. Por ejemplo, una serie temporal que muestra un aumento constante en las ventas de una empresa.
- **Series temporales estacionales:** son aquellas que muestran patrones repetitivos o estacionales, es decir fluctuaciones que se repiten a intervalos regulares a lo largo del tiempo, relacionándose con factores estacionales, como el mes del año, día o semana. Un ejemplo claro de este tipo de series temporales sería las ventas de una tienda de juguetes que se miden durante varios años, donde se vería que las ventas de juguetes pueden mostrar una tendencia estacional cada año con aumentos significativos en noviembre y diciembre debido a las compras navideñas.
- **Series temporales discretas:** estas series son aquellas donde las observaciones se toman en momentos discretos y específicos del tiempo, como un día, una hora o un segundo. Cada observación se refiere a una instantánea de la serie temporal en un instante de tiempo específico y no hay observaciones entre dos instantes discretos. Un ejemplo de una serie temporal discreta podría ser el número de visitantes a un sitio web por hora. En este caso, las observaciones se tomarían en momentos específicos, como a las 9 a.m., a las 10 a.m. y así sucesivamente, y el número de visitantes a la página web en cada hora se registraría como una única observación.
- **Series temporales continuas:** en estas series las observaciones se toman en un rango continuo y sin interrupciones de tiempo o en lapsos muy pequeños, a diferencia de las discretas, las series

continuas representan una medida continua y sin interrupciones de la variable a lo largo del tiempo. Un ejemplo de serie temporal continua podría ser la monitorización de una señal biológica, como la frecuencia cardíaca, en tiempo real y de manera continua. En este caso, medimos la señal en intervalos de tiempo muy pequeños, y así podemos tener una visión más detallada y precisa del comportamiento de la variable a lo largo del tiempo.

2.1.2 Componentes de una serie temporal

Cuando realizamos un análisis o procesamos los datos de las series temporales debemos tener en cuenta que estas presentan varias características, denominadas componentes. Estos componentes son diferentes patrones que se pueden descomponer para así entender la dinámica de los datos, así como poder realizar estudios específicos a cambios o tendencias que observamos de los mismos.

Podemos observar cuatro componentes principales en las series temporales:

- **Tendencia (T):** es un patrón en la serie temporal que muestra su comportamiento, movimiento o dirección a lo largo del tiempo. Este puede ser descendente, ascendente o constante..

  La importancia de estudiar la tendencia es que nos muestra la evolución subyacente en nuestra serie temporal. Como ya hemos visto en los tipos de series temporales esta tendencia puede ser influenciada por diversos factores, dependiendo de nuestro caso de estudio, como puede ser el clima.

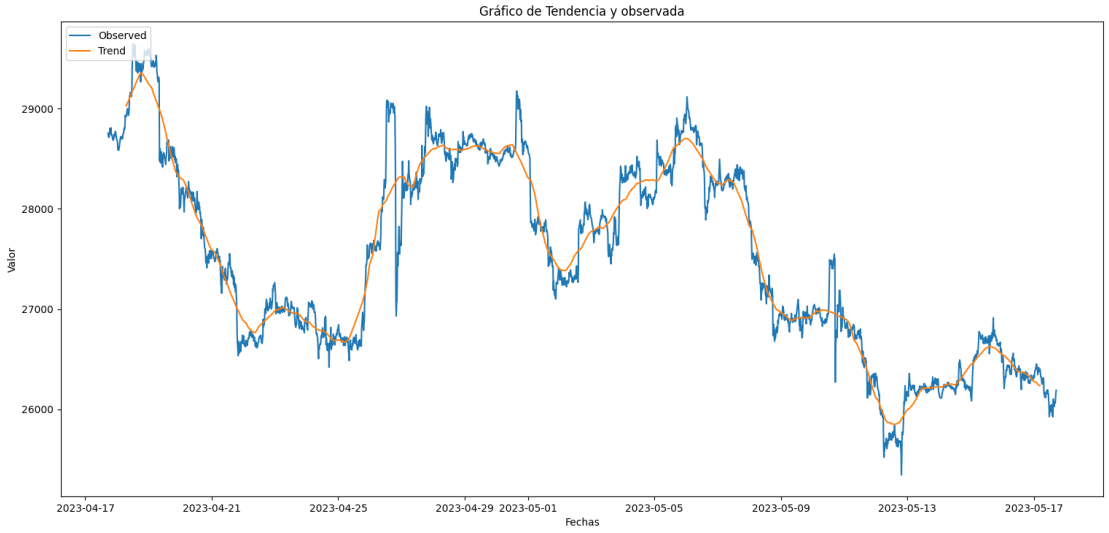

Figura 2. 1: Tendencia de una serie temporal

- **Estacionalidad o componente estacional (E):** es un patrón repetitivo de oscilaciones que se presenta en un periodo de tiempo específico. Esta estacionalidad puede ser diaria, semanal, mensual o anual. Un ejemplo del componente estacional en una serie temporal sería un registro de ventas de ropa de invierno, a lo largo del año, donde veremos que, en las estaciones más frías, habrá un mayor volumen de ventas.

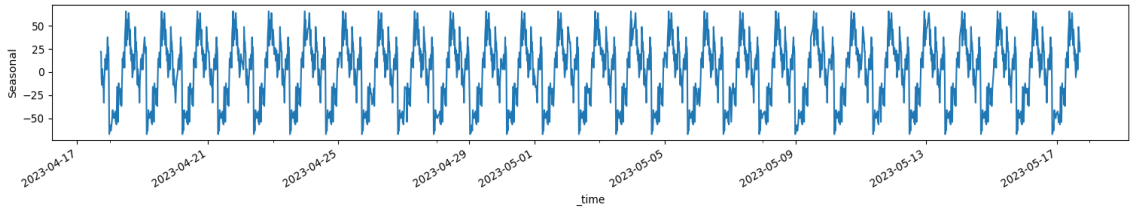

Figura 2. 2: Estacionalidad de una serie temporal

- **Ciclicidad o componente cíclico (C):** este componente se refiere a las fluctuaciones regulares que podemos observar en los datos, estas serían secuencias alternas de datos con valores inferiores o superiores a la línea de tendencia. Por ejemplo, la actividad económica puede tener ciclos de auge y caída que se dan a lo largo de varios años, y estas fluctuaciones pueden ser identificadas como componentes cíclicos de una serie temporal. La identificación del componente cíclico puede ser utilizada para hacer predicciones futuras sobre la serie temporal y para entender mejor los factores que afectan la misma.

- **Ruido o componente irregular (R):** el componente de ruido de una serie temporal se refiere a la variación aleatoria e impredecible en los datos que no están relacionados con ninguna de las componentes identificadas de la serie temporal, como la tendencia, estacionalidad o el componente cíclico. El ruido es un componente de la serie temporal que no se puede explicar y que no se puede predecir con precisión.

  El ruido puede estar presente en cualquier tipo de serie temporal, desde la temperatura del aire hasta el precio de las acciones, y puede ser causado por una variedad de factores, como los errores de medición, los eventos aleatorios que afectan la serie y otras fuentes de variación impredecible.

  Es importante entender y modelar el componente de ruido para analizar una serie temporal de manera efectiva. El análisis del ruido puede ayudar a identificar patrones que no están relacionados con la tendencia, la estacionalidad o el componente cíclico. La eliminación del ruido de una serie temporal puede mejorar la calidad de los datos y hacer que sea más fácil identificar patrones significativos y hacer predicciones precisas.

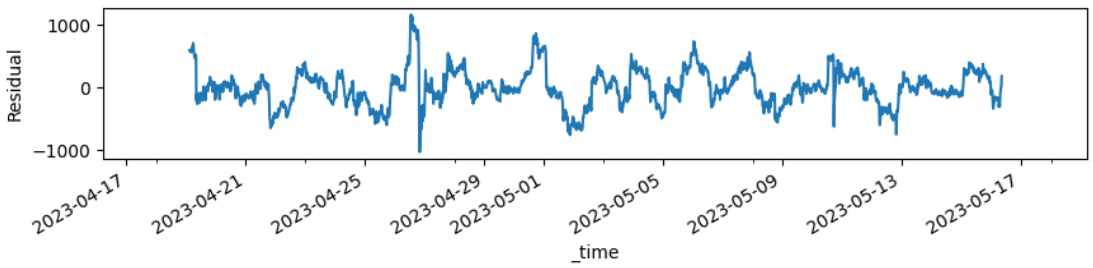

Figura 2. 3: Ruido de una serie temporal

Estos cuatro principales componentes son los que nos proporcionan las series temporales, y podemos integrarlos en esquemas distintos según el problema que estemos tratando de resolver o las características de nuestros datos, donde denotaremos Y como la serie temporal, T como la tendencia de la serie, E como su estacionalidad, C como su componente cíclico y R como el ruido que está presente, los modelos son:

- **Modelo aditivo**: en este modelo obtenemos la serie temporal como resultado de la suma de los componentes, la fórmula que representa este modelo es la siguiente:

$$Y_t = T_t + E_t + C_t + R_t$$

- **Modelo multiplicativo**: en este modelo la serie temporal se obtiene como resultado del producto de los componentes de la misma, la fórmula que representa este modelo es la siguiente:

$$Y_t = T_t \cdot E_t \cdot C_t \cdot R_t$$

La elección entre un modelo aditivo y un modelo multiplicativo dependerá del tipo de estacionalidad que se observe en los datos de la serie temporal. Si no hay cambios significativos en la amplitud de la estacionalidad, el modelo aditivo será más apropiado, es decir, cuando el patrón estacional no depende de los datos. Si la amplitud de la estacionalidad cambia a medida que la serie temporal crece, es decir, cuando el patrón de la estacionalidad en este caso sí que depende de la magnitud de los datos, se debería usar un modelo multiplicativo para descomponer la serie temporal en sus componentes subyacentes.

2.1.3 Ventajas y desventajas del análisis de series temporales

- Ventajas

El análisis de series temporales nos permite prepararnos según nuestros datos pasados, y ver eventos importantes que en un primer momento pasan desapercibido, de forma que el analizar las series temporales nos ofrece ciertas ventajas que debemos tener en cuenta según los estudios que estemos haciendo, entre ellas:

- Pronóstico: El análisis de series temporales permite a los profesionales pronosticar valores futuros. A través de modelos estadísticos, es posible prever cómo cambiarán los datos en el futuro. Esto ayuda en la planificación de estrategias en los negocios, la elaboración de presupuestos, la gestión de recursos, entre otros.
- Detección de patrones y tendencias: Al estudiar las series temporales a lo largo del tiempo, se pueden detectar patrones y tendencias en los datos que no se perciben al observar los valores aislados. La detección de tendencias proporciona una comprensión más clara de la evolución histórica de los datos y permite predecir su comportamiento futuro.
- Identificación de valores atípicos: Los valores atípicos pueden ser indicativos de problemas o anomalías en los datos. El análisis de series temporales permite detectar valores atípicos de manera temprana para poder investigar las causas subyacentes y tomar medidas correctivas cuando sea necesario.
- Identificación de relaciones causales: El análisis de series temporales permite la identificación de relaciones causales entre diferentes variables. Al entender las relaciones entre las variables, es posible tomar medidas diseñadas específicamente para mejorar el desempeño del sistema.
- Mejora en la calidad del pronóstico: Los modelos de series temporales permiten una mejora en la calidad del pronóstico en comparación con otros modelos. Al ser capaces de detectar patrones y tendencias a lo largo del tiempo, los modelos de series temporales pueden generar predicciones más precisas y confiables.

- Desventajas

También debemos mencionar algunas de las limitaciones que presentan estos análisis, entre ellas:

- Riesgo de sobreajuste: Al igual que con otros métodos de modelado estadístico, existe el riesgo de ajustar demasiado los datos para un modelo de series temporales. Si un modelo se ajusta demasiado

a los datos históricos, esto puede resultar en una falta de capacidad para predecir futuros valores de la serie temporal.

- Dependencia de la calidad de los datos: El análisis de series temporales depende de la calidad de los datos disponibles. Si los datos están incompletos o se han registrado de manera inconsistente, es posible que los modelos de series temporales no proporcionen resultados precisos. Además, los datos irrelevantes o ruidosos pueden afectar la calidad de los modelos y reducir la precisión de las predicciones.
- Dificultad en la identificación de variables importantes: En algunas series temporales, es difícil identificar qué variables son importantes. Un modelo puede incluir variables que no aportan información relevante, lo que da como resultado modelos demasiado complejos que son difíciles de interpretar y de utilizar en la toma de decisiones.
- Limitaciones para predecir eventos inesperados: Un modelo de series temporales se basa en datos históricos y patrones, lo que puede limitar su capacidad para predecir eventos inesperados o al azar, como una pandemia o desastres naturales. Dado que estos eventos no se han producido en el pasado, es difícil prever su impacto en la serie temporal.

Es importante tener en cuenta estas limitaciones y utilizar el análisis de series temporales de manera cuidadosa y adecuada al conjunto de datos.

2.2 Evolución en el tiempo de las bases de datos

Las bases de datos surgieron con la necesidad de almacenar grandes cantidades de información en sistemas informáticos, y han evolucionado significativamente a lo largo de la historia.

Desde sus inicios en los años 60, las bases de datos se han transformado desde simples sistemas de archivos hasta grandes sistemas de gestión de bases de datos. En esta década, se desarrollaron los primeros sistemas de bases de datos jerárquicos, que utilizaban una estructura de árbol para organizar la información en varios niveles: el nivel superior contenía la información más general, y cada nivel inferior se ramificaba en subniveles de información cada vez más específicos.

En la década de los 70, los sistemas de bases de datos en red surgieron como alternativa más flexible a los modelos jerárquicos. Estos sistemas permitían relaciones más complejas entre los datos, lo que permitió una mayor organización y clasificación de la información. Los Sistemas de Gestión de Bases de Datos (SGBD) jerárquicos y de red fueron la primera generación de SGBD. Sin embargo, presentaron algunos inconvenientes, como la necesidad de escribir programas complejos para responder a cualquier tipo de consulta de datos, la mínima independencia de datos y la falta de un fundamento teórico.

En 1970, Edgar Frank Codd de los laboratorios de investigación de IBM, escribió un artículo presentando el modelo relacional. En este artículo presentaba también los inconvenientes de los sistemas previos, el jerárquico y el de red [1]. Como resultado de la evolución de los sistemas de bases de datos, surgieron dos grandes avances importantes. El primero fue la creación del lenguaje estructurado SQL, que se ha convertido en el estándar para los sistemas relacionales. El segundo, que ya en los años ochenta se empezaron a desarrollar los primeros sistemas relacionales, surgiendo así los primeros modelos de bases de datos relacionales, que permitían que los datos se guardarán en tablas y que se pueden relacionar entre sí mediante una clave común.

Durante la década de los 80, los sistemas de gestión de bases de datos relacionales se volvieron cada vez más avanzados y eficientes en la manipulación y organización de la información, empezaron a ganar adeptos, convirtiéndose en la tecnología líder para manejar grandes volúmenes de datos. Surgieron nuevos sistemas de bases de datos más robustos y eficientes en la recuperación de la información. En esa época también surgieron los sistemas de bases de datos objeto-relacionales, que permitían tratar objetos complejos como instancias de una clase, proporcionando una mayor flexibilidad en la gestión de datos.

Durante los años 90, los sistemas de bases de datos relacionales se volvieron aún más sofisticados, agregando nuevas características y mejorando la funcionalidad, como la adición de interfaces de usuario gráficas, el soporte para el almacenamiento y manipulación de datos multimedia.

Surgiendo a finales de esta década una nueva clase de sistemas de bases de datos conocida como bases de datos NoSQL (no solo SQL). Estos sistemas fueron creados para abordar los desafíos que presentaban los

grandes volúmenes de datos no estructurados, como los que se encuentran en redes sociales y aplicaciones web. En lugar de utilizar un modelo de tabla relacional, las bases de datos NoSQL utilizan modelos de datos más flexibles y escalables. Este enfoque se convirtió en la base de los modernos sistemas de almacenamiento y procesamiento de big data.

A medida que surgieron nuevas tecnologías, las bases de datos también evolucionaron para adaptarse a ellas. En la actualidad, la disponibilidad de bases de datos se ha incrementado exponencialmente gracias a la popularidad de Internet. Su evolución actualmente se centra en la nube, la inteligencia artificial y el aprendizaje automático.

Con la creciente popularidad de los servicios en la nube, surgen las bases de datos en la nube,lo cual permite a las empresas almacenar grandes cantidades de datos en servidores remotos y acceder a ellos desde cualquier lugar del mundo. Además, el aumento de la inteligencia artificial y el aprendizaje automático lleva al desarrollo de bases de datos cognitivas, que utilizan tecnologías avanzadas para analizar datos y extraer información útil. Llevándonos a manejar grandes cantidades de datos en tiempo real y teniendo en las bases de datos una interfaz más intuitiva y sencilla para los usuarios.

Conforme va pasando el tiempo se generan cada vez más datos, de más tipos y se obtienen de más fuentes, de manera que se necesitan gestionar grandes volúmenes de datos relacionados con el tiempo, lo que llevó a la creciente popularidad de las bases de datos de series temporales. Estos sistemas están diseñados para manejar grandes volúmenes de datos relacionados con el tiempo, incluyendo series de tiempo financieras, sensores de IoT y registros de accesos a aplicaciones web.

Las bases de datos de series temporales son optimizadas para consultas y análisis de tiempo y proporcionan una gestión avanzada de datos de series temporales, con operaciones de datos que incluyen agregación, interpolación y remuestreo. Además, estos sistemas están optimizados para el procesamiento de datos en tiempo real, lo que los hace ideales para aplicaciones como monitoreo en línea y análisis predictivo.

2.3 Introducción a las bases de datos

La gestión de información es una parte esencial de cualquier estudio, análisis u organización empresarial, y las bases de datos se han convertido en una herramienta fundamental para ello.

Una base de datos es un conjunto de datos almacenados en memoria, que se organizan en una estructura en serie, de tablas, relaciones y consultas, lo que permite almacenar y recuperar información de manera eficiente.

Así como guardamos los datos en sistema centralizado de colección de los mismos, que es la base de datos, también es conveniente disponer de un entorno para poder gestionar las interacciones a estos datos, este entorno es lo que denominamos Sistema Gestor de Bases de Datos (SGBD) [2].

El modelo de datos de una base de datos de series temporales se enfoca en el tiempo y cómo los datos cambian a lo largo del mismo. Este modelo se utiliza para almacenar y analizar datos que cambian con el tiempo, como los datos meteorológicos, los datos financieros o las mediciones médicas. En un modelo de datos de series temporales, los datos se organizan en tablas, donde cada fila representa un punto de tiempo, y cada columna representa un valor para una variable específica. También las tablas en un modelo de series temporales también incluyen un campo de fecha o tiempo para indicar cuándo se registró cada valor.

Además, los modelos de series temporales también incluyen técnicas de análisis y pronóstico para predecir valores futuros en función de datos históricos. Esto implica el uso de algoritmos estadísticos y modelos matemáticos para entender la estructura de los datos y las correlaciones entre los valores a lo largo del tiempo.

2.3.1 Sistema Gestor de Bases de Datos (SGBD)

Un sistema gestor de bases de datos (SGBD) o DataBase Management System (DBMS) en inglés, es un software que permite la gestión y el acceso a una base de datos. Este sistema proporciona herramientas para la creación, almacenamiento, organización, gestión y recuperación de información almacenada en una base

de datos. El objetivo principal de un SGBD es facilitar el acceso a la información de manera rápida, eficiente y segura.

Un SGBD se encarga de manejar, administrar y garantizar la integridad y confidencialidad de los datos almacenados, permitiendo a los usuarios y aplicaciones trabajar con los datos de manera sencilla y eficiente. Esto se logra mediante la implementación de diferentes funcionalidades y características, como la capacidad de consultar y actualizar datos, la gestión de transacciones para garantizar la consistencia de los mismos, la creación y modificación de estructuras de la base de datos, la indexación de los datos para mejorar el rendimiento de las consultas, entre otros.

Hay una gran variedad de modelos de bases de datos, como jerárquicos, relacionales o no relacionales, orientadas a objetos, etc. En nuestro caso las bases de datos orientadas a series temporales son un modelo único entre las mismas, ya que son bases de datos optimizadas para almacenar y analizar datos que se recopilan y registran en intervalos de tiempo regulares.

2.3.2 Sistema de Base de Datos

Ahora bien, al conjunto de los conceptos de base de datos, SGBD y un modelo de datos, es lo que se denomina sistema de base de datos. El sistema de base de datos es un conjunto de componentes que interactúan entre sí para almacenar, recuperar y gestionar grandes cantidades de datos. Los componentes básicos de un sistema de base de datos son los siguientes:

- Datos: Los datos en sí son el componente más importante de un sistema de bases de datos. Estos pueden incluir cualquier tipo de información que se desee almacenar, como nombres, direcciones, fechas, imágenes y más.
- Software: es el programa que gestiona la base de datos y permite que los usuarios la utilicen. Puede haber diferentes tipos de software de base de datos, como sistemas de gestión de bases de datos relacionales y NoSQL.
- Hardware: es la infraestructura física necesaria para ejecutar el sistema de base de datos. Esto incluye servidores, discos duros, redes y otros equipos.
- Lenguajes de acceso: son los lenguajes que utilizan los usuarios para interactuar con la base de datos y realizar consultas, como SQL o Flux.
- Usuarios: son las personas que utilizan el sistema de base de datos para buscar, crear, modificar y eliminar datos. Los usuarios pueden acceder a la base de datos a través de aplicaciones, interfaces web u otros medios.
- Procedimientos o reglas: son las políticas que regulan cómo se deben utilizar y gestionar los datos en la base de datos, y cómo los usuarios deben interactuar con ella.
- SGDB: Es el Sistema Gestor de Bases de Datos, el cual proporciona una interfaz entre la base de datos, el software y los usuarios.

Un sistema de base de datos es una herramienta fundamental para la gestión de grandes volúmenes de información, y su éxito depende de una combinación efectiva de hardware, software, usuarios y datos, todos coordinados a través del Sistema Gestor de Bases de Datos (SGBD).

2.4 Bases de datos orientadas a series temporales

Ya sabemos que son las series temporales y que son las bases de datos, ahora la pregunta es, ¿qué son exactamente las bases de datos orientadas a series temporales?

Una base de datos orientada a series temporales es un sistema de software diseñado para almacenar y gestionar datos de estas series temporales. Los datos de series temporales son datos que se indexan a lo largo del tiempo, normalmente con un intervalo de tiempo regular entre puntos de datos.

Las bases de datos de series temporales están optimizadas para manejar grandes cantidades de datos de series temporales, lo que las hace útiles en aplicaciones en las que es necesario analizar o procesar grandes cantidades de datos en tiempo real. Una de las características clave de las bases de datos de series temporales es su capacidad para almacenar y recuperar rápidamente grandes volúmenes de datos con una sobrecarga de rendimiento mínima. Estas bases de datos suelen utilizar un formato de almacenamiento de datos diferente al de las bases de datos relacionales tradicionales, lo que les permite lograr un mejor rendimiento para los datos de series temporales.

Además de sus beneficios de rendimiento, las bases de datos de series temporales a menudo brindan una funcionalidad especializada para el análisis y la visualización de datos, como la capacidad de agregar y resumir datos en diferentes intervalos de tiempo. Esto los convierte en opciones populares para aplicaciones como IoT, análisis financiero e investigación científica, donde es necesario analizar e interpretar grandes cantidades de datos de series temporales.

2.4.1 Sistemas de bases de datos de series temporales

En este apartado se analizarán los distintos sistemas de bases de datos que gestionan series temporales más destacadas en la actualidad. Dentro de cada base de datos a analizar se verán algunas de las características más relevantes o que las diferencian, y también se analizarán algunos puntos comunes de aspectos técnicos que tener en cuenta a la hora de analizar cualquier base de datos. Por ello primero se definirán estos aspectos a analizar:

- Tipos de datos: se refiere a los diferentes tipos de datos que se pueden almacenar en la base de datos y pueden ser divididos en dos categorías principales: tipos de datos numéricos y tipos de datos no numéricos o alfanuméricos. Los tipos de datos numéricos incluyen enteros, números de coma flotante y números decimales, mientras que los tipos de datos no numéricos incluyen texto, fechas, horas, imágenes, videos y archivos de audio.

- Particionamiento: el particionamiento de una base de datos es un proceso mediante el cual una tabla se divide en varias partes o particiones, lo que permite la administración más eficiente de grandes cantidades de datos. Las particiones pueden ser definidas de diversas maneras, como por rango, hash, lista o combinación de ellas. El particionamiento también puede utilizarse para distribuir los datos de una tabla en diferentes discos o servidores, lo que mejora la escalabilidad y la disponibilidad del sistema. Además, el particionamiento puede aumentar la velocidad de los procesos de respaldo y recuperación, ya que permite realizar copias de seguridad por particiones específicas en lugar de por la tabla completa. Es importante tener en cuenta que el particionamiento puede afectar la complejidad del diseño del esquema de base de datos y el rendimiento de las operaciones de inserción y actualización.

- Replicación: la replicación es una técnica que se utiliza para crear copias de la base de datos en varios servidores, lo que ayuda a mejorar la disponibilidad y la redundancia de la base de datos. La replicación de bases de datos es un proceso mediante el cual se copian datos de una base de datos a otra. Esto permite tener múltiples copias de los mismos datos en diferentes ubicaciones, lo que puede mejorar la disponibilidad, el rendimiento y la seguridad de los datos. En general, la replicación se realiza mediante la creación de réplicas de la base de datos en diferentes servidores o ubicaciones. Las réplicas se mantienen sincronizadas mediante el intercambio de datos entre ellas. Cuando se produce una actualización en una réplica, esta se propaga a las demás réplicas para garantizar la consistencia de los datos.

- Transacciones: las transacciones en una base de datos se refieren a una serie de operaciones que se realizan como una sola unidad lógica. Una transacción es una secuencia de comandos de bases de datos que se ejecutan como una sola unidad y que se tratan como una única entidad. Las transacciones se utilizan para garantizar que las operaciones de la base de datos se realicen correctamente y que los datos sean precisos y consistentes. Las transacciones se usan comúnmente para garantizar la integridad y consistencia de la base de datos. Por ejemplo, si una transacción implica la actualización de varias tablas de la base de datos, todas las actualizaciones deben

realizarse o deshacerse por completo. Si se realiza una parte de la actualización y la transacción falla, la base de datos está en un estado inestable y los datos pueden estar corrompidos.

- APIs, o Interfaces de Programación de Aplicaciones, protocolos y herramientas que permiten que diferentes aplicaciones se comuniquen entre sí. Los métodos de acceso son las formas en que los desarrolladores pueden acceder a estas APIs para enviar y recibir información.
- Si soporta XML, ya que las APIs que soportan datos en formato XML permiten que las aplicaciones se comuniquen con los servicios web u otras aplicaciones que utilizan XML.
- Los sistemas operativos en los que se pueden desplegar estos sistemas y así poder utilizar sus APIs.
- Los lenguajes de programación que integra y permiten a los desarrolladores crear aplicaciones y scripts que pueden ser utilizados por otros desarrolladores.
- Si soporta el lenguaje SQL permitiendo que las aplicaciones realicen consultas y actualizaciones en una base de datos mediante el lenguaje de consultas SQL.
- Si soporta concurrencia de manera que varias aplicaciones accedan y actualicen simultáneamente la misma base de datos, de forma segura y controlada.
- Si los datos son consistentes, es decir, los datos están actualizados y son precisos.

Rank			DBMS	Database Model	Score		
May 2023	Apr 2023	May 2022			May 2023	Apr 2023	May 2022
1.	1.	1.	InfluxDB	Time Series, Multi-model	29.90	+1.31	+0.35
2.	2.	2.	Kdb	Time Series, Multi-model	8.03	-0.44	-0.95
3.	3.	3.	Prometheus	Time Series	7.43	+0.44	+1.29
4.	4.	4.	Graphite	Time Series	6.26	-0.05	+0.80
5.	5.	5.	TimescaleDB	Time Series, Multi-model	4.73	+0.36	+0.03
6.	6.	↑ 7.	RRDtool	Time Series	3.61	+0.42	+1.11
7.	↑ 8.	↑ 9.	DolphinDB	Time Series, Multi-model	3.42	+0.81	+1.77
8.	↓ 7.	↓ 6.	Apache Druid	Multi-model	3.07	+0.31	+0.06
9.	9.	↑ 15.	TDengine	Time Series, Multi-model	2.95	+0.34	+2.04
10.	↑ 11.	↓ 8.	OpenTSDB	Time Series	2.50	+0.32	+0.67
11.	↑ 12.	11.	GridDB	Time Series, Multi-model	2.37	+0.35	+1.14
12.	↓ 10.	12.	QuestDB	Time Series, Multi-model	2.24	0.00	+1.05
13.	13.	↓ 10.	Fauna	Multi-model	1.88	+0.14	+0.52
14.	↑ 15.	↓ 13.	Amazon Timestream	Time Series	1.27	+0.23	+0.30
15.	↓ 14.	↑ 18.	VictoriaMetrics	Time Series	1.21	+0.10	+0.63

Figura 2. 4: Tabla ranking de bases de datos de series temporales

Las bases de datos que se van a comparar en este capítulo serán las cinco primeras del ranking de bases de datos:

1. INFLUXDB
2. KDB
3. PROMETHEUS
4. GRAPHITE
5. TIMESCALEDB

Desde la página Rankind databases []se puede ver el ranking de las bases de datos de series temporales más populares del último mes. Esta página se actualiza mensualmente. Para el cálculo de las puntuaciones en el ranking se basan en las búsquedas del sistema de base de datos en sitios web, el número de menciones en redes sociales, y el número de personas que tienen ese tipo de sistema en su perfil profesional.

Para ver con más facilidad la comparativa de las características de estas bases de datos, se ha realizado la siguiente tabla:

BASES DE DATOS					
CARACTERISTICAS	INFLUXDB	KDB	PROMETHEUS	GRAPHITE	TIMESCALEDB
TIPOS DE DATOS	Datos numéricos y cadenas de caracteres	Datos numéricos y cadenas de caracteres	Sólo datos numéricos	Sólo datos numéricos	Datos numéricos, cadenas, booleanos, matrices, Blobs JSON, dimensiones geoespaciales, monedas, datos binarios, otros tipos de datos complejos
PARTICIONAMIENTO	SI	SI	SI	NO	SI
REPLICACION	Maestro-Esclavo asimétrica o simétrica	Sí, replicación simétrica o asimétrica	Federación asimétrica o simétrica	NO	Líder-Seguidor simétrica o asimétrica
TRANSACCIONES	NO	NO	NO	NO	ACID
APIS Y METODOS DE ACCESO	HTTP API JSON over UDP	HTTP API JDBC Jupyter Kafka ODBC WebSocket	RESTful HTTP/JSON API	HTTP API Sockets	ADO.NET JDBC librería C nativa ODBC API de transmisión para objetos grandes
SOPORTA DATOS EN FORMATO XML	NO	SI	NO	NO	SI
SISTEMAS OPERATIVOS	Linux OS X	Linux OS X Solaris Windows	Linux Windows	Linux Unix	Linux OS X Windows

LENGUAJES DE PROGRAMACION	.Net Clojure Erlang Go Haskell Java JavaScript(Node.js) Lisp Perl PHP Python R Ruby Rust Scala	C C# C++ Go J Java JavaScript Lua MatLab Perl PHP Python R Scala	.Net C++ Go Haskell Java JavaScript (Node.js) Python Ruby	JavaScript (Node.js) Python	.Net C C++ Delphi Java info JavaScript Perl PHP Python R Ruby Scheme Tcl
SOPORTA SQL	Lenguaje de consulta similar a SQL	Lenguaje de consulta similar a SQL	NO	NO	SI
SOPORTA CONCURRENCIA	Sí, soporta escrituras y consultas concurrentes	Sí, soporta escrituras y consultas concurrentes	Sí, soporta escrituras y consultas concurrentes	NO	Sí, soporta escrituras y consultas concurrentes
DATOS CONSISTENTES	SI	SI	SI	SI	SI

Tabla 2. 1: Tabla comparativa características BBDD

- **InfluxDB:** es una base de datos de series temporales de código abierto, diseñada específicamente para almacenar y consultar grandes cantidades de datos de series temporales en tiempo real. Algunos puntos para destacar de esta base de datos son:
    - Su alta velocidad y rendimiento: ya que está diseñado para manejar grandes volúmenes de datos de series de tiempo y puede realizar consultas y escrituras en ellos a alta velocidad.
    - Tiene un almacenamiento optimizado: los datos de InfluxDB se almacenan de forma optimizada, aumentando así el almacenamiento real del que dispongamos para guardar nuestros registros.
    - Escalabilidad horizontal: InfluxDB se puede escalar horizontalmente para manejar grandes cantidades de datos de series temporales.
    - API RESTful: InfluxDB se comunica con una API RESTful simple, lo que lo hace muy flexible y altamente compatible con otros sistemas, ya que es una interfaz en la que dos sistemas se pueden comunicar entre sí para intercambiar información de forma segura a través de la red.

  En cuanto a los entornos de despliegue, InfluxDB puede ser desplegado en diferentes entornos, como sistemas locales y entornos en la nube como Amazon EC2, Google Cloud Platform y Microsoft Azure y también se puede desplegar como una aplicación de contenedor utilizando Docker.
- **Kdb**: es una tecnología de base de datos en memoria potente y rápida diseñada para manejar grandes cantidades de datos de series temporales. Algunas características clave de Kdb incluyen:
    - Almacenamiento orientado a columnas: Kdb almacena datos en columnas en lugar de filas, lo que permite consultas más rápidas de conjuntos de datos grandes.
    - Lenguaje de programación integrado: Kdb incluye un lenguaje de programación integrado Q, que sirve como lenguaje de consulta más sencillo, optimizado para el procesamiento de grandes cantidades de datos.
    - Flexibilidad: Kdb+ admite todos los principales sistemas operativos, incluidos Linux, Windows y macOS, y se puede utilizar con una variedad de lenguajes de programación como C++, Python y Java.

- **Prometheus:** es una plataforma de monitoreo y alerta de sistemas de código abierto ampliamente utilizada. Algunas de sus principales características incluyen:
    - Modelo de datos multidimensional: Prometheus utiliza un modelo de datos multidimensional que permite etiquetar y estructurar los datos de manera efectiva.
    - Lenguaje de consulta flexible: Prometheus incluye una sintaxis de consulta flexible que facilita la exploración y análisis exhaustivo de los datos.

- **Graphite:** es una base de datos y una herramienta de visualización de series temporales, de código abierto, destaca en su:
    - Entrada flexible de datos: Graphite puede recibir datos desde una gran variedad de fuentes, incluyendo StatsD, collectd y scripts personalizados.
    - Visualización: Graphite posee una aplicación web integrada que permite a los usuarios visualizar datos de series de tiempo a través de gráficos interactivos y paneles de control.

  Graphite se puede desplegar en entornos de la nube, al igual que InfluxDB, y también como una máquina virtual o una aplicación contenerizada. Debido a que Graphite es de código abierto, también se puede personalizar e integrar en aplicaciones personalizadas.

- **TimescaleDB**: base de datos de series temporales, también de código abierto, escrita en lenguaje C y se distribuye como una extensión de la base de datos relacional PostgreSQL, donde destaca en:
    - Su compatibilidad con SQL: TimescaleDB soporta comandos SQL completos y cuenta con compatibilidad con PostgreSQL para facilitar su integración con otras aplicaciones.
    - Análisis avanzado de datos: cuenta con funciones de agregación avanzadas y algoritmos de análisis de series de tiempo para realizar análisis de datos complejos.
    - Integración sencilla: pudiendo integrarse fácilmente con otras herramientas y sistemas como Grafana, Python y Prometheus.

De estas bases de datos se han escogido InfluxDB y TimescaleDB, para realizar un estudio en mayor profundidad, para ver las diferencias entre una base de datos temporal completamente SQL como TimescaleDB, e InfluxDB, que es la que tiene mayor puntuación del ranking, y su lenguaje no es un SQL convencional.

2.5 Ventajas e inconvenientes

En este apartado se analizarán las ventajas e inconvenientes más relevantes que presentan las bases de datos orientadas a series temporales.

2.5.1 Ventajas de las bases de datos de series temporales

Las bases de datos de series temporales tienen varias ventajas utilizadas como herramientas de trabajo, predicción o análisis, entre ellas:

- Monitoreo de alertas: Las bases de datos de series de tiempo permiten el monitoreo de alertas en tiempo real para detectar inconvenientes en el sistema reduciendo el tiempo de inactividad y previniendo pérdidas.
- Almacenamiento y archivo fácil: Las bases de datos de series de tiempo funcionan como una herramienta de gestión de datos muy eficiente para almacenar y archivar grandes cantidades de datos que cambian con el tiempo. Los datos pueden ser escritos y actualizados fácilmente, permitiendo una disponibilidad continua de los datos.
- Predicción de fallos: La capacidad de predecir un fallo en una serie de tiempo puede ser crucial en áreas como la construcción, la aeronáutica, y la industria automovilística. El análisis de datos de series de tiempo también se puede utilizar para predecir los fallos en equipos y maquinarias industriales.
- Análisis de tendencias: Los datos de series de tiempo capturan los cambios de una variable a lo largo del tiempo, lo que permite a los usuarios examinar las tendencias en los datos y detectar patrones. Estos patrones pueden ayudar a entender mejor la situación y así preverlos, permitiendo una mejor toma de decisiones.
- Análisis de la eficiencia: Las series de tiempo también se pueden utilizar para hacer un seguimiento del rendimiento de las operaciones y equipos a lo largo del tiempo y usar los datos para identificar y solucionar cualquier problema de eficiencia.

Las bases de datos de series de tiempo son una herramienta útil para almacenar y analizar datos de problemas que cambian con el tiempo. Al permitir una mejor visualización y análisis de los datos, las bases de datos de series de tiempo pueden mejorar la toma de decisiones en una variedad de áreas, desde la fabricación a la medicina.

2.5.2 Desventajas de las bases de datos de series temporales

Aunque hemos visto que las bases de datos de series temporales pueden llegar a ser muy relevantes en diversos ámbitos que nos rodean y tienen numerosos usos, estas tampoco son perfectas y si bien varios de los inconvenientes encontrados están relacionados con problemas de las tecnologías actuales, debido a limitaciones físicas, o virtuales se destacan las siguientes desventajas:

- Complejidad: Las bases de datos de series de tiempo pueden ser más complejas y difíciles de manejar que otras formas de almacenamiento de datos. Esto se debe a que los datos se organizan por fechas y tiempos, lo que requiere cierto nivel de conocimiento técnico y habilidad para trabajar con ellos.
- Menor flexibilidad: En algunas bases de datos de series de tiempo, puede requerirse una estructura de datos determinada que debe ser seguida rigurosamente. Esto significa que puede haber menos flexibilidad en la forma en que los datos se organizan y almacenan.
- Limitaciones en el tamaño de la base de datos: Las series de tiempo pueden generar una gran cantidad de datos en un corto período de tiempo, por lo que existe el riesgo de que la base de datos pueda

crecer demasiado rápido y superar sus límites. Esto puede llevar a problemas de rendimiento y la necesidad de actualizar o migrar con frecuencia a una estructura más grande.

- Problemas de redundancia: Dado que los datos se registran y guardan como puntos en el tiempo, puede ser difícil para la base de datos manejar datos que se repiten con frecuencia. Esto puede crear problemas de redundancia y hacer que la base de datos crezca más de lo necesario.
- Problemas de calidad de datos: Al igual que con cualquier otra base de datos, las series de tiempo también están sujetas a problemas de calidad de datos. Esto puede incluir datos incorrectos, faltantes o duplicados, lo que puede dificultar el análisis y la forma de obtener información precisa.

Sin embargo, si se utilizan de manera adecuada, estas desventajas pueden ser minimizadas y se puede lograr un buen manejo de datos.

Capítulo 3

Análisis de los sistemas

3.1 Introducción

En este capítulo se realizará un análisis en profundidad sobre los sistemas InfluxDB y TimescaleDB. Se tratará de exponer las diferencias claves entre las bases de datos de series temporales y las bases de datos relacionales, cogiendo como ejemplo de base de datos relacional PostgreSQL. Puesto que, de distinto modo a las bases de datos relacionales, que se organizan mediante tablas, las bases de datos de series temporales se organizan en torno a la información temporal, como el tiempo y los eventos asociados con él. Permitiendo un acceso más rápido para el análisis de este tipo de datos. En este apartado, se discutirán las características técnicas fundamentales de ambas bases de datos y se compararán en términos de rendimiento, escalabilidad y capacidad para manejar grandes conjuntos de datos. Al final, se espera tener una comprensión sólida de las diferencias clave entre estos dos tipos de bases de datos y ser capaces de seleccionar la opción más adecuada para una aplicación determinada. Hay varios aspectos de estos sistemas en los que se profundizará para ver como tratan los datos y las diferencias que tienen:

- La estructura de una base de datos se refiere a cómo se organizan los datos en la misma. En términos más simples, se trata de la forma en que los datos se almacenan en la base de datos y la relación que existe entre ellos. En general, la estructura de una base de datos consiste en tablas que contienen información específica, donde cada tabla está compuesta por columnas que representan los campos de la información, y filas que representan los registros. Para analizar la estructura de una base de datos, es necesario comprender la relación entre las diferentes tablas y cómo se conectan entre sí. Esto se puede hacer mediante la identificación de las claves primarias y foráneas que existen en cada una de las tablas. La clave primaria es el campo que identifica de forma única cada registro en una tabla, mientras que la clave foránea es el campo que establece la relación entre dos o más tablas. Una vez que se han identificado las claves primarias y foráneas, se puede analizar en detalle la estructura de la base de datos. Esto puede incluir la identificación de tablas redundantes o mal diseñadas, la optimización de la estructura para mejorar el rendimiento de la base de datos, y la creación de nuevas tablas o diagramas de relaciones que mejoren la eficiencia de la base de datos.

- Índices: se refiere a la manera en que se organizan los datos dentro de la base de datos para permitir un acceso más rápido y eficiente. Los índices de una base de datos son estructuras que se utilizan para identificar y acelerar el acceso a los datos. Estos índices se crean en una o varias columnas de una tabla y proporcionan una manera rápida de buscar datos de una tabla determinada. Los índices pueden ser creados en una o varias columnas de una tabla, dependiendo del método de indexación y los requerimientos del usuario. También es importante tener en cuenta que la creación de índices puede afectar negativamente a la velocidad de inserción, actualización y eliminación de datos en una tabla, ya que cada índice debe actualizarse cada vez que se inserta, actualiza o elimina un registro. Por lo tanto, es importante encontrar el equilibrio adecuado entre el número de índices y la velocidad y capacidad de almacenamiento de la base de datos. Los índices también se pueden utilizar para mejorar el rendimiento de las consultas complejas. Por ejemplo, si una consulta requiere la unión de varias tablas, se pueden crear índices en las columnas que se utilizarán en la unión para acelerar el proceso.

- Modelado de datos: es el proceso de definir la estructura lógica de los datos. Esto implica identificar las entidades (objetos o conceptos de la vida real que se almacenan en la base de datos), las relaciones entre ellas y las reglas que rigen las operaciones sobre estos datos. Se utiliza para diseñar y construir bases de datos de alta calidad y bien estructuradas que permitan la recuperación y manipulación de

los datos de manera eficiente y efectiva. Un modelo de datos se suele representar en forma de diagrama, en el que se utilizan símbolos para representar las entidades, las relaciones y otros elementos importantes. El modelo de datos permite a los usuarios visualizar la estructura de la base de datos y entender cómo se relacionan los diferentes elementos.

Además de los tres anteriores puntos las características se analizarán las siguientes características de las bases de datos, para posteriormente hacer la comparación:

- Lenguajes de consulta.
- Ventajas de usar este sistema para series temporales.
- Entornos de despliegue.
- Almacenamiento de los datos.
- Operaciones de lectura, escritura y acceso a los datos.

3.2 InfluxDB

3.2.1 Introducción a InfluxDB

InfluxDB como ya hemos visto, es una base de datos de series temporales, de código abierto con versión gratuita, diseñada específicamente para almacenar y analizar grandes conjuntos de datos que cambian con el tiempo. Utiliza una arquitectura altamente optimizada para admitir una escritura rápida y una consulta de datos eficiente en series temporales.

En cuanto a su uso práctico, InfluxDB se utiliza a menudo en aplicaciones IoT, para la monitorización del rendimiento de nuestras aplicaciones, registros en la nube, para la recopilación de datos en tiempo real de sistemas y aplicaciones, para realizar búsquedas mediante filtros de nuestros datos y análisis de los mismos.

3.2.2 Estructura de InfluxDB

La estructura de datos de InfluxDB gira en torno a la idea de las series temporales.

Cada una de nuestras series temporales se pueden identificar haciendo uso de etiquetas, que se utilizan para agrupar y filtrar puntos similares, y un conjunto de campos, que corresponden a los valores almacenados en cada punto. Además, InfluxDB usará la marca de tiempo de nuestros puntos de datos como el índice para ordenarlos.

Dentro de nuestras bases de datos que creemos, pueden existir múltiples series temporales y cada una de ellas puede tener diferentes etiquetas, campos y marcas de tiempo.
La estructura de InfluxDB para tratar las series temporales se basa en los conceptos de buckets (contenedores), measurements (mediciones), tags (etiquetas), fields (campos) y políticas de retención. Estos conceptos permiten organizar y almacenar grandes cantidades de datos de series temporales mientras se mantiene la eficiencia de la base de datos.

1. Buckets (contenedores): son contenedores virtuales dentro de una base de datos que se utiliza para almacenar y organizar datos relacionados. Los buckets están diseñados para facilitar la organización y la administración de datos. Dentro del bucket almacenaremos las mediciones que contienen campos para datos, etiquetas que permiten agregar información adicional a los datos y podremos establecer políticas de retención de datos para controlar cuánto tiempo se almacenan los datos. En InfluxDB, los buckets son específicos de la base de datos en la que se crean y no se pueden compartir entre bases de datos.

2. Tags (etiquetas): Las etiquetas en InfluxDB son pares de clave-valor que se utilizan para etiquetar los datos almacenados en una base de datos. Las etiquetas se pueden usar para filtrar, agrupar, buscar datos, utilizándose como criterio de búsqueda para seleccionar los datos. Las etiquetas se definen para una serie y no cambian con el tiempo. Cada serie puede tener múltiples etiquetas y cada etiqueta tiene un valor asociado. Con el uso de etiquetas podemos agregar información adicional y descriptiva a los datos almacenados en los campos.

3. Fields (campos): los campos son valores numéricos vinculados a una serie, que se almacenan en InfluxDB junto con las etiquetas, almacenar datos de tipo integer, float, booleano o strings. A diferencia de las etiquetas, los campos pueden cambiar con el tiempo. Cada punto en una serie puede tener múltiples campos asociados.

4. Measurements (mediciones) y Series: en InfluxDB, una medición es un conjunto lógico de datos que se almacenan junto con un nombre de medición, son una forma de organizar y agrupar series de datos. Los datos se organizan en series, que son instancias de una medición. Una serie consta de un nombre de medición, un conjunto de etiquetas y un conjunto de campos. Los Measurements se configuran en una base de datos y definen el esquema o estructura de una serie. Cada serie se agrupa en un Measurement y cada punto de datos en la serie se almacena como un registro en la base de datos, y también se utilizan para definir las políticas de retención de datos.

5. Políticas de retención: se utilizan para establecer durante cuánto tiempo se almacenarán los datos en la base de datos. Las políticas de retención pueden establecerse a nivel de bucket. Nos sirve como herramienta para controlar la cantidad de datos almacenados en una base de datos. Las políticas de retención permiten que los datos se eliminen automáticamente después de cierto periodo de tiempo, lo que ayuda a reducir el tamaño de la base de datos y garantiza que solo se almacenen los datos más recientes, optimizando el espacio, y los más relevantes que queremos guardar como histórico de la base de datos.

   En InfluxDB, se pueden crear múltiples políticas de retención para una base de datos. Las políticas de retención se pueden aplicar a toda la base de datos o solo a una parte específica de ella, como un Measurement o una serie. Podemos definir las políticas de retención en función del tiempo o del tamaño de los datos, teniendo en cuenta que una vez que eliminemos los datos el borrado de los mismo es permanente.

De manera que en la estructura de InfluxDB tendremos nuestra base de datos donde almacenaremos nuestros buckets, que contienen datos relacionados. Los datos se etiquetan y se organizan en mediciones y series. Los campos se almacenan junto con las etiquetas, y se pueden indexar para facilitar su búsqueda y agregación. Las políticas de retención se utilizan para establecer durante cuánto tiempo se almacenarán los datos en la base de datos.

3.2.3 Índices de InfluxDB

En InfluxDB, los índices se crean automáticamente para las etiquetas y campos que se usan con frecuencia en las consultas. Las etiquetas son clave-valor pares que se agregan a cada punto de datos para metadatos adicionales sobre la serie de tiempo. Los campos son valores numéricos o de cadena que se registran en la base de datos como parte del punto de datos, y se utilizan para realizar cálculos o consultas.

Las etiquetas se indexan utilizando una versión modificada de una tabla hash, que permite búsquedas rápidas de etiquetas y sus puntos de datos asociados. Esta estructura de indexación está altamente optimizada para datos de series de tiempo y permite consultas extremadamente rápidas y eficientes, incluso en conjuntos de datos grandes. En InfluxDB, las etiquetas y los campos se indexan por separado. Las etiquetas se indexan utilizando una estructura de datos llamada TSI (índice de conjunto de etiquetas), mientras que los campos se indexan utilizando una implementación personalizada de la estructura de datos "B-tree".

InfluxDB utiliza estructuras de índice de "B-tree" para organizar los datos y permitir búsquedas eficientes. Estas estructuras de datos se componen de nodos que contienen un número fijo de claves y punteros a otros nodos del árbol. Las claves son utilizadas para organizar los datos de una manera jerárquica, y los punteros se utilizan para buscar y acceder a los datos.

Estos índices se dividen en tres niveles: nivel de base de datos, nivel de serie de tiempo y nivel de punto de datos. Cada nivel tiene su propio índice, y los índices más bajos (nivel de punto de datos) son más detallados y específicos que los niveles superiores.

Algunas técnicas para optimizar las consultas incluyen la selección de los campos y etiquetas de las series de tiempo para incluir en las consultas, así como el uso de funciones de agregación para reducir el volumen de datos retornados por la consulta.

3.2.4 Modelo de los datos de InfluxDB

En InfluxDB, cada punto de datos se compone de un tiempo, uno o varios campos y etiquetas opcionales. El tiempo es una marca de tiempo Unix (número de milisegundos desde el 1 de enero de 1970 UTC) que determina cuándo se registró el punto de datos. Los campos son los valores que se registran y se almacenan, como la temperatura, la humedad o la presión. Las etiquetas son pares clave-valor que se utilizan para identificar y agrupar los puntos de datos, como la ubicación, el tipo de dispositivo o el ID de usuario.

Los puntos de datos se organizan en series de tiempo, que son identificadas por su nombre y etiquetas. Cada serie de tiempo tiene al menos un punto de datos y puede tener cientos, miles o millones de puntos de datos. Las series de tiempo se almacenan en la base de datos de InfluxDB y están indexadas por nombre y etiquetas para facilitar la búsqueda y el análisis. Además, InfluxDB utiliza un modelo de compresión de datos llamado TDB (Time-Structured Merge Tree) que permite una reducción significativa en el espacio de almacenamiento necesario para los datos de series de tiempo. TDB se basa en técnicas de segmentación, purga y compresión de datos, lo que permite un alto rendimiento en la lectura y escritura de datos mientras se mantiene la eficiencia de almacenamiento.

InfluxDB también utiliza particionamiento automático en las series de tiempo para distribuir los datos en diferentes máquinas y mejorar la escalabilidad y el rendimiento del sistema. El particionamiento se basa en la marca de tiempo de los datos y se realiza en función de intervalos de tiempo específicos. InfluxDB puede configurarse para distribuir las series de tiempo en diferentes máquinas para mejorar el rendimiento y escalar en varias dimensiones.

3.2.5 Características de InfluxDB

Dentro de las características de InfluxDB primero hablaremos de los lenguajes de consulta, este sistema utiliza dos lenguajes de consulta: InfluxQL y Flux. Ambos lenguajes de consulta permiten a los usuarios realizar consultas de datos, filtrar datos, agregar datos y realizar otras operaciones en los datos almacenados en la base de datos de InfluxDB:

- InfluxQL es un lenguaje de consulta SQL-Like, optimizado para trabajar con datos de series temporales, es el lenguaje utilizado en las antiguas versiones de InfluxDB.
- Flux es un lenguaje de scripting de datos diseñado para consultar, analizar y actuar sobre datos de series temporales específicamente. Flux es una nueva adición a InfluxDB a partir de la versión 1.7 y es compatible con InfluxDB 2.0, versión que utilizaremos en este trabajo. Flux ofrece más flexibilidad y capacidad de personalización en la abstracción de datos

Seguidamente hablaremos de las principales ventajas que podemos aprovechar con InfluxDB para trabajar con series temporales:

- Eficiencia en el almacenamiento: ya que InfluxDB ha sido diseñado específicamente para el almacenamiento y análisis de datos de series temporales, lo que le permite almacenar y recuperar datos de manera eficiente. InfluxDB utiliza un modelo de almacenamiento basado en series de tiempo, lo que significa que los datos se ordenan cronológicamente, de manera que se puede acceder a los datos, y filtrarlos de manera más rápida y eficiente.

- Escalabilidad: InfluxDB puede manejar grandes volúmenes de datos a alta velocidad, y es altamente escalable horizontalmente. Esto significa que se pueden agregar más nodos de InfluxDB a un clúster para manejar un mayor volumen de datos a medida que se necesite.

- Monitorización y visualización: InfluxDB se integra bien con herramientas y complementos de monitorización y visualización como Grafana, Telegraf, Kapacitor y Chronograf ofreciendo así una mayor funcionalidad y flexibilidad, para el almacenamiento y el análisis de datos de series temporales, lo que nos permite crear dashboards personalizados con información en tiempo real.

- API de escritura: InfluxDB cuenta con una API de escritura que permite interactuar con la base de datos a través de HTTP. Esta API RESTful de InfluxDB admite todas las operaciones CRUD (Crear,

Leer, Actualizar, Eliminar). Gracias a la API de InfluxDB podemos interactuar con la base de datos de una manera sencilla y flexible, lo que facilita la administración y el análisis de los datos. Además, al ser una API RESTful, puede integrarse fácilmente con otras aplicaciones y sistemas de terceros.

- Comunidad activa: InfluxDB cuenta con una comunidad activa de usuarios y desarrolladores que contribuyen al desarrollo y mejora de la base de datos. Esto significa que InfluxDB está en constante evolución y mejora.

Además, InfluxDB se puede desplegar en diferentes entornos:

1. Despliegue en la nube: InfluxDB se puede desplegar en la nube, utilizando servicios como Amazon Web Services, Google Cloud y Microsoft Azure.
2. Despliegue en el propio dispositivo del usuario: InfluxDB también puede ser desplegado en servidores locales o en dispositivos de Internet de las cosas (IoT).
3. Despliegue en sistemas de contenedores: InfluxDB se puede desplegar en contenedores, como Kubernetes y Docker, para facilitar su gestión y escalabilidad.

InfluxDB también tiene disponible una versión de pago para empresas con funcionalidades adicionales que no se encuentran en la versión gratuita. Algunas de estas funcionalidades incluyen la recuperación ante desastres, la replicación de datos a través de múltiples clústeres, la capacidad de escalar horizontalmente a más nodos y la integración con herramientas de seguridad empresarial.

3.2.6 Operaciones de lectura, escritura, acceso y almacenamiento de los datos

- **Almacenamiento:**

InfluxDB almacena los datos utilizando una estructura de datos llamada "grupo de escritura", que es una colección de índices y segmentos de datos escritos en el disco.
Los datos se escriben en primer lugar en un registro de escritura caché llamado "registro de escritura de memoria", que es una estructura de almacenamiento temporal en la memoria RAM. A medida que los datos se van acumulando en el registro de escritura de memoria, InfluxDB los va escribiendo en archivos de segmentos de disco llamados "archivos de segmentos de TSM" (Table Storage Memory-mapped files), que son archivos ordenados y compactos. Estos segmentos de disco son los que se utilizan finalmente para hacer la lectura de datos.

Hace uso de un mecanismo de compresión para reducir el tamaño de los datos almacenados, conocido como "compresión de nivel variable". En esta técnica, InfluxDB compara el último valor registrado con el valor actual y si es el mismo, se almacena un puntero que indica que ambos valores son iguales, en lugar de almacenar el segundo valor completo. Esto reduce significativamente el tamaño de los datos almacenados en disco.

A diferencia de las bases de datos relacionales, InfluxDB es una base de datos "sin esquema", es decir, que no existe una estructura predefinida para los datos, por lo que los datos pueden ser escritos en la base de datos sin ningún tipo de validación previa asegurando la coherencia y consistencia de los datos. Para manejar esa problemática y asegurar la coherencia, InfluxDB ofrece un modelo de datos y separación de datos a través de series. Donde cada serie de tiempo se compone de un nombre y un conjunto de etiquetas y campos.

Las etiquetas se utilizan para agregar metadatos a la serie de tiempo, como información contextual adicional que no cambia con el tiempo. Por ejemplo, si se está monitoreando la temperatura en diferentes regiones, las etiquetas podrían incluir información sobre la ubicación o el tipo de sensor utilizado.

Los campos son la información real almacenada en la serie de tiempo y pueden cambiar con el tiempo. Por ejemplo, en el caso de la temperatura, el campo podría ser el valor numérico que representa la temperatura en ese momento. La separación de datos en series permite una mayor granularidad y flexibilidad en la organización y consulta de datos, lo que a su vez ayuda a asegurar la coherencia de los datos.

- **Lectura:**

Utiliza una estructura de datos de índice invertido llamada índice de campo (FIELD index) para un acceso rápido a los puntos de datos. El proceso de lectura de datos en InfluxDB funciona de la siguiente manera:

1. Consulta: la persona usuaria realiza una consulta a la base de datos especificando los puntos de datos que se desean recuperar. Estas consultas pueden ser en lenguaje InfluxQL o Flux.

2. Optimización de consulta: InfluxDB realiza una optimización de consulta para tratar de recuperar los datos de manera más eficiente. Para ello, utiliza información de metadata almacenada con los datos, como los tamaños de retención y el intervalo de retención.

3. Indexado: Siempre que sea posible, InfluxDB aprovechará los índices para reducir el número de puntos de datos que son escaneados. Podría utilizar uno de los tres índices disponibles: índice de campo (FIELD index), índice de tag (TAG index) y índice de tiempo (TIME index).

4. Escaneo: InfluxDB escanea los puntos de datos que cumplen con la consulta y los retorna en el formato solicitado. Para mejorar el rendimiento, InfluxDB utiliza una técnica de compresión llamada codificación delta y codificación de ejecución (delta and run-length encoding) que reduce el espacio de almacenamiento y aumenta la velocidad de acceso a los datos.

- **Acceso y escritura de los datos:**

En InfluxDB, los datos se escriben y se acceden mediante el uso de una interfaz HTTP/HTTPS. Los datos son almacenados en una base de datos de series de tiempo, que es un tipo de base de datos optimizada para consultas de tiempo. Los datos se almacenan en un formato lineal llamado Protocol Buffer, que se comprime según el algoritmo de compresión snappy.

Para escribir datos en InfluxDB, se envía una solicitud HTTP a una de las API de escritura de InfluxDB. La solicitud incluye los datos a ser escritos, así como la información de la base de datos y la serie de tiempo. Los datos se insertan en la base de datos y se pueden agregar automáticamente a una serie de tiempo existente o crear una nueva serie de tiempo. Para acceder a los datos en InfluxDB, se realiza una consulta a través de una de las API de consulta de InfluxDB. Las consultas se escriben en el lenguaje de consulta InfluxQL, que es similar a SQL. Las consultas se ejecutan en la base de datos de series de tiempo y devuelven los resultados solicitados en formato JSON.

- **Replicación:**

En InfluxDB, la replicación se utiliza para mantener redundancia en los datos y garantizar su disponibilidad en caso de fallo en uno o varios nodos del cluster. La replicación se realiza mediante una estrategia distribuida que utiliza grupos de alta disponibilidad (HA), los cuales se componen de un líder y varios seguidores.

El líder en un grupo de HA es el nodo maestro que maneja todas las solicitudes de escritura y controla la sincronización de los datos con los seguidores. El líder en un grupo de HA se elige automáticamente mediante el algoritmo de consenso Raft, que selecciona al líder en función de la disponibilidad y el estado de los nodos del cluster. Los seguidores en un grupo de HA son nodos esclavos que mantienen una copia sincronizada de los datos almacenados en el líder. Los seguidores también pueden servir solicitudes de lectura en caso de que el líder falle. La sincronización entre el líder y los seguidores se realiza mediante el uso de un protocolo de replicación basado en la técnica de registro de seguimiento binario (binary log replay).

En este protocolo, el líder registra todas las solicitudes de escritura en un registro binario, que es utilizado luego para sincronizar los seguidores. Los seguidores se conectan al líder y recuperan los datos a través de la copia del registro binario registrado por el líder. En caso de que un seguidor se quede atrás en la sincronización, el líder envía los datos necesarios para asegurarse de que los datos en el seguidor están actualizados.

3.3 TimescaleDB

3.3.1 Introducción a TimescaleDB

TimescaleDB es una base de datos de series temporales de código abierto, construida como una extensión de PostgreSQL. Como base de datos de series temporales, TimescaleDB se especializa en almacenar y recuperar datos relacionados con el tiempo, incluyendo series de datos financieros, sensores de IoT, registros de aplicaciones web y mucho más.

TimescaleDB utiliza una arquitectura distribuida y escalable que está diseñada para manejar grandes volúmenes de datos de series temporales. Permitiendo el almacenamiento de datos en nodos distribuidos, lo que facilita el aumento de la escala al manejar grandes volúmenes de datos, e incluye una variedad de características avanzadas para el procesamiento de datos de series temporales, incluyendo agregación de datos, interpolación de datos, filtrado flexible y soporte para múltiples zonas horarias.

También incluye soporte para consultas SQL complejas y una variedad de herramientas y bibliotecas, lo que facilita la integración con otras aplicaciones. Esto lo hace una excelente opción para aplicaciones que requieren análisis y visualización de datos en tiempo real.

3.3.2 Estructura de TimescaleDB

La estructura de los datos de TimescaleDB es orientada a columnas. Esto significa que, en lugar de almacenar toda la información para una única fila en una tupla, los datos se almacenan en columnas separadas. Esto permite una mayor eficiencia en la gestión de grandes volúmenes de datos y facilita el acceso a la información.

Los datos se dividen en particiones basadas en el rango de tiempo y se distribuyen en varias máquinas. Esto permite la gestión eficiente de grandes volúmenes de datos sin comprometer el rendimiento. Cada partición contiene un conjunto separado de datos de series temporales y es gestionada por una instancia separada de TimescaleDB, estas tienen una marca de tiempo inicial y final.

La estructura de datos de TimescaleDB se basa en la funcionalidad de particionamiento temporal de PostgreSQL. Este sistema hace una partición automática de la matriz de datos, donde el flujo de datos de entrada se distribuye automáticamente entre las tablas particionadas.

Las secciones se crean en función del tiempo (cada sección almacena datos durante un cierto período de tiempo) o en relación con una clave arbitraria (por ejemplo, identificador del dispositivo, ubicación, etc.). Las tablas particionadas se pueden distribuir entre diferentes unidades.

Lo que permite a los usuarios dividir los datos de series temporales en fragmentos de tiempo predeterminados, como minutos, horas o días, para que puedan ser procesados y consultados de manera más eficiente. Cada fragmento se guarda como una tabla separada en la base de datos, lo que se llama una "hipertabla". Una hipertabla es una representación virtual de muchas tablas separadas en las que se acumulan datos entrantes.

Además, utiliza indexación automática para acelerar las consultas en datos de series temporales. TimescaleDB aprovecha el índice de Postgres llamado B-tree, así como estructuras de índice de longitud variable, para acelerar las lecturas de datos de series temporales, lo que evita la necesidad de realizar búsquedas secuenciales en el disco.

Otra característica importante de TimescaleDB es su conjunto completo de funciones y operadores para el análisis de series temporales. Estas funciones y operadores incluyen cálculos estadísticos, agregaciones, transformaciones, interpolaciones y resampling, entre otros. TimescaleDB también proporciona un lenguaje especializado de consultas para el análisis de series temporales.

Al igual que InfluxDB utiliza técnicas de compresión inteligente para reducir la cantidad de espacio de almacenamiento necesario y acelerar el acceso a los datos.

3.3.3 Índices de TimescaleDB

La indexación en TimescaleDB se divide en dos tipos de índices: los índices B-tree, que son similares a los índices en muchas otras bases de datos relacionales, y los índices de hipercubo, que son específicos de la funcionalidad de series temporales de TimescaleDB.

Los índices B-tree se basan en estructuras de árboles, se utilizan para identificar rápidamente los elementos de una tabla que contienen un valor determinado de una o varias columnas. En TimescaleDB, los índices B-tree se usan para acelerar búsquedas y consultas rápidas en datos que se almacenan por rangos de tiempo.

Los índices de hipercubo, por otro lado, se utilizan para acelerar búsquedas y filtrado por múltiples dimensiones. En TimescaleDB, estos índices se crean a partir de las series temporales, que son tablas especializadas diseñadas para contener datos temporalmente relacionados.

En la creación de un índice de hipercubo, se especifica la dimensión de la tabla que se desea indexar, como el tiempo y una o varias columnas específicas, que se usarán para construir un hipercubo multidimensional. Los datos se organizan en hipercubos (slots) que contienen múltiples valores de los campos o tags indexados.

La indexación en TimescaleDB también se beneficia de la capacidad del motor de la base de datos de limitar y optimizar el conjunto de datos que se deben procesar. Esto se realiza mediante la segmentación de las series temporales en segmentos de tiempo más pequeños conocidos como chunks. Cada chunk se divide en filas, y las filas pueden contener múltiples campos, etiquetas y marca de tiempo.

Estos chunks de tiempo se indexan usando tanto índices B-tree como de hipercubo, lo cual permite que la base de datos pueda realizar búsquedas específicas y ágiles en los datos históricos y actuales utilizando diferentes dimensiones o etiquetas.

3.3.4 Modelo de los datos de TimescaleDB

El modelo de datos de TimescaleDB es similar al de cualquier base de datos relacional. Sin embargo, a diferencia de las bases de datos relacionales tradicionales, se especializa en el almacenamiento y la manipulación de datos de series temporales, como mediciones de sensores, logs de servidores y datos de IoT.

Para este propósito, TimescaleDB introduce dos conceptos: las series temporales y las etiquetas.

- Las series temporales como una forma de organizar los datos por fecha y hora, es decir, el tiempo es la dimensión más importante.
- Las etiquetas son campos adicionales que se utilizan para asignar más metadata a los datos y permiten una indexación más eficiente y una búsqueda más rápida.

Cada serie de tiempo en TimescaleDB se almacena en una tabla, y un conjunto de series de tiempo relacionadas se agrupan en hipertablas. Además, incluye muchas funciones para manipular y transformar datos de series de tiempo. Estas funciones incluyen operaciones básicas de agregación y consulta, así como muchas funciones especializadas para trabajar con datos de series de tiempo, como la interpolación, la extracción de características y el pronóstico.

TimescaleDB permite escalamiento horizontal útil en la gestión de grandes conjuntos de datos en dónde el incremento de datos no impacta la operación de la base de datos, esto permite el crecimiento de la base, lo que es conveniente para dispositivos IoT y la Ingeniería de tiempo real.

3.3.5 Características de TimescaleDB

Primeramente, hablaremos de los lenguajes de consulta:

- Los lenguajes de consulta de TimescaleDB son los mismos que los de la base de datos PostgreSQL en la que se basa. Esto incluye el lenguaje de consulta estructurado (SQL) estándar de la industria, así como algunas extensiones específicas de PostgreSQL. Las consultas también pueden combinarse con otras características de PostgreSQL, como funciones almacenadas y vistas materializadas.

Dentro de las ventajas que tenemos al usar este sistema de base de datos orientado a series temporales encontramos las siguientes:

- Su escalabilidad horizontal: TimescaleDB ha sido diseñado para escalar horizontalmente, lo que permite agregar nodos adicionales a medida que crece el volumen de datos. Esto significa que puedes manejar grandes cantidades de datos sin sacrificar el rendimiento.

- Eficiencia de almacenamiento: utiliza una técnica llamada compresión de datos para reducir el tamaño de los datos almacenados. Esto permite almacenar más datos en menos espacio de almacenamiento.

- Compatibilidad: es completamente compatible con SQL y ofrece muchas de las mismas características de escalabilidad que se encuentran en las aplicaciones convencionales. Además, puede ser utilizado en combinación con otras herramientas de datos como Apache Kafka y Apache Spark.

- Alta disponibilidad y resiliencia: utiliza una combinación de replicación y particionamiento para garantizar la disponibilidad y la resiliencia en caso de fallos del hardware o pérdida de datos.

Los entornos de despliegue que encontramos en TimescaleDB son también los mismos que en el caso de InlfuxDB:

1. En la nube: se integra con proveedores de servicios en la nube como Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP).

2. En el mismo dispositivo: TimescaleDB puede ser desplegado en un entorno on-premise, permitiendo a los usuarios controlar la ubicación y el acceso de sus datos.

3. En contenedores: ya que es compatible con contenedores de Docker, lo que facilita su implementación en contenedores y en entornos que hacen uso de tecnologías de orquestación de contenedores como Kubernetes.

3.3.6 Operaciones de lectura, escritura, acceso y almacenamiento de los datos

- **Almacenamiento:**

TimescaleDB almacena los datos en una estructura de tabla dividida en particiones de tiempo, que se organiza en índices de hipercubos, B-tree. Estos índices son multidimensionales, lo que significa que permiten consultas ad-hoc en diferentes dimensiones, como pueden ser tiempo y otros atributos con los que las tablas han sido particionadas (por ejemplo, geolocalización). Los datos se almacenan en forma de tuplas (filas), que contienen los diferentes atributos o variables asociados a un punto en el tiempo o espacio.

Otra técnica de almacenamiento utilizada por TimescaleDB es la compresión de datos. El motor de TimescaleDB comprime automáticamente los datos usando algoritmos de compresión sin pérdida, lo que permite a los usuarios reducir el tamaño de los datos de sus series temporal. En lugar de descomprimir todos los datos antes de una consulta, TimescaleDB descomprime solo los datos relevantes en el momento, lo que acelera el proceso de consulta e incrementa la eficiencia.

- **Lectura:**

Cuando se realiza una consulta en TimescaleDB, el motor primero busca en los índices de hipercubos B-tree para encontrar los datos relevantes. A continuación, busca solo en las particiones de la tabla que contienen los datos seleccionados por la consulta, lo que acelera el proceso de búsqueda. TimescaleDB también utiliza la compresión de datos para reducir la cantidad de datos que se deben leer del disco, lo que mejora aún más el rendimiento de las consultas.

Una vez que se han encontrado los datos relevantes, TimescaleDB los devuelve al usuario en forma de tuplas (filas) que contienen los diferentes atributos o variables asociados a un punto en el tiempo (o espacio). TimescaleDB también permite la agregación de datos y el cálculo de estadísticas en tiempo real, lo que puede ser útil para la monitorización y el análisis de rendimiento.

- **Acceso y escritura de los datos:**

Los datos pueden ser escritos y modificados en TimescaleDB utilizando consultas SQL estándar.

Para enviar una consulta, TimescaleDB primero verifica los índices de hipercubo B-tree para identificar las particiones de tabla que contienen los datos que son relevantes para la consulta. A continuación, accede a las particiones en el disco que contienen los datos seleccionados por la consulta.

Cuando se escribe un nuevo conjunto de datos, TimescaleDB lo almacena de manera eficiente en las particiones de la tabla correspondientes al rango de tiempo apropiado. Los datos pueden ser escritos directamente a través de consultas INSERT, o mediante aplicaciones que escriban datos a través de una API.

En caso de que se modifiquen los datos, se usa una técnica de escritura diferencial para evitar sobrescribir datos antiguos innecesariamente. En lugar de eliminar los datos anteriores y escribir nuevos datos, escribiendo sólo los datos que han cambiado desde la última escritura. Esta técnica de escritura diferencial es especialmente útil para mantener el alto rendimiento y la escalabilidad de las escrituras incluso en grandes conjuntos de datos.

Además, TimescaleDB utiliza transacciones ACID para garantizar la integridad de los datos durante la escritura y la modificación. Esto significa que las transacciones se completan o se anulan en su totalidad, y todas las operaciones dentro de una transacción se realizan con éxito o no se realizan en absoluto.

La arquitectura distribuida de TimescaleDB permite el acceso a los datos a través de múltiples nodos y réplicas de base de datos, lo que mejora la disponibilidad y el rendimiento del acceso a los datos. Además, TimescaleDB cuenta con herramientas de monitorización y diagnóstico para ayudar a los usuarios a optimizar el rendimiento de sus consultas y mejorar la eficiencia del acceso a los datos.

- **Replicación:**

La replicación en TimescaleDB se realiza mediante el proceso de streaming replication, que se basa en el envío de datos binarios de transacciones a un conjunto de réplicas que se encuentran sincronizadas con el nodo primario.

También cuenta con replicación del nodo activo a un nodo pasivo para garantizar la disponibilidad y la continuidad del servicio. Y dispone de técnicas para resolver posibles errores de replicación, como la validación del registro de transacciones y la reconciliación de datos faltantes.

Otra técnica importante que se utiliza en TimescaleDB se llama chunking, que consiste en la división de los datos de series temporales en segmentos más pequeños y manejables vistos anteriormente, llamados chukns.

Para replicar estos chunks en un nodo secundario, se utiliza el mecanismo de "continuous Archiving and Point-In-Time Recovery" o PITR, que permite la restauración del estado de una base de datos a un momento específico en el pasado. De este modo, se puede replicar y mantener sincronizado el conjunto completo de datos en múltiples nodos sin interrupciones del servicio.

3.4 Comparativa TimescaleDB con PostgreSQL

3.4.1 Introducción a las bases de datos relacionales (PostgreSQL)

Las bases de datos relacionales son aquellas donde los datos se organizan en una o más tablas o relaciones, cada una de las cuales tiene una clave o identificador único, y que constan de filas y columnas. Las relaciones entre estas tablas se definen mediante el uso de claves externas, que vinculan los datos de una tabla con los de otra.

Por otro lado, las bases de datos de series temporales son una extensión de las bases de datos relacionales que se utilizan para almacenar y gestionar grandes cantidades de datos de series temporales. Las bases de datos de series temporales están diseñadas para manejar grandes cantidades de datos de series temporales mientras mantienen altas tasas de inserción constantes.

En este apartado se analizará la base de datos relacional de PostgreSQL, para comprobar las ventajas que obtenemos al usar bases de datos como su extensión TimescaleDB o InfluxDB para tratar los datos de series temporales.

PostgreSQL es una base de datos relacional de código abierto, y su extensión TimescaleDB agrega funcionalidades que sirven para manejar series temporales. Con TimescaleDB, los usuarios pueden escalar a miles de millones de filas en PostgreSQL, pudiendo almacenar sus metadatos relacionales y datos de series temporales juntos en la misma base de datos.

3.4.2 Características de PostgreSQL

En este apartado se analizarán las principales características de PostgreSQL, para luego hacer una comparativa entre este sistema de base de datos relacional puro, con InfluxDB y TimescaleDB como sistemas de bases de datos temporales, viendo así la importancia en la forma de manejar estos datos por parte de estos 2 sistemas.

Como características destacables de PostgreSQL, encontramos los siguientes puntos:

- PostgreSQL soporta una amplia variedad de tipos de datos complejos como matriz, JSON, XML, hstore y geometría, lo cual permite a los desarrolladores almacenar y manipular los datos de manera más flexible y eficiente.
- Escalabilidad: gracias a esta puede manejar grandes cantidades de datos. Admitiendo también la replicación y la partición de datos.
- Índices avanzados ya que admite una amplia variedad, como índices de texto completo, índices de expresiones e índices espaciales.
- Soporte para JSON y XML pudiendo así trabajar con datos no estructurados, de forma eficiente.
- Seguridad ya que se puede hacer separación de privilegios, limitando el uso, acceso, y modificación de los datos según los usuarios que tengan permisos o no.

Además, PostgreSQL admite soporte en múltiples plataformas como Linux, Windows, macOS y soportando múltiples lenguajes de programación para trabajar en el sistema, como son C, C++, Java, Perl, Python y Ruby

- **Consultas SQL:**

Las consultas SQL (Structured Query Language) en PostgreSQL son los comandos que se utilizan para interactuar con una base de datos relacional.

Una consulta SQL en PostgreSQL puede incluir varios comandos, como SELECT (para extraer datos de la base de datos), INSERT (para agregar nuevos datos a la base de datos), UPDATE (para modificar datos existentes en la base de datos) y DELETE (para eliminar datos de la base de datos).

Para la selección de datos, la cláusula WHERE se utiliza para filtrar los datos basados en un conjunto de condiciones. Además, se pueden utilizar varios operadores como AND, OR e IN junto con las condiciones para agregar una granularidad más fina. También puede tener funciones y expresiones que se pueden utilizar en las consultas, estas pueden ser funciones matemáticas, de texto, de fecha y hora, entre otras.

- **Funciones almacenadas:**

Las funciones almacenadas en PostgreSQL son un conjunto de instrucciones que se almacenan dentro del motor de la base de datos y que se pueden llamar desde una consulta SQL o desde una aplicación externa. Estas funciones son útiles para realizar tareas específicas que no se pueden realizar con las consultas SQL estándar, como, por ejemplo, la ejecución de tareas complejas o la creación de funciones personalizadas para un uso específico.

Las funciones almacenadas se pueden dividir en dos categorías:

1. Las funciones de usuario son aquellas que son creadas por el usuario y que realizan una tarea específica.
2. Las funciones internas son aquellas que están integradas en el propio sistema por defecto y que se utilizan para realizar tareas específicas, como, por ejemplo, la conversión de tipos de datos.

El uso de estas funciones tiene varias ventajas, como una mayor eficiencia que las consultas SQL estándar, ya que están precompiladas y optimizadas para su ejecución en el motor de la base de datos, y su uso es útil para la creación de aplicaciones complejas, al modular las funciones.

- **Triggers:**

Los triggers en PostgreSQL son objetos que permiten la ejecución automática de una serie de comandos en respuesta a un evento específico en la base de datos. Estos eventos pueden ser la inserción, actualización o eliminación de datos en una tabla específica, o la ejecución de una sentencia SQL específica en la base de datos.

Los triggers se implementan como procedimientos almacenados, y se pueden definir en una tabla específica o en la base de datos en su conjunto. Una vez que se ha definido un trigger, se activará automáticamente en respuesta a los eventos especificados.

Estos pueden ser útiles para realizar validaciones de datos o auditar los cambios en una tabla específica. También se pueden utilizar para actualizar automáticamente los datos en otras tablas cuando se insertan, actualizan o eliminan registros en una tabla específica, lo que puede ahorrar mucho tiempo y esfuerzo en el mantenimiento y actualización de la base de datos.

Un ejemplo de estos triggers son los comandos BEFORE y AFTER. Los triggers de tipo BEFORE se activan antes de que se realice una acción en una tabla específica, mientras que los triggers de tipo AFTER se activan después de que se haya realizado una acción en una tabla específica.

- **Vistas:**

Las vistas en PostgreSQL son objetos virtuales que se comportan como tablas, pero que no contienen datos propios y se definen mediante una consulta SQL. Las vistas se utilizan para simplificar consultas complejas, permitiendo que los datos se vean desde diferentes perspectivas y que los usuarios finales puedan acceder fácilmente a los datos que necesitan.

Las vistas se crean utilizando la sintaxis `CREATE VIEW` y pueden contener cualquier número de columnas definidas en la consulta. También se pueden utilizar para crear consultas complejas que involucren múltiples tablas y uniones. Al igual que las tablas, las vistas se pueden consultar, actualizar y eliminar utilizando instrucciones de SQL.

Las vistas en PostgreSQL tienen varias ventajas: son muy flexibles y permiten que los datos se vean desde diferentes perspectivas sin tener que crear nuevas tablas, se pueden ocultar datos confidenciales y sólo permitir el acceso a datos no sensibles, lo que mejora la privacidad y la seguridad de los datos.

Sin embargo, no son adecuadas para grandes volúmenes de datos, ya que el rendimiento puede verse afectado, y además no sirven para realizar operaciones de actualización complejas, como insertar, actualizar o eliminar varias filas de datos.

PostgreSQL utiliza un enfoque de registro para lograr la durabilidad de los datos a través del uso de transacciones.

Una transacción en PostgreSQL es una secuencia de comandos de base de datos que se ejecutan como una única unidad atómica. Esto significa que todas las operaciones en la transacción se ejecutan o ninguna se ejecuta. Si alguna operación falla, se hace un rollback de la transacción y se deshacen todas las operaciones. De esta forma, se asegura la integridad de los datos.

- **Lecturas:**

Las consultas de lectura en SQL son operaciones que recuperan los datos de la base de datos, mientras que las operaciones de escritura son las que modifican los datos, insertando, actualizando o eliminando registros en una tabla. Las operaciones de lectura también utilizan archivos de registro para asegurar la durabilidad de los datos. Cuando se realiza una lectura en PostgreSQL, la operación lee la información actual de la tabla o vista en la base de datos y la guarda en la memoria caché del servidor. Esta operación de lectura también se registra en el archivo de registro.

Cuando realizamos una operación de lectura en PostgreSQL, la base de datos carga los datos solicitados en la memoria caché del servidor para que estén disponibles para futuras consultas. Si otra consulta solicita los mismos datos, PostgreSQL puede acceder a ellos directamente en la memoria caché, lo que aumenta el rendimiento y reduce la sobrecarga en el sistema. Sin embargo, si los datos no están en la memoria caché, se generarán operaciones de E/S para recuperarlos del disco. La operación de lectura también se registra en el archivo de registro para asegurar la durabilidad de los datos.

- **Escrituras:**

En PostgreSQL, la escritura de datos se realiza mediante la inserción o actualización de registros en una tabla de la base de datos. El proceso de escritura de datos es esencial para la persistencia de los datos en la base de datos y es necesario para garantizar que los datos estén disponibles para su lectura en el futuro. Cuando se inserta un nuevo registro en una tabla de PostgreSQL, se crea un nuevo registro vacío en la tabla con un identificador único que se genera automáticamente. Luego, se insertan los datos en las columnas correspondientes del registro. Si hay restricciones de integridad referencial en la tabla, se verifica que se cumplan para garantizar la coherencia de los datos.

Cuando realizamos una operación de escritura, PostgreSQL garantiza la integridad de los datos y la durabilidad a través de la utilización del registro WAL (Write-Ahead Logging). Este mecanismo de registro garantiza que las operaciones de escritura se registren antes de que se actualicen los datos en disco, lo que previene la pérdida de datos y mantiene la integridad de la base de datos en caso de fallos del sistema.

También como medida de seguridad con PostgreSQL escribimos en disco un archivo de datos (data). Este archivo de datos es una imagen de la tabla o base de datos modificada. Si el sistema falla antes de que el archivo de registro se haya escrito en disco, se puede restaurar la copia más reciente de la tabla o la base de datos utilizando el archivo de datos. Sin embargo, hay que tener en cuenta que esto puede resultar en una pérdida de datos ya que las operaciones más recientes no se habrían registrado en el archivo de registros.

3.4.3 Limitaciones y comparativa con TimescaleDB

Ahora entendiendo un poco el sistema de PostgreSQL, se estudiarán las diferentes limitaciones que presenta, y la forma en la que sistemas como TimescaleDB, que es una extensión del propio PostgreSQL, amplían sus funcionalidades, viendo la importancia de utilizar sistemas específicos para tratar con los datos de series temporales.

Aunque PostgreSQL es una base de datos muy potente y flexible, tiene algunas carencias para tratar datos de series temporales. A continuación, se presentan algunas de las carencias más destacadas de PostgreSQL para tratar datos de series temporales:

- Falta de soporte nativo para series temporales, ya que no tiene soporte nativo para series temporales. Aunque PostgreSQL admite el almacenamiento y la manipulación de datos de fecha y hora, no tiene una estructura de datos dedicada para series temporales.
- Falta de funciones de agregación para series temporales: PostgreSQL no tiene funciones de agregación dedicadas para series temporales. Aunque PostgreSQL admite funciones de agregación como SUM y AVG, estas funciones no están optimizadas para series temporales.
- Falta de soporte para la interpolación de datos faltantes: PostgreSQL no tiene soporte para la interpolación de datos faltantes en series temporales. La interpolación de datos faltantes es una técnica común para rellenar los valores faltantes en una serie temporal.
- Falta de soporte para la detección de anomalías: PostgreSQL no tiene soporte para la detección de anomalías en series temporales. La detección de anomalías es una técnica común para identificar patrones inusuales en los datos de series temporales

Respecto a las mejoras que se obtienen con TimescaleDB y añade funcionalidades que PostgreSQL no tiene incorporadas, se pueden destacar los siguientes puntos:

- Políticas de retención de datos automáticas: TimescaleDB permite a los usuarios definir políticas de retención de datos que eliminan automáticamente los datos antiguos que ya no son necesarios. Esto ayuda a mantener la base de datos limpia y organizada.
- Agregados continuos: TimescaleDB permite a los usuarios crear agregados continuos que calculan automáticamente los valores agregados de los datos de series temporales. Esto permite a los usuarios realizar consultas más rápidas y eficientes.
- Compresión de datos: TimescaleDB utiliza técnicas de compresión de datos para reducir el tamaño de los datos almacenados en la base de datos. Esto ayuda a reducir el costo de almacenamiento y mejora el rendimiento de la base de datos.
- Integración con herramientas de análisis: TimescaleDB está diseñado para integrarse con herramientas de análisis como Grafana y Prometheus. La extensión proporciona una capa de API para tener aplicaciones que extiendan su uso, o tan útiles como Grafana, que facilitan la visualización de las series temporales.
- Estructura de la base de datos: una de las mayores diferencias entre PostgreSQL y TimescaleDB es la estructura de la base de datos. PostgreSQL está diseñado para manejar datos estructurados y no tiene una capacidad integrada para optimizar el manejo de series temporales. Por otro lado, TimescaleDB tiene una estructura de base de datos integrada para manejar series temporales, que se implementa mediante la creación de las hipertablas. Siendo capaces de dividir los datos en bloques de tiempo y optimizar las operaciones de lectura y escritura en los datos.
- Rendimiento: el manejo de una gran cantidad de datos de series temporales puede tener un impacto significativo en el rendimiento del sistema de gestión de bases de datos. PostgreSQL puede verse sobrecargado, especialmente si se manejan series temporales en grandes cantidades. TimescaleDB, por otro lado, ha sido diseñado específicamente para manejar series temporales y tiene características específicas para mejorar el rendimiento. Por ejemplo, se utiliza una técnica de particionamiento horizontal que mejora la escalabilidad.
- Funciones específicas para series temporales: TimescaleDB ha sido diseñado para manejar datos de series temporales de forma nativa, lo que implica la disponibilidad de funciones y características específicas para series temporales. Por ejemplo, cuenta con la capacidad de optimizar lecturas y escrituras de datos a través de la compresión, además de disponer de herramientas de agregación y cálculo específicas, como fill() para rellenar datos ausentes.

- **Comparación de lecturas:**

Las consultas en PostgreSQL pueden ser simples o complejas, y una gran cantidad de opciones están disponibles para personalizar la forma en que se leen los datos. PostgreSQL utiliza el modelo de procesamiento por lotes, lo que significa que las consultas se ejecutan en un lote, lo que puede ser más lento que el procesamiento de datos en tiempo real.
Cuando se leen los datos de la serie de tiempo en TimescaleDB, primero se buscan en un índice de la serie de tiempo que utiliza técnicas especializadas para realizar búsquedas muy rápidas para estos tipos de datos. Luego, los datos se recuperan automáticamente de las particiones correspondientes. Siendo más rápido que PostgreSQL para la lectura de grandes conjuntos de datos de la serie de tiempo.

De manera que, aunque tanto PostgreSQL como TimescaleDB utilizan SQL para realizar consultas, pero TimescaleDB utiliza una capa adicional de abstracción para hacer que la búsqueda de la serie de tiempo sea más rápida y eficiente.

- **Comparación de acceso y escritura de los datos:**

Para insertar datos en una tabla en PostgreSQL, se usa la cláusula INSERT INTO seguida de los valores que se desean insertar. La modificación de datos se realiza mediante la cláusula UPDATE, que permite actualizar los valores de las columnas en una tabla. La eliminación de datos se realiza a través de la cláusula DELETE, que permite eliminar registros completos de una tabla, utilizando transacciones para garantizar la integridad de los datos.

Por otro lado, cuando se escriben datos en TimescaleDB, primero se comprueba si el registro ya existe y, de ser así, se actualizan los registros con nuevos valores. Si no existe, se inserta un nuevo registro en la tabla

correspondiente. También utiliza una capa de abstracción llamada "chunking", como se ha explicado anteriormente, que divide los datos en trozos manejables, pudiendo así insertar grandes conjuntos de datos de la serie de tiempo de manera más eficiente en TimescaleDB. En cuanto a la modificación de datos, TimescaleDB permite la actualización y eliminación de los registros existentes utilizando consultas SQL estándar y añade la eliminación "automática" de datos antiguos en la serie de tiempo, lo que puede ser útil para mantener los datos actualizados.

- **Comparación de indexación:**

Para crear un índice en una tabla en PostgreSQL, se debe especificar la clase de índice a utilizar, así como la columna o conjunto de columnas que se deben indexar. PostgreSQL soporta diferentes tipos de índices, como índices de B-tree o hash.
Para mejorar la búsqueda de datos de la serie de tiempo, TimescaleDB utiliza un enfoque de particionamiento temporal en el que se dividen los datos de la serie de tiempo en particiones basadas en el tiempo. Cada partición se almacena en su propia tabla y se indexa por separado.

Para crear una tabla particionada en TimescaleDB, se especifica la columna temporal y el tamaño de partición deseado. Esto creará una tabla principal y varias particiones que se indexan automáticamente. Las particiones se crean automáticamente para cada intervalo de tiempo especificado, lo que significa que no es necesario ajustar las particiones manualmente, además de que TimescaleDB añade la funcionalidad de los índices hipercubo.

- **Comparación de estructura de las bases de datos:**

PostgreSQL y TimescaleDB son sistemas de bases de datos relacionales que comparten la estructura básica de una base de datos relacional. Ambos sistemas están diseñados para almacenar, organizar y procesar grandes cantidades de datos, pero TimescaleDB se ha especializado en datos de la serie de tiempo.

El motor de base de datos utiliza tablas para almacenar los datos y una variedad de técnicas de indexación para acceder a ellos de forma más rápida. Las tablas están compuestas por columnas y filas, donde cada columna define un tipo de dato y cada fila corresponde a un registro específico. Las tablas pueden estar relacionadas entre sí mediante claves primarias y extranjeras, lo que permite una mejor consulta y manipulación de datos.

La estructura de la base de datos en TimescaleDB es similar a la estructura en PostgreSQL, pero con enfoque en datos de la serie de tiempo. TimescaleDB utiliza una técnica llamada particionamiento temporal, que divide los datos de la serie de tiempo en pequeños trozos de tiempo, permitiendo una indexación más rápida para este tipo de datos. En TimescaleDB, cada partición se almacena en su propia tabla, a diferencia de PostgreSQL que tiene una sola tabla. Además, TimescaleDB tiene la capacidad de automatizar la creación y eliminación de particiones, lo que permite el almacenamiento eficiente y a largo plazo de los datos de la serie de tiempo.

Capítulo 4

Caso práctico de estudio de una base de datos de series temporales

4.1 Introducción

En este capítulo se hará un caso práctico de puesta en marcha de las bases de datos orientadas en series temporales, estudiadas en el Capítulo 3. De manera que se hará una guía de instalación de ambas bases de datos, se comprobará cómo se almacenan y se accede a los datos, así como formas de visualización y ejemplos sobre cómo analizar las series temporales que manejemos.

4.2 Guía de instalación del sistema

En este apartado se hará una guía por pasos sobre la instalación del sistema InfluxDB.

Como primero de todo hay que comentar que para la realización de esta parte práctica para la puesta en marcha de InfluxDB, y comprobar el funcionamiento de los distintos algoritmos ARIMA, se ha creado una máquina virtual con sistema operativo Ubuntu, versión 22.04.

4.2.1 Instalación de InfluxDB

1. Para instalar InfluxDB en nuestra máquina virtual debemos añadir el repositorio de la web de InfluxData a nuestro sistema con el siguiente comando:

```
system@system-VirtualBox:~$ wget -q https://repos.influxdata.com/influxdata-archive_compat.key
system@system-VirtualBox:~$ echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c in
fluxdata-archive_compat.key' | sha256sum -c && cat influxdata-archive_compat.key | gpg --dearmor | su
do tee /etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg > /dev/null
influxdata-archive_compat.key: La suma coincide
system@system-VirtualBox:~$ echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg
] https://repos.influxdata.com/debian stable main' | sudo tee /etc/apt/sources.list.d/influxdata.list
deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdata.com/deb
ian stable main
```

2. Seguidamente actualizamos nuestra librería de paquetes con el siguiente comando:

```
system@system-VirtualBox:~$ sudo apt-get update
```

3. Y ya podremos empezar a instalar el sistema de InfluxDB, desde el terminal, con el siguiente comando:

```
system@system-VirtualBox:~$ sudo apt-get install influxdb2
```

En este caso hemos instalado la última versión de InfluxDB, lo cual se puede comprobar con el siguiente comando:

```
system@system-VirtualBox:~$ influxd version
InfluxDB v2.7.1 (git: 407fa622e9) build_date: 2023-04-28T13:24:27Z
```

4. Una vez instalado iniciaremos el servicio de InfluxDB con el siguiente comando, para iniciar el servicio:

```
system@system-VirtualBox:~$ sudo service influxdb start
```

5. Para comprobar que se ha iniciado el servicio correctamente haremos uso del siguiente comando, para saber el estatus del servicio:

```
system@system-VirtualBox:~$ sudo systemctl status influxdb
```

Con el cuál, si todo funciona correctamente, podremos ver el siguiente resultado:

```
system@system-VirtualBox:~$ sudo systemctl status influxdb
● influxdb.service - InfluxDB is an open-source, distributed, time series database
     Loaded: loaded (/lib/systemd/system/influxdb.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2023-05-13 13:09:23 CEST; 1min 19s ago
       Docs: https://docs.influxdata.com/influxdb/
    Process: 6954 ExecStart=/usr/lib/influxdb/scripts/influxd-systemd-start.sh (code=exited, status=>
   Main PID: 6955 (influxd)
      Tasks: 15 (limit: 12328)
     Memory: 45.7M
        CPU: 3.949s
     CGroup: /system.slice/influxdb.service
             └─6955 /usr/bin/influxd
```

6. Opcionalmente, para dejar habilitado el servicio, cada vez que iniciemos la máquina, podremos poner el siguiente comando:

```
system@system-VirtualBox:~$ systemctl enable influxdb
Synchronizing state of influxdb.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable influxdb
```

7. Ahora debemos poder iniciar InfluxDB, desde la consola web, o por CLI (Command-Line Interface o interfaz de línea de comandos), en este caso lo haremos por la web.
   Para iniciar la sesión por web deberemos abrir nuestro navegador, en este caso se ha usado el navegador Firefox, y en el buscador navegar a la siguiente dirección: http://localhost:8086. Ya que InfluxDB hará uso del puerto 8086.
   Iniciamos la sesión del usuario, y nos aparecerá el siguiente cuadro, con información a completar, sobre nuestro usuario, contraseña y nombres:

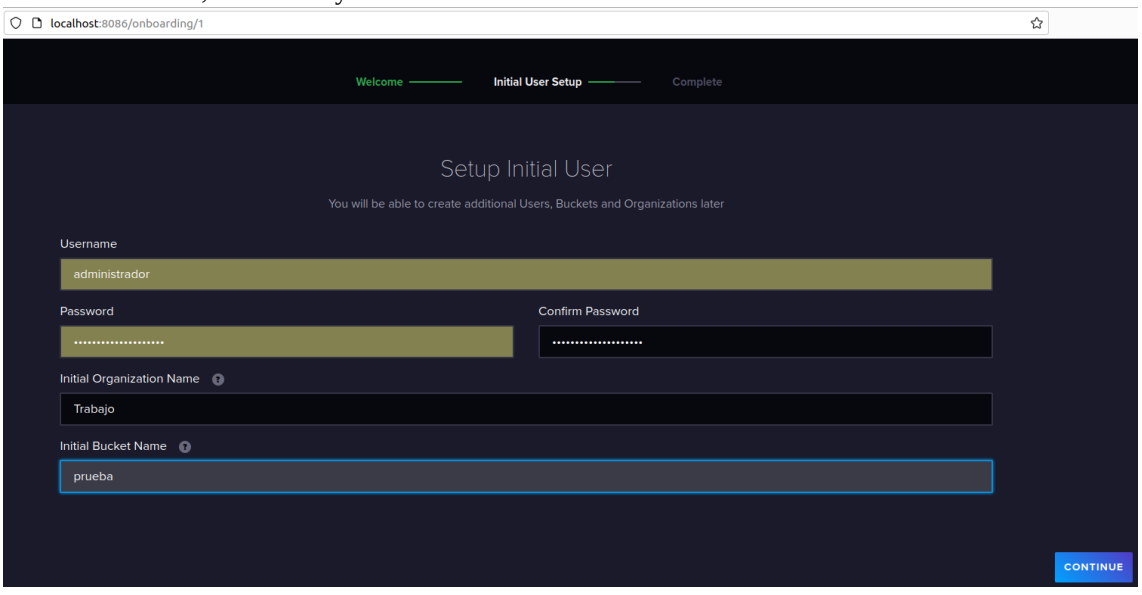

Figura 4. 1: Captura de registro en InfluxDB

4.2.2 Configuración para la conexión a InfluxDB desde la terminal

En este apartado se explicarán brevemente los pasos a seguir para conectarnos y tener permisos para acceder a InfluxDB, desde la terminal de Ubuntu.

1. Ejecutamos los siguientes comandos para descargar la carpeta comprimida de InfluxDB, una vez descargada la descomprimimos, y así podremos iniciar la configuración:

```
system@system-VirtualBox:~/Documentos/InfluxDB$ wget https://dl.influxdata.com/influxdb/releases/infl
uxdb2-client-2.3.0-linux-amd64.tar.gz
```

```
system@system-VirtualBox:~/Documentos/InfluxDB$ tar xvzf influxdb2-client-2.3.0-linux-amd64.tar.gz
```

2. Para inicializar el cliente, tal y como nos indican en la interfaz web, deberemos escribir el siguiente comando en la terminal, cambiando los valores de token, organización y url, que se ajusten a nuestro caso.

   Como recomendación debemos crear un token específico para nuestra base de datos al cual damos los permisos que creamos necesarios para que el usuario que esté haciendo uso de la base de datos pueda o no modificar los datos. En este caso se creará un token con todos los permisos de lectura y escritura. La generación del token es única y personal para cada caso.

```
influx config create --config-name onboarding \
    --host-url "http://localhost:8086" \
    --org "ea99a57bd8d40873" \
    --token "qJPNEsC01Psc8OPWebr2Hmf3wC95VYa74Y390fO6mLHv8-klx7fTjOhhFaC33f879ym9PY83JxQ9nHu3JsHSwg==" \
    --active
```

3. Seguidamente crearemos los buckets que nos sirven para guardar los datos que almacenamos, y desde los cuales podremos consultar los datos que tengamos almacenados en cada bucket. Esto se puede hace fácilmente desde la interfaz web, y creando un bucket con el nombre que queramos:

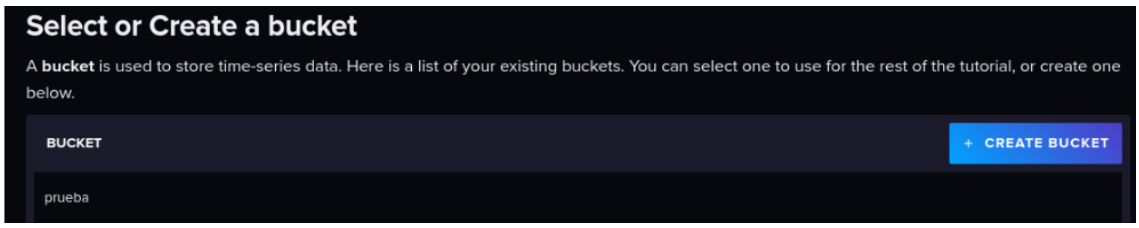

Figura 4. 2: Captura ejemplo creación de bucket en InfluxDB

4. Luego para cargar los datasets que tengamos desde la interfaz web elegiremos el bucket donde queremos subir los datos, y desde allí podremos cargarlos directamente, como en la siguiente imagen:

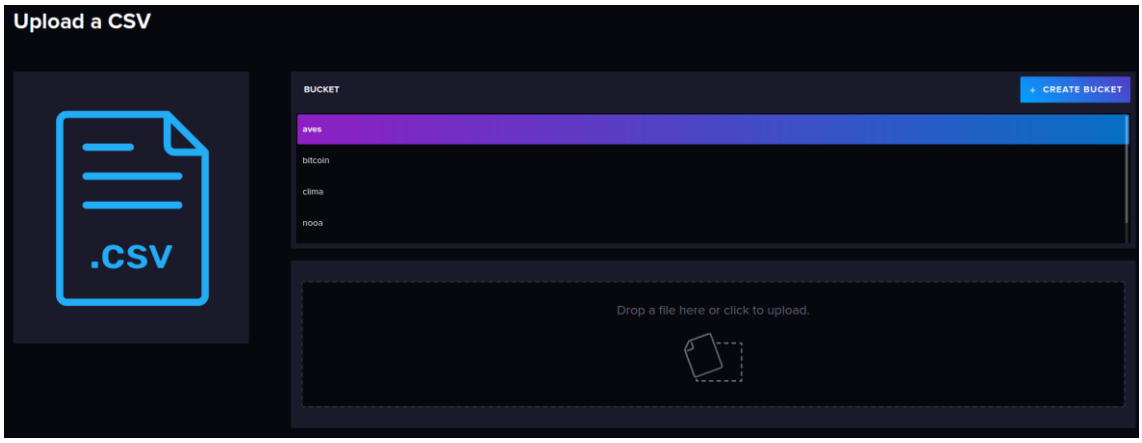

Figura 4. 3: Captura ejemplo cargar csv en un bucket

Simplemente tendremos que arrastrar el conjunto de datos que queramos subir para cargarlo en la base de datos.

Es importante recalcar que los datos han de estar en formato 'csv' y deben estar escritos con la notación específica de InfluxDB para poder subirlos.

4.2.3 Instalación de TimescaleDB

1. Para poder usar TimescaleDB primero debemos instalar PostgreSQL, ya que como se explicó en capítulos anteriores, este sistema es una extensión de PostgreSQL. Para ello entramos como usuario root, y ejecutaremos el siguiente comando:

```
root@system-VirtualBox:~# apt install gnupg postgresql-common apt-transport-https lsb-release wget
```

Seguidamente ejecutamos el script de configuración del repositorio:

```
root@system-VirtualBox:~# /usr/share/postgresql-common/pgdg/apt.postgresql.org.sh
```

2. Ahora podremos agregar el repositorio de TimescaleDB:

```
root@system-VirtualBox:~# echo "deb https://packagecloud.io/timescale/timescaledb/ubuntu/ $(lsb_relea
se -c -s) main" | sudo tee /etc/apt/sources.list.d/timescaledb.list
```

3. Finalmente, para terminar de instalar la extensión debemos ejecutar el siguiente script:

```
root@system-VirtualBox:~# apt install timescaledb-2-postgresql-14
```

4.2.4 Configuración para la conexión TimescaleDB a PostgreSQL desde la terminal

Ahora deberemos conectarnos desde la terminal al cliente de PostgreSQL para utilizar la extensión de TimescaleDB, para ello instalaremos el paquete del cliente con el siguiente comando:

```
root@system-VirtualBox:~# apt-get install postgresql-client
```

Y seguidamente reiniciar el proceso de PostgreSQL, para así iniciar con TimescaleDB.

```
root@system-VirtualBox:~# systemctl restart postgresql
```

1. En este punto accederemos al usuario de postgres como un superusuario, ejecutando el siguiente comando:

```
system@system-VirtualBox:~$ sudo -u postgres psql
```

2. Deberemos crear una contraseña para el usuario postgres, salir de proceso que lo ejecuta, haciendo uso de estos comandos:

```
postgres=# \password postgres
Enter new password for user "postgres":
Enter it again:
postgres=#
\q
```

3. Ahora volvemos a conectar el cliente con PostgreSQL con:

```
system@system-VirtualBox:~$ psql -U postgres -h localhost
```

4. Una vez habiendo accedido con la contraseña creada en el paso anterior, podremos crear desde el terminal de PostgreSQL nuestras bases de datos con comandos SQL:

```
postgres=# CREATE database tsdb;
CREATE DATABASE
```

5. Una vez creadas nos podremos conectar a nuestra base de datos, en este caso llamada 'tsdb', con el siguiente comando, que nos dará el siguiente resultado:

```
postgres=# \c tsdb
psql (15.3 (Ubuntu 15.3-1.pgdg22.04+1), servidor 14.8 (Ubuntu 14.8-1.pgdg22.04+1))
Conexión SSL (protocolo: TLSv1.3, cifrado: TLS_AES_256_GCM_SHA384, compresión: desactivado)
Ahora está conectado a la base de datos «tsdb» con el usuario «postgres».
```

Confirmándose que hemos establecido conexión con la base de datos.
Después de haber creado la base de datos, ya nos podremos conectar directamente accediendo con el usuario desde el terminal, con el comando:

```
system@system-VirtualBox:~$ psql -U postgres -h localhost -d tsdb
```

6. En este paso crearemos la extensión de TimescaleDB dentro de la base de datos con el siguiente comando:

```
tsdb=# CREATE EXTENSION IF NOT EXISTS timescaledb;
```

Que si se crea correctamente deberá devolvernos el siguiente resultado:

```
tsdb=# CREATE EXTENSION IF NOT EXISTS timescaledb;
WARNING:
WELCOME TO
 _____ _                               _     ____________
|_   _(_)                             | |   |  _  \ ___ \
  | |  _ _ __ ___   ___  ___ ___ __ _| | ___| | | | |_/ /
  | | | | '_ ` _ \ / _ \/ __|/ __/ _` | |/ _ \ | | | ___ \
  | | | | | | | | |  __/\__ \ (_| (_| | |  __/ |/ /| |_/ /
  |_| |_|_| |_| |_|\___||___/\___\__,_|_|\___|___/ \____/
               Running version 2.11.1
For more information on TimescaleDB, please visit the following links:

 1. Getting started: https://docs.timescale.com/timescaledb/latest/getting-started
 2. API reference documentation: https://docs.timescale.com/api/latest
 3. How TimescaleDB is designed: https://docs.timescale.com/timescaledb/latest/overview/core-concepts

Note: TimescaleDB collects anonymous reports to better understand and assist our users.
For more information and how to disable, please see our docs https://docs.timescale.com/timescaledb/latest/how-to-guides/configuration/telemetry.

CREATE EXTENSION
```

También podremos comprobar la versión, y que está instalado, usando el comando '\dx', que nos devuelve el siguiente resultado:

```
tsdb=# \dx
                                    Listado de extensiones instaladas
   Nombre    | Versión |  Esquema   |                                Descripción
-------------+---------+------------+---------------------------------------------------------------------------
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
 timescaledb | 2.11.1  | public     | Enables scalable inserts and complex queries for time-series data (Community Edition)
(2 filas)
```

Para crear la tabla de datos necesaria para hacer la prueba ponemos el siguiente código:

```
CREATE TABLE clima(
    fecha       TIMESTAMPTZ       NOT NULL,
    estacion    TEXT              NOT NULL,
    temperatura DOUBLE PRECISION  NULL,
    humedad     DOUBLE PRECISION  NULL,
    precipitacion DOUBLE PRECISION NULL
);
```

Este paso es necesario para poder posteriormente insertar los datos del dataset desde la terminal.

4.2.5 Instalación de Grafana

Grafana es una herramienta de visualización de datos open source y gratuita que se utiliza para visualizar datos de diversas fuentes en una interfaz web. Con Grafana, se pueden crear paneles personalizados para mostrar estadísticas de datos en tiempo real o históricas en forma de gráficos, tablas o mapas. También se puede interactuar con los paneles de Grafana para explorar los datos desde diferentes perspectivas.

Tanto InfluxDB como TimescaleDB se pueden utilizar como fuentes de datos para Grafana.

Entre las características más destacadas de Grafana se encuentra la capacidad de mostrar datos en tiempo real, lo que es muy útil para monitorear sistemas y servicios críticos. Además, ofrece una amplia variedad de opciones de visualización personalizable, lo que permite a los usuarios crear gráficos, tablas y paneles personalizados de acuerdo a las necesidades específicas del usuario.

Para la instalación de grafana desde su página web, se descarga la versión Open Source (OSS), con los siguientes comandos:

```
system@system-VirtualBox:~$ sudo apt-get install -y adduser libfontconfig1
```

El primer comando es una herramienta específica para sistemas basados en Debian o Ubuntu. Este comando instala automáticamente el usuario "adduser" y la librería de fuentes "libfontconfig1".

```
system@system-VirtualBox:~$ wget https://dl.grafana.com/oss/release/grafana_10.0.1_amd64.deb
```

El segundo comando descarga el paquete de instalación de Grafana directamente desde el sitio web oficial de Grafana.

```
system@system-VirtualBox:~$ sudo dpkg -i grafana_10.0.1_amd64.deb
```

El tercer comando es utilizado para instalar el paquete de Grafana descargado en el paso anterior. Este comando desempaqueta y configura la aplicación dentro del sistema.

Para empezar a ejecutar Grafana en nuestro terminal usamos el siguiente comando:

```
system@system-VirtualBox:~$ sudo /bin/systemctl start grafana-server
```

Para ver el estado del servicio, tendremos que poner el mismo comando de 'status', de forma que la respuesta del servicio, en caso de estar activa debería ser la siguiente:

```
system@system-VirtualBox:~$ sudo systemctl status grafana-server
● grafana-server.service - Grafana instance
     Loaded: loaded (/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
     Active: active (running) since Fri 2023-07-07 21:33:23 CEST; 5min ago
       Docs: http://docs.grafana.org
   Main PID: 6683 (grafana)
      Tasks: 15 (limit: 12314)
     Memory: 64.5M
        CPU: 10.203s
     CGroup: /system.slice/grafana-server.service
             └─6683 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfil>

jul 07 21:33:31 system-VirtualBox grafana[6683]: logger=provisioning.alerting t=2023-07-07T21:33:31.>
jul 07 21:33:31 system-VirtualBox grafana[6683]: logger=modules t=2023-07-07T21:33:31.986045581+02:0>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=grafanaStorageLogger t=2023-07-07T21:33:31.9>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=ngalert.state.manager t=2023-07-07T21:33:32.>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=http.server t=2023-07-07T21:33:31.999654276+>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=ngalert.state.manager t=2023-07-07T21:33:32.>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=ticker t=2023-07-07T21:33:32.171464821+02:00>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=ngalert.multiorg.alertmanager t=2023-07-07T2>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=plugins.update.checker t=2023-07-07T21:33:32>
jul 07 21:33:32 system-VirtualBox grafana[6683]: logger=grafana.update.checker t=2023-07-07T21:33:32>
lines 1-21/21 (END)...skipping...
```

Ahora sólo tendremos que acceder desde el puerto 3000 a Grafana, insertando la dirección desde un navegador: 'http://localhost:3000'

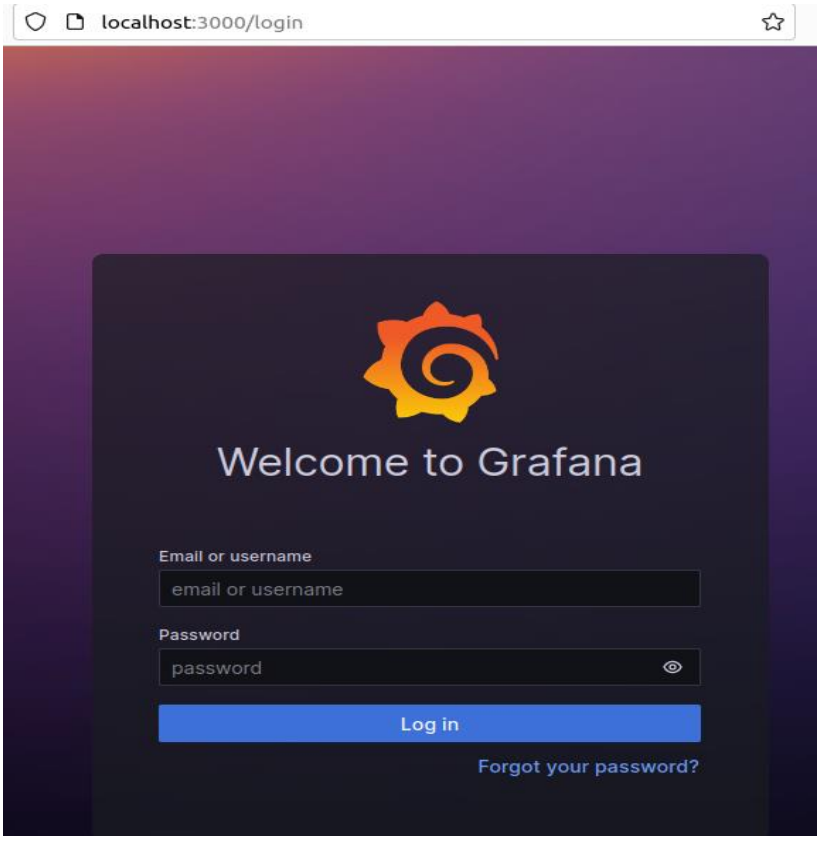

Figura 4. 4: Captura registro en Grafana

En este paso iniciaremos sesión en Grafana, y ya podremos añadir las herramientas que usemos dentro de la interfaz en el apartado 'Data Source'.

- En el caso de TimescaleDB deberemos añadir PostgreSQL, porque como ya se ha explicado TimescaleDB es una extensión del propio Postgres.
- En caso de usar InfluxDB, este ya tiene su propio 'data source' el cual añadir a la herramienta para luego conectar con la base de datos.

4.3 Visualización de los datos de series temporales

Tanto desde la interfaz de InfluxDB, como en la aplicación de Grafana. Entre las formas más útiles de visualización se encuentran:

- Visualización de datos directamente mediante una tabla: para ver los valores numéricos, campos y estructura de nuestra serie temporal de manera sencilla, dividiendo los tiempos, fields y measurements por columnas:

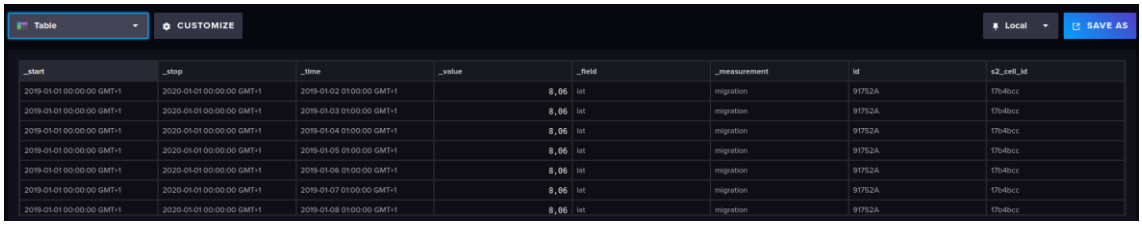

Figura 4. 5: Serie temporal en forma de tabla numérica

- Visualización mediante un histograma: es útil cuando se quiere tener una idea de la distribución de los datos a lo largo del tiempo. El histograma es un gráfico de barras en el que se muestran las frecuencias de diferentes valores de una variable en un intervalo de tiempo determinado. Cada barra del histograma representa una clase o intervalo, y la altura de la barra representa la frecuencia o número de veces que los datos caen dentro de ese intervalo. Además, a través del histograma, es posible examinar la dispersión y concentración de valores en diferentes momentos del tiempo.

Figura 4. 6: Serie temporal en forma de histograma

- Visualización mediante un mapa de calor: Los mapas de calor son gráficos que representan la serie temporal en una matriz de colores en la que se asigna a cada punto de datos un valor de color en función de su magnitud o frecuencia. En el contexto de una serie temporal, los mapas de calor pueden utilizarse para resumir la información en diferentes intervalos de tiempo y mostrar la relación de los valores de la serie con otras variables, como la temporada, la hora del día o el día de la semana.

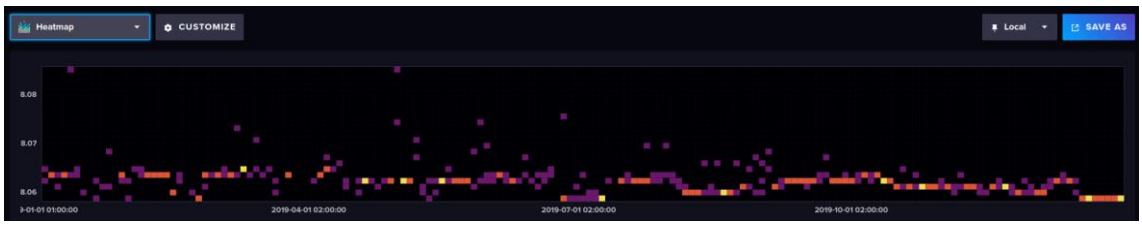

Figura 4. 7: Serie temporal en forma de mapa de calor

- Visualización mediante un gráfico lineal: la variable se traza en el eje vertical, mientras que el tiempo se traza en el eje horizontal. Es una forma efectiva de ver la evolución de la variable a lo largo del tiempo y también para comparar múltiples variables en un solo gráfico. El gráfico lineal permite detectar patrones como tendencias, picos y valles en la serie temporal. Es útil para realizar comparaciones entre diferentes variables y también para hacer proyecciones a futuro. Además, con la ayuda de líneas de tendencia, es posible identificar la dirección y la fuerza de la tendencia a lo largo del tiempo.

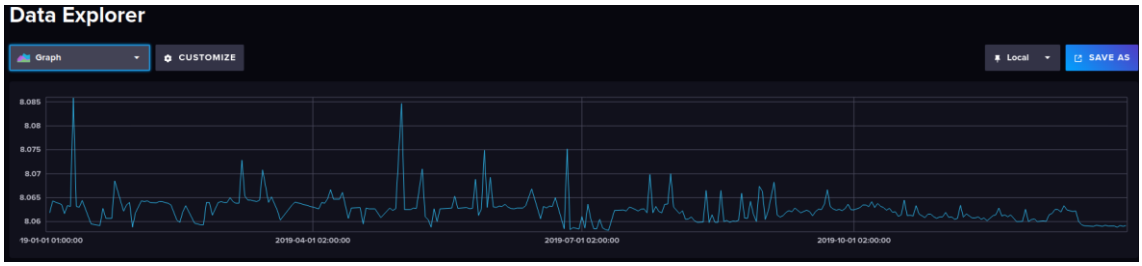

Figura 4. 8: Serie temporal en forma de gráfico lineal

- Visualización mediante un diagrama de dispersión: cada punto representa un par de valores de dos variables diferentes. Un eje horizontal representa una variable y el eje vertical representa la otra. Si se ha recolectado data de dos variables relacionadas en dos momentos diferentes del tiempo, se pueden representar en un gráfico de dispersión para observar la relación entre ellas. Si la relación es fuerte, los puntos se agruparán en una línea. Si la relación es débil o inexistente, los puntos estarán más dispersos. Observar cómo se relacionan dos variables puede ser útil en situaciones donde se esté interesado en el impacto de un cambio en una variable en otra como la relación entre la publicidad y las ventas de un producto.

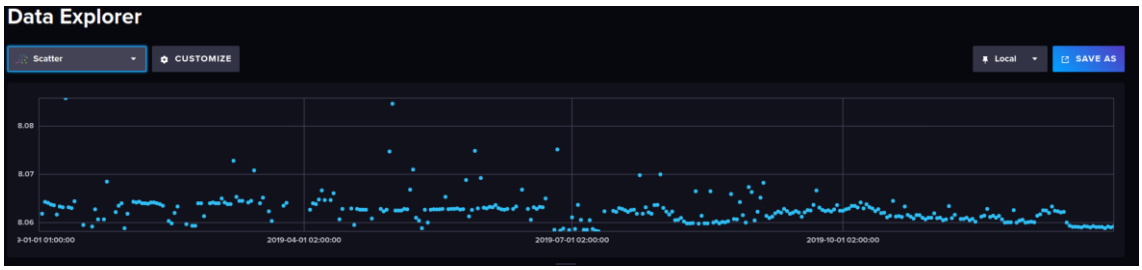

Figura 4. 9: Serie temporal en forma de diagrama de dispersión

4.4 Dataset utilizados en el trabajo

Para la realización de las pruebas comparativas de datos de series temporales en este proyecto, se han creado una serie de scripts en Python, para poder tratar el mismo tipo de datos, y la misma cantidad en ambos sistemas de bases de datos.

Estos scripts sirven para generar datos de manera aleatoria, con ciertas limitaciones en rango para que se asemejen dentro de lo posible a los rangos de valores del mundo real. En este caso se generará un dataset de datos climáticos. Estos dataset dispone de los siguientes campos:

- Fecha: definida como 'año-mes-día-hora', representaría la fecha donde es recogido ese dato del clima.
- Estación: sirve como delimitador para cambiar los rangos de valores que podemos obtener según la estación en la que nos encontremos durante el año. Por ejemplo la estación 'invierno' abarca desde el mes 12 (diciembre) con día 31, hasta el mes 3 (marzo) con día 20, y durante este mes los rangos de temperatura que se pueden conseguir son entre (-5 y 15) grados Celsius.
- Temperatura: representaría un valor medio de la temperatura medida a una hora en concreto de un día en concreto, simulando estar en unidades de grados Celsius.
- Humedad: representaría el valor promedio, medido como porcentaje de la humedad del aire a una hora de un día en concreto.
- Precipitaciones: representaría el valor promedio de precipitaciones medidas en mililitros, que se han producido en una fecha en concreto.

Este dataset rudimentario nos sirve para posteriormente comparar el funcionamiento de cada base de datos, midiendo los tiempos de inserción, lectura y acceso. Para realizar estas pruebas se crearán varios miles de registros, para pronunciar así cualquier cambio significativo a la hora de calcular los rendimientos de cada base de datos.

Una vez que se ha creado este dataset en formato '.csv' se han ejecutado los scripts desde la terminal para cargarlos directamente en la base de datos que corresponda.

Para insertar los datos crearemos una tabla en TimescaleDB que cubra estos campos, con el siguiente código:

```
tsdb=# CREATE TABLE clima(
        fecha    TIMESTAMPTZ      NOT NULL,
        estacion         TEXT     NOT NULL,
        temperatura      DOUBLE PRECISION        NULL,
        humedad DOUBLE PRECISION          NULL,
        precipitacion    DOUBLE PRECISION        NULL
);
CREATE TABLE
```

Además, se ha creado una base de datos dentro de PostgreSQL, llamada 'basepostgresql', donde no se ha añadido la extensión de TimescaleDB, para comparar el rendimiento y la diferencia que hay con la extensión instalada. Dentro de esta base se crea el mismo tipo de tabla.

Y en InfluxDB creamos directamente desde la web un bucket al que llamamos clima, donde se cargarán estos datos:

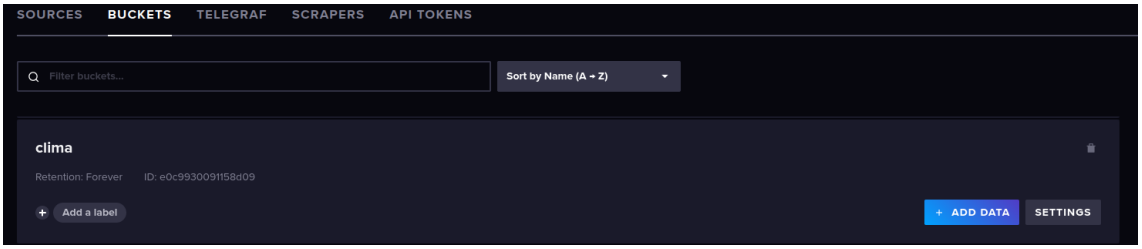

Figura 4. 10: Captura ejemplo carga de datos en InfluxDB

Ahora ya podremos cargar los datos en cada sistema desde la terminal del equipo.

4.5 Operaciones y comandos básicos de los sistemas para manejar datos

4.5.1 Operaciones y comandos de lectura

InfluxDB y TimescaleDB son dos bases de datos distintas que utilizan diferentes lenguajes de consulta para recuperar datos. En el caso de InfluxDB, se utiliza el lenguaje Flux mientras que en TimescaleDB se utiliza SQL. Flux es un lenguaje de consulta de alto nivel diseñado específicamente para trabajar con datos de series temporales. Por otra parte, SQL es un lenguaje de consulta más ampliamente utilizado en las bases de datos relacionales. SQL es capaz de hacer consultas avanzadas, pero no está optimizado específicamente para trabajar con datos de series temporales.

Para recuperar datos en InfluxDB, se utiliza la sentencia from() para especificar el origen de los datos, seguida de la función de agregación range() para especificar el rango de tiempo de los datos solicitados. A continuación, se utiliza filter() para reducir el conjunto de datos de acuerdo con ciertos criterios de filtrado. Por último, se utiliza aggregateWindow() para agrupar los datos y utilizar funciones de agregación como mean(), count(), sum().

En el caso de TimescaleDB, se puede utilizar la sintaxis SQL para recuperar los datos. La sentencia SELECT es la principal utilizada para recuperar datos de una tabla en TimescaleDB. Se utilizan cláusulas como WHERE para establecer criterios de filtrado, GROUP BY para agrupar el conjunto de datos, y funciones de agregación como AVG(), SUM(), COUNT(), entre otras.

Algunos ejemplos de comandos Flux para la lectura de datos:

```
```
from(bucket: "clima")
            |> range(start: 2018-01-01T00:00:00Z, stop: 2023-01-
01T00:00:00Z)
            |> filter(fn: (r) => r["_measurement"] == "Datos clima")
            |> filter(fn: (r) => r["_field"] == "humedad" or r["_field"]
== "precipitacion" or r["_field"] ==
"temperatura")
            |> filter(fn: (r) => r["tag1"] == "valor1")
            |> aggregateWindow(every: 1h, fn: max, createEmpty: false)
```
```

Algunos ejemplos de comandos SQL para la lectura de datos son:

```
```
SELECT time, temperature, location
FROM conditions
WHERE time > NOW() - INTERVAL '1 hour'
```
```

4.5.2 Operaciones de escritura y borrado de datos

La escritura de los datos en InfluxDB se realiza mediante el comando "influx write". Este comando permite escribir datos en InfluxDB mediante una variedad de protocolos, como HTTP, UDP o la API de InfluxDB. Para escribir los datos en InfluxDB, se utiliza un protocolo de línea de comando que permite especificar el tiempo de la escritura, el nombre de la serie y los campos de datos. Para escribir datos en InfluxDB mediante HTTP, se debe utilizar un método POST que contenga los datos a escribir en formato JSON. El formato JSON debe contener los campos de datos y el tiempo de la escritura en formato UNIX. Además, se pueden especificar etiquetas que permiten categorizar los datos en función de su origen.

Por otro lado, el comando "influx delete" permite borrar datos almacenados en InfluxDB. Este comando se utiliza para eliminar datos que ya no son necesarios y para reducir el tamaño de la base de datos. El comando "influx delete" se puede utilizar para borrar puntos de datos individuales, series completas o incluso bases de datos enteras. Para borrar datos de InfluxDB, se utiliza una consulta que especifica la serie y los datos que se deben eliminar. Es posible especificar múltiples condiciones en la consulta para borrar datos seleccionados en función de etiquetas y campos específicos. También es posible especificar el rango de tiempo para borrar datos en función del tiempo de escritura.
Algunos ejemplos de comandos Flux para escribir datos son:

```
```
influx write \
  -b clima \
  -o orgName \
  -p s \
  'temp_and_humid,location=myLocation
temperature=68,humidity=0.8,precipitation=0.2'
```
```

Algunos ejemplos de comandos Flux para borrar datos:

```
```
influx delete --bucket clima \
  --start '1970-01-01T00:00:00Z' \
  --stop $(date +"%Y-%m-%dT%H:%M:%SZ") \
  --predicate '_measurement="temperature"'
```
```

De otra forma en TimescaleDB debemos de haber creado una tabla con anterioridad, donde se pueden insertar nuevos datos utilizando una consulta INSERT INTO. La consulta debe especificar el nombre de la tabla y los valores de la columna temporal y las columnas de datos.

Para borrar datos en TimescaleDB, se utiliza una consulta DELETE que especifica la tabla y los datos que se deben eliminar. Es posible especificar múltiples condiciones en la consulta para borrar datos seleccionados en función de etiquetas y campos específicos. También es posible especificar el rango de tiempo para borrar datos en función del tiempo de escritura.

Ejemplos de comandos SQL para insertar datos son:

```
```
INSERT INTO clima (fecha, temperatura, humedad) VALUES ('2023-07-24
12:10:00', 26.3, 74.1);
INSERT INTO clima (fecha, temperatura, humedad) VALUES ('2023-07-24
12:15:00', 26.8, 71.9);
```
```

Como ejemplo de borrado de datos:

```
```
DELETE FROM clima WHERE temperatura > 30 AND fecha < '2022-07-24
12:05:00';
```
```

4.5.3 Rendimiento de las operaciones de inserción y lectura sobre el dataset

En este apartado compararemos los rendimientos calculados a la hora de realizar estas operaciones en ambas bases de datos, cogiendo un tiempo inicial desde que se inicia la ejecución de la operación y un tiempo final hasta que finaliza esta ejecución, aportando capturas de estos tiempos para cada operación. Así se puede tener una base con código en Python para realizar operaciones de comparación.

1. Tiempo de carga de datos en los sistemas: para insertar los datos en ambos sistemas se ha conseguido con scripts en Python de lectura, y carga de filas desde un archivo csv, hacia la conexión con la base de datos, ejecutando una consulta que inserte los datos en la tabla. En esta primera prueba se hará con 43825 filas de datos a un dato recogido por hora, del 01/01/2018 hasta el 01/01/2023:
    a. InfluxDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de inserción de los datos desde la terminal en InfluxDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de inserción de los datos desde la terminal en InfluxDB fue de:  4.920135021209717  segundos.
```

    b. TimescaleDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de inserción de los datos desde la terminal en TimescaleDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de inserción de los datos desde la terminal en TimescaleDB fue de:  15.2774338722229  segundos.
```

Como ya se ha mencionado ambos sistemas cargan la misma base de datos, sin embargo, el tiempo de ejecución en la inserción de la tabla desde la terminal haciendo uso de scripts en Python es muy diferente, ya que en el caso de InfluxDB (4.9 segundos), a diferencia de TimescaleDB (15.27 segundos). Los scripts tienen la misma estructura de conexión y lectura del fichero, por lo que la única diferencia que puede explicar este cambio en el tiempo de ejecución es en la propia estructura y petición de la consulta, de forma que la consulta de InfluxDB está peor optimizada para insertar datos.

Esta prueba de inserción es la estándar, también se realizarán pruebas con un número mucho más reducido de datos y grandes cantidades de hasta 1000000 de filas de datos. En la inserción de los scripts de las bases de datos se crea una lista de objetos y agregamos cada punto a la lista a medida que leemos los datos del archivo CSV. Cuando la lista alcanza un determinado tamaño de puntos, escribimos los puntos en la base de datos para realizar estas pruebas y que no haya errores en el tiempo de respuesta del servidor se ha definido este número de puntos a 10000.

Las pruebas del rendimiento de inserción para cada número de filas han dado los siguientes resultados:

- 8761 filas de datos
    - InfluxDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de inserción de los datos desde la terminal en InfluxDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de inserción de los datos desde la terminal en InfluxDB fue de:  1.364267349243164  segundos.
```

    - TimescaleDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de inserción de los datos desde la terminal en TimescaleDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de inserción de los datos desde la terminal en TimescaleDB fue de:  2.878892421722412  segundos.
```

- 525961 filas de datos
    - InfluxDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de inserción de los datos desde la terminal en InfluxDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de inserción de los datos desde la terminal en InfluxDB fue de:  71.80228686332703  segundos.
```

    - TimescaleDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de inserción de los datos desde la terminal en TimescaleDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de inserción de los datos desde la terminal en TimescaleDB fue de:  140.8262288570404  segundos.
```

En todos los casos se puede apreciar que TimescaleDB tiene un mayor tiempo de inserción, de mínimo el doble en cualquier tipo de operación empezando por la tabla más pequeña.

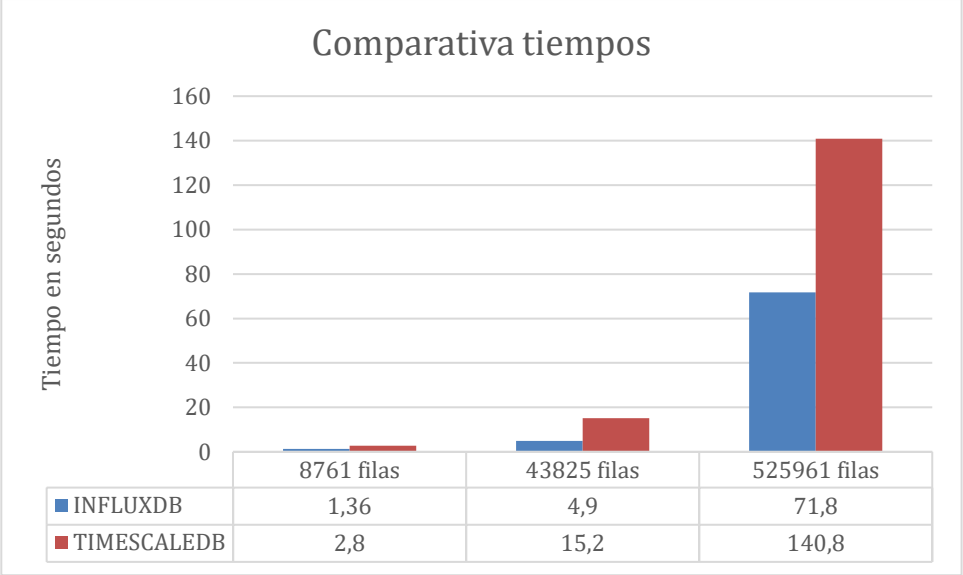

Tabla 4. 1: Diagrama de tiempos de inserción

2. Tiempo de búsqueda de los en los sistemas: para este ejercicio, se hará una consulta en las bases de datos con un script en Python donde queremos extraer todos los datos de la temperatura, y mostrarlos por pantalla.
    a. InfluxDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print("El tiempo de busqueda de los datos desde la terminal en InfluxDB fue de: ", tiempo_ejecucion, " segundos.")
El tiempo de busqueda de los datos desde la terminal en InfluxDB fue de:  6.850727796554565  segundos.
```

    b. TimescaleDB:

```
>>> # Imprimimos el tiempo de ejecución
>>> print(f"El tiempo de busqueda de los datos desde la terminal en TimescaleDB fue de: {tiempo_ejecucion} segundos.")
El tiempo de busqueda de los datos desde la terminal en TimescaleDB fue de: 0.07964563369750977 segundos.
```

En esta prueba de lectura vemos que en la base de datos de InfluxDB tiene un retardo mayor, a la hora de consultar los datos. De forma que en este ejercicio de lectura la diferencia está en la optimización de la búsqueda por el lenguaje de consulta, ya que Flux ha sido más lento, mientras que SQL con TimescaleDB ha tenido un mejor rendimiento.

Capítulo 5

Conclusiones y líneas futuras de estudio

5.1 Conclusiones

Durante el desarrollo de este trabajo se ha visto que los datos de las series temporales abarcan diversos campos de aplicación de mucha relevancia en nuestro día a día, como las finanzas, economía o salud, y poder analizar los datos de los que disponemos es muy importante, ya que es la mejor forma de adquirir conocimiento de esta información. Las bases de datos de series temporales son cruciales para poder almacenar estas grandes cantidades de datos de una manera óptima, y luego poder consultarlos de forma eficaz y rápida, de nada nos sirve tener estos datos.

Prestando especial atención sobre cómo funcionan estas bases de datos, y sobre todo los datos que estamos tratando de almacenar o visualizar con estos sistemas, para así luego poder realizar análisis de esta información y adquirir ese conocimiento que nos puede servir para hacer predicciones del comportamiento a futuro y posibles eventos de los campos que estemos tratando. Es por esto que en el presente trabajo se ha realizado un estudio teórico para entender bien los sistemas de bases de datos de series temporales, y las formas que tienen de tratar los datos, así como un análisis de dos tipos diferentes de estas bases de datos, comprobando así las diferencias que hay en tratar los datos con un tipo de lenguaje u otro, y comprobando también las diferencias de rendimiento entre estas base de datos, y cómo mejoran el tratamiento de las series temporales, comparando las mejoras que TimescaleDB proporcionaba a PostgreSQL.

5.2 Posibles líneas futuras de trabajo

Como posibles ampliaciones a este trabajo, se pueden comparar los resultados de rendimiento a la hora de ejecutar operaciones en más bases de datos orientadas a series temporales, y ver las formas de obtención, almacenamiento, lectura y escritura que tienen otros sistemas dentro del campo de las bases de datos relacionales y cómo estos procesos son más o menos efectivos tratando según los tipos de datos que estemos estudiando. Así cómo ampliar la línea de investigación comprobando los tipos de análisis que hay de series temporales, y cuáles son los más efectivos para usar con este tipo de base de datos orientado a este tipo de datos.

Capítulo 6

Bibliografía

[1] EF Codd, "Derivación del modelo relacional para sistemas de gestión de bases de datos relacionales", IBM Systems Journal, vol. 16, núm. 4, págs. 366-377, 1977.

[2] M. Marqués, "Bases de datos," D - Universitat Jaume I. Servei de Comunicació i Publicacions, Castelló de la Plana, España, 2009. Disponible en: https://elibro.net/es/ereader/bibliouah/51645?page=23.

[3] Departamento de Matemáticas. "Series Temporales." [En línea]. Disponible en: http://www.estadistica.mat.uson.mx/Material/seriesdetiempo.pdf

[4] "Series Temporales". Alojamientos - Universidad de Valladolid UVa. [En línea]. Disponible en: http://www5.uva.es/estadmed/datos/series/series.htm.

[5] "Ranking comparativo de bases de datos temporales (2023)," Db-engines. [En línea]. Disponible en: https://db-engines.com/en/ranking/time+series+dbms.

[6] InfluxData. Sitio web oficial (2023). [En línea]. Disponible en: https://www.influxdata.com/.

[7] Python. Sitio web oficial (2023). [En línea]. Disponible en: https://www.python.org/.

[8] P. Beynon-Davies. Sistemas de bases de datos. Reverte, 2018.

[9] Biblioteca de la Universidad de Alcalá. "El impacto de la tecnología en la educación". [En línea]. Disponible en: https://elibro.net/es/ereader/bibliouah/51645.

[10] Universidad de Carlos III de Madrid. "Series Temporales". [En línea]. Disponible en: https://halweb.uc3m.es/esp/Personal/personas/imolina/MiDocencia/SeriesTemporales/Tema2SeriesEstud.pdf.

[11] Universidad de La Laguna. "Almacenamiento y visualización de series temporales". [En línea]. Disponible en: https://riull.ull.es/xmlui/bitstream/handle/915/10417/Almacenamiento%20y%20visualizacion%20de%20series%20temporales.pdf?sequence=1&isAllowed=y.

[12] Muñoz, I. "Sistema de almacenamiento y visualización de series temporales para la monitorización de infraestructuras". [En línea]. Disponible en: https://oa.upm.es/67931/1/TFM_ISMAEL_MUNOZ_AZTOUT.pdf.

[13] Open Source Lab. "Cómo instalar Ubuntu en VirtualBox". [En línea]. Disponible en: https://osl.ugr.es/2020/09/29/como-instalar-ubuntu-en-virtual-box/.

[14] InfluxData, "Install InfluxDB v2.0," InfluxDB v2.0 documentation, 2021. [En línea]. Disponible en: https://docs.influxdata.com/influxdb/v2.0/install/?t=Linux.

[15] DB-Engines, "Time Series DBMS Ranking," DB-Engines Ranking, 2021; disponible en: https://db-engines.com/en/ranking/time+series+dbms.

[16] R. J. Rasilla-Díaz, "Introducción al Análisis de las Series Temporales con Python (Parte 2)", QuantSpace, 2020; disponible en: https://quantspace.es/2020/08/01/analisis-de-series-temporales-con-python-parte-2/.

[17] R. J. Rasilla-Díaz, "Series temporales", Universidad de Cantabria, 2021; disponible en: https://personales.unican.es/rasillad/docencia/G14/tema_4_series_temporales.html#introducci%C3%B3n-a-las-series-temporales.

[18] InfluxData, "Query Sample Data," InfluxDB v2.7 documentation, 2021; disponible en: https://docs.influxdata.com/influxdb/v2.7/query-data/execute-queries/query-sample-data/.

[19] F. Argüello, "Series Temporales en Castellano - Un enfoque clásico", GitHub, 2021; disponible en: https://github.com/FrancisArgnR/SeriesTemporalesEnCastellano#enfoque-cl%C3%A1sico.

[20] L. J. Prieto, "Almacenamiento y Recuperación de la Información. ELIBRO.NET," E-Libro, Biblioteca de la Universidad de Alcalá, 2015, disponible en: https://elibro.net/es/ereader/bibliouah/57439?page=3.

[21] A. Gómez-González, "Evolución de las bases de datos," AdSalsa, 2019, disponible en: https://www.adsalsa.com/evolucion-de-las-bases-de-datos/.

[22] J. Xu et al., "SARS-CoV-2 induces transcriptional signatures in human lung epithelial cells that promote lung fibrosis," Sci. Rep., vol. 11, no. 1, pp. 1-11, 2021, doi: 10.1038/s41598-021-88255-2.

[23] J. Berbel Carballal, "Análisis y mejora del rendimiento en PostgreSQL," Universidade de A Coruña, 2021, disponible en: https://ruc.udc.es/dspace/bitstream/handle/2183/28878/BerbelCarballal_Jorge_TFG_2021.pdf?sequence=3.

[24] G. Buzsáki et al., "TimescaleDB: the open-source time-series database," TimescaleDB, 2021, disponible en: https://docs.timescale.com/.

[25] Timescale. (2021). TimescaleDB Documentation. [Online]. Available: https://docs.timescale.com/

[26] Jindal, A. (2018). TimescaleDB: An open-source database built for analyzing time-series data at scale. [Online]. Available: https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/

[27] InfluxData. (2021). InfluxDB Documentation. [Online]. Available: https://docs.influxdata.com/influxdb/

[28] Jindal, A. (2018). TimescaleDB: An open-source database built for analyzing time-series data at scale. [Online]. Available: https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/

[29] PostgreSQL Global Development Group. (2021). PostgreSQL Documentation. [Online]. Available: https://www.postgresql.org/docs/

[30] PostgreSQL Tutorial. (2021). PostgreSQL vs. MySQL vs. MariaDB: What's the Difference? [Online]. Available: https://www.postgresqltutorial.com/postgresql-vs-mysql/

Apéndice A
Códigos utilizados en este trabajo

A.1 Introducción

En este apartado se mostrarán los códigos que se han utilizado para hacer la comparación de funciones de TimescaleDB e InfluxDB en este trabajo. Todos los scripts se han escrito en Python, y ejecutado desde la terminal de la máquina virtual, haciendo uso del comando 'python3' para iniciar la aplicación e insertar el código. Estos códigos abarcan desde las conexiones con los sistemas de bases de datos e inserción de los mismo, así como los comandos de lectura, escritura, borrado para comprobar su funcionamiento mediante consultas que se enviaban y recibían con la conexión abierta a la base de datos.

A.2 Códigos para generar archivo de datos

```
#Generacion de datos simulados:
import csv
from datetime import datetime, timedelta
import random

# Definir las estaciones del año
SEASONS = {"invierno": ((1, 1), (3, 20)),
           "primavera": ((3, 21), (6, 20)),
           "verano": ((6, 21), (9, 20)),
           "otoño": ((9, 21), (12, 20)),
           "invierno": ((12, 21), (12, 31))}

# Definir los rangos para cada campo durante cada estación del año
SEASON_RANGES = {"invierno": {"temperatura": (-5, 15),
                              "humedad": (50, 90),
                              "precipitacion": (0, 5)},
                 "primavera": {"temperatura": (5, 25),
                               "humedad": (40, 80),
                               "precipitacion": (0, 7)},
                 "verano": {"temperatura": (20, 40),
                            "humedad": (30, 70),
                            "precipitacion": (0, 10)},
                 "otoño": {"temperatura": (5, 20),
                           "humedad": (50, 90),
                           "precipitacion": (0, 8)}}

start_date = datetime(2018, 1, 1, 0)  # Hora de inicio: 00:00)
end_date = datetime(2023, 1, 1)

data = []
current_date = start_date
current_season = None
while current_date <= end_date:
    for hour in range(24):  # repeat for each hour of the day
        for season, ((start_month, start_day), (end_month, end_day)) in
SEASONS.items():
            if (start_month == current_date.month and start_day <=
current_date.day) or (end_month == current_date.month and end_day >=
current_date.day):
                current_season = season
                break
            if current_season is None:
```

```
                # Asignar estación invierno por defecto
                current_season = "invierno"
    temperature =
random.randint(SEASON_RANGES[current_season]["temperatura"][0],
SEASON_RANGES[current_season]["temperatura"][1])
    humidity =
random.randint(SEASON_RANGES[current_season]["humedad"][0],
SEASON_RANGES[current_season]["humedad"][1])
    precipitation =
random.randint(SEASON_RANGES[current_season]["precipitacion"][0],
SEASON_RANGES[current_season]["precipitacion"][1])
    data.append((current_date, current_season, temperature, humidity,
precipitation))
    current_date += timedelta(hours=1)  # Actualizar en una hora en
lugar de en un día

campos = ['Fecha', 'Estacion', 'Temperatura', 'Humedad',
'Precipitaciones']
# Exportamos los datos a un archivo CSV
with open('datos-climaticos.csv', mode='w', newline='') as archivo:
    writer = csv.writer(archivo)
    # Escribimos los nombres de los campos en la primera fila
    writer.writerow(campos)
    # Escribimos los datos en las filas restantes
    for fila in data:
        writer.writerow(fila)
```

A.3 Códigos de inserción de datos en los sistemas

- En InfluxDB:

```
# Importamos las librerías necesarias
from influxdb_client import InfluxDBClient, Point, WriteOptions
from influxdb_client.client.write_api import SYNCHRONOUS
import csv
import time

# Definimos las credenciales y la configuración necesaria para conectar
con la base de datos
url="http://localhost:8086"
token="NF7yoowctVTRQIEsBhFeOIDgFcz3liqDiIKwrvFtyh7r8-
YyCxYRbhQmz9fWTKTbY41JUlFAfzz6qQGKRcYzaQ=="
org="Trabajo"
bucket = "clima_grande"

# Creamos una instancia del cliente de InfluxDB
cliente = InfluxDBClient(url=url, token=token, org=org, timeout=1000000)

## Obtenemos una instancia del API de escritura
escritor = cliente.write_api(write_options=SYNCHRONOUS)

quedan = 525961

# Definimos la ruta del archivo CSV que contiene los datos a cargar
ruta_csv = "datos-climaticos_grande.csv"

#iniciamos el contador para ver cuanto tarda en insertar los datos
inicio = time.time()
```

```
# Abrimos el archivo CSV y leemos los datos
with open(ruta_csv, mode='r') as archivo_csv:
    csv_reader = csv.reader(archivo_csv, delimiter=',')
    # Ignoramos la primera fila, ya que contiene los nombres de los
campos
    next(csv_reader)
    # Creamos una lista para almacenar los puntos de datos
    puntos = []
    # Iteramos sobre las filas de datos restantes y los agregamos a la
lista de puntos
    for fila in csv_reader:
        fecha = fila[0]
        estacion = fila[1]
        temperatura = float(fila[2])
        humedad = float(fila[3])
        precipitacion = float(fila[4])
        # Creamos un objeto Point con los datos de la fila
        p = Point("Datos clima").tag("tag1", "valor1").field("estacion",
estacion).field("temperatura", temperatura).field("precipitacion",
precipitacion).field("humedad", humedad).time(fecha)
        # Agregamos el punto a la lista de puntos
        puntos.append(p)
        quedan = quedan-1
        print(quedan)
        # Si la lista de puntos tiene más de 1000 elementos, los
insertamos en la base de datos
        if len(puntos) >= 10000:
            escritor.write(bucket=bucket, org=org, record=puntos)
            puntos = []
    # Insertamos los puntos restantes en la base de datos
    if len(puntos) > 0:
        escritor.write(bucket=bucket, org=org, record=puntos)

# Calculamos el tiempo de ejecución
tiempo_ejecucion = time.time() - inicio

# Imprimimos el tiempo de ejecución
print("El tiempo de inserción de los datos desde la terminal en InfluxDB
fue de: ", tiempo_ejecucion, " segundos.")
```

- En TimescaleDB:

```
# Importamos las librerías necesarias
import psycopg2
import csv
import time

# Establecemos la conexión con la base de datos TimescaleDB
conexion = psycopg2.connect(database="tsdb", user="postgres",
password="123", host="localhost", port="5432")

# Definimos la ruta del archivo CSV que contiene los datos a cargar
ruta_csv = "datos-climaticos.csv"

quedan = 525961

#iniciamos el contador para ver cuanto tarda en insertar los datos
```

```
inicio = time.time()
# Abrimos el archivo CSV y leemos los datos
with open(ruta_csv, mode='r') as archivo_csv:
    csv_reader = csv.reader(archivo_csv, delimiter=',')
    # Ignoramos la primera fila, ya que contiene los nombres de los
campos
    next(csv_reader)
    # Creamos un cursor para realizar la carga masiva de los datos
    cursor = conexion.cursor()
    # Iteramos sobre las filas de datos y los adicionamos a una lista a
bulkear
    bulk_data = []
    for fila in csv_reader:
        # Obtenemos los valores de la fila
        fecha = fila[0]
        estacion = fila[1]
        temperatura = float(fila[2])
        humedad = float(fila[3])
        precipitacion = float(fila[4])
        # Agregamos los valores a la lista de datos a insertar
        bulk_data.append((fecha, estacion, temperatura, humedad,
precipitacion))
        # Si la lista de datos tiene más de 10000 elementos, los
cargamos en la base de datos
        if len(bulk_data) >= 10000:
            # Creamos la consulta SQL para insertar los datos en la
tabla
            consulta = f"INSERT INTO clima_pequeno(fecha, estacion,
temperatura, humedad, precipitacion) VALUES (%s, %s, %s, %s, %s);"
            cursor.executemany(consulta, bulk_data)
            conexion.commit()
            bulk_data.clear()
    # Insertamos los puntos restantes en la base de datos
    if bulk_data:
        consulta = f"INSERT INTO clima_pequeno(fecha, estacion,
temperatura, humedad, precipitacion) VALUES (%s, %s, %s, %s, %s);"
        cursor.executemany(consulta, bulk_data)
        conexion.commit()
        bulk_data.clear()

# Calculamos el tiempo de ejecución
tiempo_ejecucion = time.time() - inicio

# Imprimimos el tiempo de ejecución
print("El tiempo de inserción de los datos desde la terminal en
TimescaleDB fue de: ", tiempo_ejecucion, " segundos.")
```

A.4 Códigos comparativos de funciones en los sistemas

- En InfluxDB:

```
# Importamos las librerías necesarias
from influxdb_client import InfluxDBClient, Point, WriteOptions
from influxdb_client.client.write_api import SYNCHRONOUS
import csv
import time

# Definimos las credenciales y la configuración necesaria para conectar
con la base de datos
url="http://localhost:8086"
token="NF7yoowctVTRQIEsBhFeOIDgFcz3liqDiIKwrvFtyh7r8-
YyCxYRbhQmz9fWTKTbY41JUlFAfzz6qQGKRcYzaQ=="
org="Trabajo"
bucket = "clima"

# Creamos una instancia del cliente de InfluxDB
cliente = InfluxDBClient(url=url, token=token, org=org)

#iniciamos el contador para ver cuanto tarda en insertar los datos
inicio = time.time()


# Crear un objeto query API para realizar la consulta
query_api = cliente.query_api()

# definir la consulta
query = '''from(bucket: "clima")
            |> range(start: 2018-01-01T00:00:00Z, stop: 2023-01-
01T00:00:00Z)
            |> filter(fn: (r) => r["_measurement"] == "Datos clima")
            |> filter(fn: (r) => r["_field"] == "humedad" or r["_field"]
== "precipitacion" or r["_field"] == "temperatura")
            |> filter(fn: (r) => r["tag1"] == "valor1")
            |> aggregateWindow(every: 1h, fn: max, createEmpty:
false)'''

# Ejecutar la consulta
result = query_api.query(query = query, org=org)

# procesar los resultados
max_temperatura = None
max_humedad = None
max_precipitacion = None

for table in result:
    for record in table.records:
        valor = record.get_value()
        field = record.get_field()
        if field == 'temperatura':
            if max_temperatura is None or valor > max_temperatura:
                max_temperatura = valor
        elif field == 'humedad':
            if max_humedad is None or valor > max_humedad:
                max_humedad = valor
        elif field == 'precipitacion':
```

```
            if max_precipitacion is None or valor > max_precipitacion:
                max_precipitacion = valor

# Calculamos el tiempo de ejecución
tiempo_ejecucion = time.time() - inicio

# Imprimimos el tiempo de ejecución
print("El tiempo de busqueda de los datos desde la terminal en InfluxDB
fue de: ", tiempo_ejecucion, " segundos.")


print(f"Máximo de temperatura: {max_temperatura}")
print(f"Máximo de humedad: {max_humedad}")
print(f"Máximo de precipitación: {max_precipitacion}")
```

- En TimescaleDB:

```
# Importamos las librerías necesarias
import psycopg2
import csv
import time

# Establecemos la conexión con la base de datos TimescaleDB
conexion = psycopg2.connect(database="tsdb",
                        user="postgres", password="123",
                        host="localhost", port="5432")

total = 0

# iniciamos el contador para ver cuanto tarda en insertar los datos
inicio = time.time()

# Creamos un cursor para ejecutar la consulta
cur = conexion.cursor()

# Ejecutamos la consulta
query = "SELECT temperatura as temp FROM clima WHERE fecha >= '2020-01-
01' AND fecha < '2023-01-01';"
cur.execute(query)

# Obtenemos los resultados de la consulta
results = cur.fetchall()

# Imprimimos los resultados de la consulta
for result in results:
    temp = result[0]
    total+=1
    #print(f'Temperatura: {temp}')

# Cerramos el cursor y la conexión
cur.close()
conexion.close()
```

```
# Calculamos el tiempo de ejecución
tiempo_ejecucion = time.time() - inicio

# Imprimimos el tiempo de ejecución
print(f"El tiempo de busqueda de los datos desde la terminal en
TimescaleDB fue de: {tiempo_ejecucion} segundos.")
print(f"Se han leido un total de datos ", total)
```

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR