UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CARRERA DE SISTEMAS Y COMPUTACIÓN

Proyecto de Investigación previo a la obtención del título de Ingeniero en Sistemas y Computación.

TRABAJO DE TITULACIÓN

Proyecto de Investigación

ANÁLISIS DE BASES DE DATOS RELACIONALES Y NO RELACIONALES APLICADO AL PROBLEMA DE LA RUTA MÁS CORTA

Presentado por:

Alex Danilo Tuapanta Daquilema

Jhonatan Israel Montenegro Garrido

Tutor:

Ing. Ximena Quintana López, PhD.

Riobamba - Ecuador

2021

VEREDICTO DE LA INVESTIGACIÓN

Los miembros del Tribunal de Graduación del proyecto de investigación de título: **"ANÁLISIS DE BASES DE DATOS RELACIONALES Y NO RELACIONALES APLICADO AL PROBLEMA DE LA RUTA MÁS CORTA"**, presentado por: Alex Danilo Tuapanta Daquilema y Jhonatan Israel Montenegro Garrido, dirigida por: PhD. Ximena Quintana López.

Una vez escuchada la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constatado el cumplimiento de las observaciones realizadas, remite la presente para uso de custodia en la biblioteca de la facultad de Ingeniería de la Universidad Nacional de Chimborazo.

Para constancia de lo expuesto firman:

XIMENA ALEXANDRA QUINTANA LOPEZ
Firmado digitalmente por XIMENA ALEXANDRA QUINTANA LOPEZ
Fecha: 2021.04.09 06:28:40 -05'00'

PhD. Ximena Alexandra Quintana López
Director de Proyecto **Firma**

PhD. Miryan Estela Narváez Vilema
Miembro del Tribunal **Firma**

MsC. Ana Elizabeth Congacha Aushay
Miembro del Tribunal **Firma**

DERECHO DE AUTORÍA

"La responsabilidad del contenido de este proyecto de Graduación corresponde exclusivamente a: Alex Danilo Tuapanta Daquilema y Jhonatan Israel Montenegro Garrido bajo la dirección de la PhD. Ximena Quintana López, y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo"

Alex Danilo Tuapanta Daquilema
060505062-4
Autor del Proyecto de Investigación

Jhonatan Israel Montenegro Garrido
060455038-4
Autor del Proyecto de Investigación

XIMENA ALEXANDRA QUINTANA LOPEZ

Firmado digitalmente por XIMENA ALEXANDRA QUINTANA LOPEZ
Fecha: 2021.03.24 08:38:50 -05'00'

Ing. Ximena Quintana López, PhD.
060355761-2
Tutor del Proyecto de Investigación

AGRADECIMIENTO

Primeramente, agradecemos a nuestro creador por brindarnos salud y vida a nosotros y a nuestras familias. Dar gracias por permitirnos llegar a la culminación de nuestra carrera.

Agradecemos a la Universidad Nacional de Chimborazo por permitirnos ser parte de su comunidad y a nuestros docentes por impartirnos todos sus conocimientos con el afán de formar profesionales éticos con un nivel de conocimiento óptimo.

Finalmente, agradecer a nuestra tutora la Ing. Ximena Quintana por ser un apoyo fundamental en el desarrollo de esta investigación.

Alex Danilo Tuapanta Daquilema

Jhonatan Israel Montenegro Garrido

DEDICATORIA

Este proyecto de investigación va dedicado a mis padres y hermanos que han sido un apoyo incondicional durante mi etapa estudiantil. A mi padre por enseñarme a ser una persona de bien y ser un ejemplo de lucha y trabajo honesto. A mi madrecita por nunca dejarme solo, con su cariño y paciencia guiarme hasta esta etapa de mi vida. Finalmente, a mis hermanos por ser ejemplo de esfuerzo, superación y por enseñarme a nunca rendirme a pesar de las adversidades que encuentre en el trayecto de la vida.

Alex Danilo Tuapanta Daquilema

El presente proyecto de investigación se lo dedico a mis padres y hermanos por ser el pilar fundamental y fuerza de lucha para culminar esta etapa. A mi madre por su infinito amor y paciencia, para levantarme cuando he caído. A mi padre por ser un apoyo incondicional durante toda mi etapa estudiantil y futura, por ayudarme a ser un hombre de bien y nunca rendirme. A mi hermana, que la amo mucho, por siempre cuidar de mí y a mi hermano por ser mi inspiración para lograr mis metas.

Jhonatan Israel Montenegro Garrido

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Años atrás, la tecnología SQL era la principal herramienta de modelado y procesamiento de datos, pero en la actualidad deja muchas falencias y no brinda las prestaciones necesarias para trabajar con datos masivos.

La presente investigación tiene como objetivo comparar el rendimiento del cálculo de la ruta más corta mediante el algoritmo de Dijkstra, en PostgreSQL con la extensión pgRouting que sirve para el análisis de redes y planificación de rutas, mientras tanto Neo4j utiliza el lenguaje declarativo inspirado en SQL de Cypher, mismo que cuenta con el algoritmo de Dijkstra en su núcleo. La comparativa se la realizó utilizando las redes viales de la ciudad de Quito, obtenidos de OpenStreetMap. Neo4j no admite el formato nativo de OpenSreetMap (.osm), por lo cual se utiliza la herramienta adicional OSMnx que es un gestor de paquetes de Python, para la descarga de redes viales de la ciudad de Quito en un formato (.graphml) aceptable para el gestor Neo4j.

Los resultados obtenidos en este estudio muestran una superioridad de rendimiento en los tiempos de respuesta del cálculo de la ruta más corta a favor de Neo4j. A pesar de no ser una base de datos orientada al enrutamiento de redes viales, se desenvuelve con una agilidad muy notoria y eficiente. En sus últimas versiones Neo4j ha ido mejorando su soporte en el tratamiento de datos geoespaciales convirtiéndola en una potente alternativa de solución ante los gestores convencionales.

Palabras Clave: Neo4j, PostgreSQL, Dijkstra, OSMnx, OpenStreetMap.

ABSTRACT

Years ago, SQL technology was the primary data modeling and processing tool. Still, in actuality, it leaves many fallancies and does not provide the necessary capabilities to work with massive data.

The present research compares the shortest path route's calculation using the Dijkstra algorithm in PostgreSQL with the pgRouting extension used for network analysis and route planning. In contrast, Neo4j uses the declarative language inspired SQL by Cypher, which has the Dijkstra algorithm at its core. The comparison was made using the road networks of the city of Quito, obtained from OpenStreetMap. Neo4j does not support the native OpenStreetMap format (.osm), so the additional tool OSMnx is used, a Python package manager, to download road networks in the city of Quito in a (.graphml) format acceptable to its Neo4j manager.

The results obtained in this study show a performance superiority in the response times of calculating the shortest path route in favor Neo4j. Despite not being a database-oriented to road networks' routine, it develops with very noticeable and efficient agility. Neo4j has improved its support in geospatial data processing in its latest versions, making it a powerful solution alternative to conventional managers.

Keywords: Neo4j, PostgreSQL, Dijkstra, OSMnx, OpenStreetMap.

Reviewed by:
Ms.C. Ana Maldonado León
ENGLISH PROFESSOR
C.I. 0601975980

INTRODUCCIÓN

El crecimiento de información ha puesto de manifiesto la necesidad de almacenamiento y procesamiento de grandes cantidades de datos. Por ejemplo, registro de celulares o de equipos inteligentes, imágenes satelitales, registros de páginas web, datos de redes sociales, entre otros. La gran cantidad de datos disponibles conlleva a contar con procedimientos capaces de almacenar, procesar y analizar datos con el fin de generar información y conocimiento. (Camargo, Camargo, & Joyanes, 2015)

Dado que las bases de datos relacionales no satisfacen las necesidades de rendimiento, escalabilidad y flexibilidad que requieren las aplicaciones en la actualidad, las empresas mainstreams han adoptado bases de datos NoSQL para el manejo de la información (Durán et al., 2019). Este tipo de bases de datos se han extendido en diferentes ámbitos del mercado, tanto en contextos académicos como empresariales, esto debido a que los tiempos de respuesta en las consultas son rápidas y eficientes, ganando tiempo y aumentando la productividad de las organizaciones. (Vega et al., 2019)

Las bases de datos NoSQL ofrecen, adaptación a aplicaciones modernas, alto rendimiento y disponibilidad de datos en plataformas de videojuegos, dispositivos móviles y web, cuyos requerimientos demandan que; las bases de datos sean flexibles, de alto rendimiento, escalables y altamente funcionales para facilitar excelentes experiencias de usuario. (Castillo et al., 2017)

Este proyecto de investigación se basará en investigación bibliográfica y cuasiexperimental. Para el análisis de bases de datos PostgreSQL y Neo4j se utilizará la investigación cuasiexperimental, con este tipo de investigación se determinará cuáles son los tiempos de respuesta al momento de resolver el problema de la ruta más corta usando el algoritmo de Dijkstra. Las variables corresponderán a las bases de datos relacionales, no relacionales y los tiempos de respuesta de los dos gestores de base de datos.

Se realizará una guía práctica para la implementación de un ambiente Big Data en Neo4j, así pues, servirá como un apoyo adicional para el manejo correcto y resolución de dudas con respecto a la creación de un ambiente Big Data.

El presente trabajo de investigación está conformado por cinco capítulos, mismos que se detallan a continuación: el Capítulo I presenta el planteamiento del problema, la justificación y los objetivos planteados en el proyecto de investigación de investigación. El Capítulo II abarca

el marco teórico relacionado a la temática. En el Capítulo III se detalla la metodología empleada en la investigación. El Capítulo IV muestra los resultados y la discusión de la investigación. Finalmente, el Capítulo V presenta las conclusiones y recomendaciones.

CAPÍTULO I

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Problema

En la actualidad el desarrollo inimaginable de la tecnología ha traído consigo un incremento considerable de datos. Dichas masas de datos son conocidas como Big Data, dado que exceden la capacidad de procesamiento de información que presentan las bases de datos convencionales. (Torres et al., 2019)

Por tal razón las organizaciones que trabajan día a día con grandes volúmenes de datos tienden a plantearse la siguiente pregunta: ¿Cuál es el rendimiento de las Bases de Datos Relacionales y no Relacionales al momento de procesar grandes cantidades de datos?, una elección correcta de base de datos dependerá de cada empresa, tomando en cuenta factores como: cantidad de datos procesados, tiempos de respuesta, escalabilidad de los datos, entre otros.

Años atrás las bases de datos relacionales eran una tecnología de punta en procesamiento y gestión de datos en las organizaciones. Pero en la actualidad su ineficiencia para abordar el Big Data se debe a distintos factores que se detallan a continuación; no está diseñada para escalabilidad y resiliencia, cuando las bases de datos experimentan un crecimiento significativo dar mantenimiento es difícil y costoso, además, suelen presentarse fallas en los tiempos de respuesta. (Gonzáles, 2015)

Por otra parte, están las bases de datos NoSQL mismas presentan características acordes a las necesidades de muchas organizaciones en lo que concierne a Big Data. Algunas de las cualidades que presentan estas bases de datos son: tiempos de respuesta rápida, datos escalables, flexibles, capacidad de manejar grandes volúmenes de información con datos estructurados, entre las más relevantes. (Robles et al., 2015)

El uso de las bases de datos NoSQL se han extendido a niveles colosales en empresas como Google, Facebook, Twitter, etc. que han conseguido gran aplicabilidad y alto rendimiento al momento de almacenar grandes volúmenes de datos, además de versatilidad y estabilidad, dado que no manejan los datos a través de una estructura en forma de tabla y relaciones, pues la información se organiza normalmente mediante documentos logrando una mayor flexibilidad. Se logra percibir que esta actual tecnología es un gran aporte para el manejo de información y su empleo abarca diversos sectores de la sociedad donde una correcta manipulación de datos es vital. (Castillo et al., 2017)

1.2. Justificación

La aparición de Internet generó enormes cambios en la informática. Las aplicaciones empezaron a cambiar y a demandar nuevos requerimientos que los SGBDs clásicos no pudieron satisfacer correctamente. Debido a lo cual, surgieron nuevos modelos de bases de datos y gestores que los soporten: bases de datos de documentos, clave-valor, orientados a columnas; entre los más destacados. Los sistemas NoSQL mantienen seis propiedades fundamentales, habilidad de escalar horizontalmente sobre muchos servidores, habilidad de replicar y distribuir datos (particiones) en muchos servidores, interfaz o protocolo simple a nivel de llamada (en contraste con el enlace de SQL), modelo de concurrencia más débil que el ACID (típico de los sistemas de bases de datos relacionales), uso eficiente de RAM e índices distribuidos y habilidad de agregar dinámicamente nuevos atributos a los registros de datos. (Migani, Vera, & Lund, 2018)

La difusión de una nueva alternativa de almacenamiento y procesamiento de datos para compensar los problemas que tiene una Base de Datos Relacional, es sin duda una base de datos NoSQL, debido a la enorme capacidad y escalabilidad al enfrentar el manejo de grandes cantidades de datos, esto en contraparte generado por la ineficiencia de las bases de datos tradicionales y la constante evolución de diversos campos, como la web, comercio electrónico, videojuegos, etc., obligan a las compañías a adoptar nuevas tecnologías. Por lo que estudiar el rendimiento y comportamiento de los dos modelos de base de datos (relacional y no relacional) en un ambiente Big Data, brindará respuestas eficientes de las capacidades de las bases de datos NoSQL.

1.3. OBJETIVOS

1.3.1. Objetivo General

Analizar bases de datos relacionales y no relacionales aplicado al problema de la ruta más corta.

1.3.2. Objetivos Específicos

- Estudiar las arquitecturas de las bases de datos relacional PostgreSQL y no relacional Neo4j en la gestión de grandes volúmenes de datos.
- Elaborar una guía práctica para la implementación de un ambiente Big Data utilizando la base de datos Neo4j en la gestión de grandes volúmenes de datos.
- Evaluar la optimización de problema de la ruta más corta en bases de datos relacionales y no relacionales utilizando conjunto de datos cualificados.

CAPÍTULO II

2. MARCO TEÓRICO

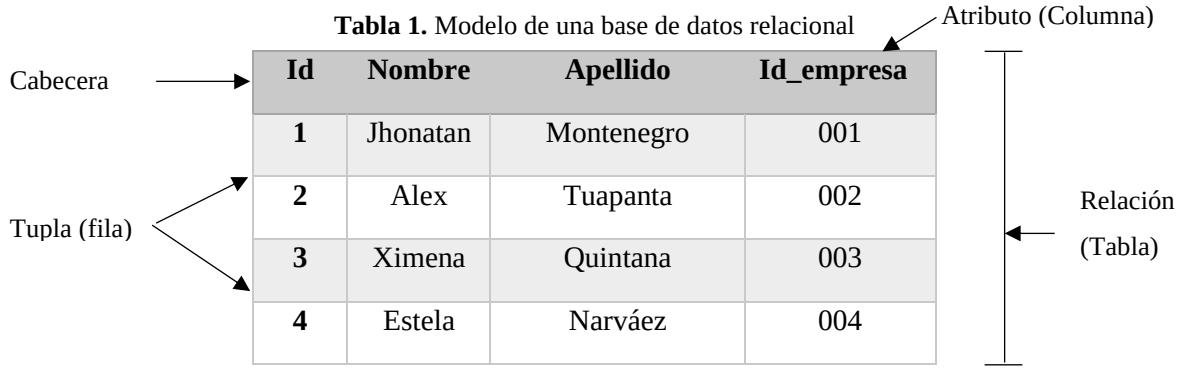
2.1. Bases de Datos

Son grandes cantidades de información almacenadas en registros para lograr una mejor eficiencia al momento de ingresar, buscar, actualizar o eliminar la información. En ciertos casos la información debe estar interrelacionada para evitar la duplicidad de información y mejor organización de la misma. (Valverde, Portalanza, & Mora, 2019)

2.2. Bases de datos SQL

En el modelo relacional una base de datos se representa como una colección de relaciones, representadas por tablas. Aplicando este modelo, una fila recibe el nombre de tupla, la cabecera de la columna recibe el nombre de atributo y el nombre de una tabla se conoce como relación. Cada tupla es identificada de manera unívoca mediante un campo único denominado clave primaria y las relaciones entre las distintas tablas se establecen a través de estos identificadores, que cuando se refieren a otra tabla reciben el nombre de clave ajena. (Rubio, Vega, & Reyes, 2020)

Tabla 1. Modelo de una base de datos relacional



Id	Nombre	Apellido	Id_empresa
1	Jhonatan	Montenegro	001
2	Alex	Tuapanta	002
3	Ximena	Quintana	003
4	Estela	Narváez	004

Fuente: Los Autores

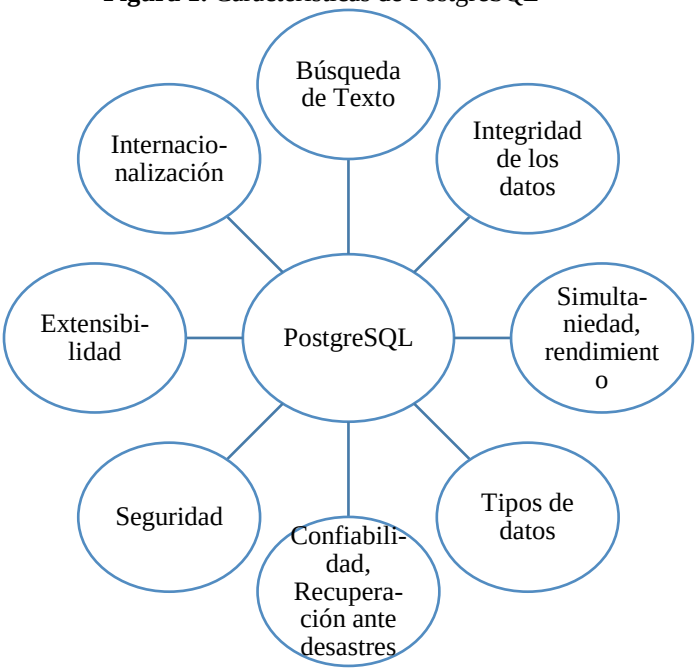
Además, se debe mencionar que estas bases de datos están establecidas por las propiedades ACID, siglas que viene de: Atomicidad, Consistencia, Aislamiento, Durabilidad, mismas que aseguran que el sistema gestor de base de datos (SGBD) cumplan sus funciones sin ningún error o interrupción. (Rubio, Vega, & Reyes, 2020)

2.2.1. PostgreSQL

La página oficial de (PostgreSQL, 2020) establece la definición siguiente; potente sistema de base de datos relacional de código abierto que utiliza y amplía el lenguaje SQL, combinado con

muchas características que almacenan y escalan de forma segura las cargas de trabajo de datos más complicados.

Figura 1. Características de PostgreSQL



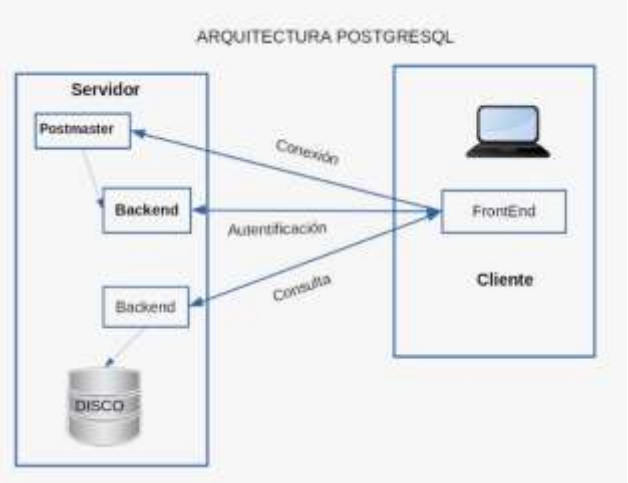
Fuente: Los Autores

Es importante mencionar que PostgreSQL se ejecuta en todos los principales sistemas operativos, es compatible con las propiedades ACID desde 2001, y tiene potentes complementos como el popular extensor de base de datos geoespacial PostGIS (PostgreSQL, 2020). Debido a su excelente rendimiento y grandes características, se ha convertido en la base de datos relacional de gran elección para muchas personas y organizaciones en el mundo.

2.2.2. Arquitectura PostgreSQL

Es un sistema de Gestión de bases de datos objeto-relacional, distribuido bajo la licencia BSD (Berkeley Software Distribution) y su código fuente es libre. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. (Ordoñez, Ríos, & Castillo, 2017)

Figura 2. Arquitectura en PostgreSQL



Fuente: Los Autores

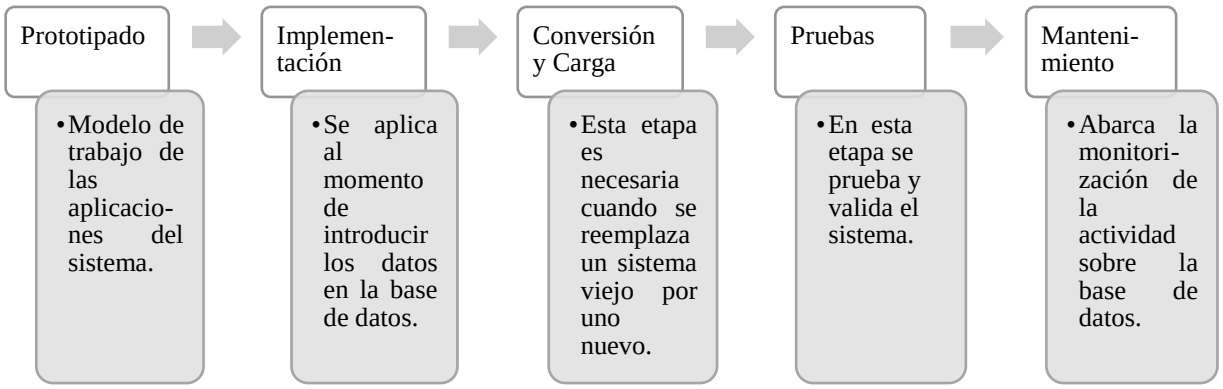
PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Si hubiese un fallo de los procesos no afectará al resto y el sistema continuará funcionando perfectamente. (Valverde, Portalanza, & Mora, 2019)

2.2.3. Diseño de Base de Datos en PostgreSQL

Define la estructura de los datos que debe tener una base de datos en un determinado sistema de información. En el caso de una base de datos relacional, la estructura es un conjunto de esquemas relacionales con sus atributos, claves primarias, claves foráneas, entre otras. (Ordoñez, Ríos, & Castillo, 2017)

2.2.3.1. Diseño de Aplicación

Figura 3. Diseño de Aplicación en PostgreSQL



Fuente: Los Autores

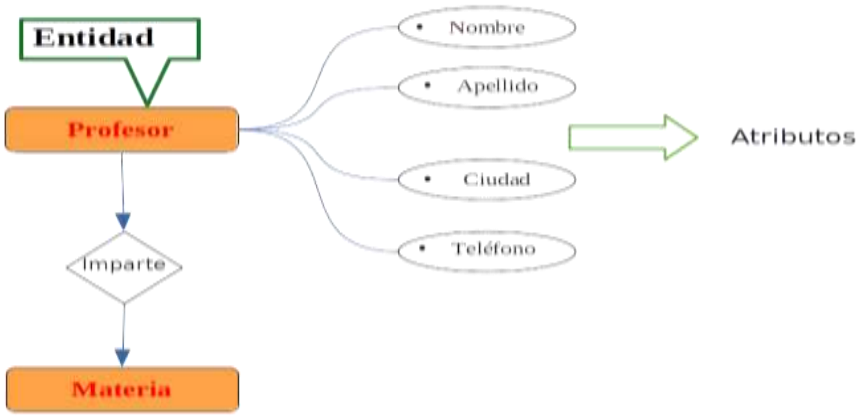
2.2.3.2. Diseño Conceptual

En esta etapa se representa una descripción total del contenido de la base de datos. Se representa en un esquema las entidades, atributos y las relaciones que existen entre ellas. (Ordoñez, Ríos, & Castillo, 2017)

2.2.3.3. Diseño Lógico

La etapa de diseño lógico consiste en obtener un esquema lógico a partir del esquema conceptual generado en la etapa anterior. El esquema lógico depende del tipo de base de datos elegido, aunque es independiente de la implementación concreta del sistema de gestión de bases de datos. (Ordoñez, Ríos, & Castillo, 2017)

Figura 4. Diseño lógico de una base de datos relacional



Fuente: Los Autores

Dentro del diseño lógico, la estructura general de una base de datos está representada de manera gráfica, las entidades se representan por medio de rectángulos, los atributos se representan con elipses, añadido a esto las relaciones entre entidades se representan con un rombo. (Ordoñez, Ríos, & Castillo, 2017)

2.2.3.4. Diseño Físico

Es la manera en la que se adapta el diseño conceptual al sistema gestor de base de datos escogido.

Modelo Entidad Relación

En el libro administración de bases de datos con PostgreSQL, se detalla que el modelo entidad relación está basado en una percepción del mundo real y consta de una colección de objetos básicos llamados entidades y relaciones entre los objetos. (Ordoñez, Ríos, & Castillo, 2017)

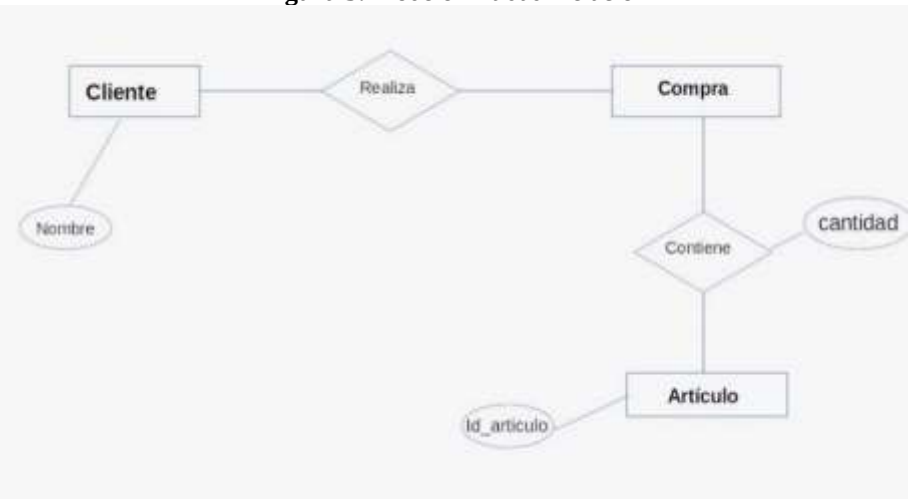
Tabla 2. Componentes del modelo entidad/relación

Ítem	Definición
Entidad	Es la representación de un objeto, evento, situación, acción, etc. del mundo real.
Relación	La relación es el vínculo que existe entre dos o más entidades.
Atributo	Los atributos son datos que serán tratados dentro de las entidades, es decir representan las cualidades que posee un objeto.

Fuente: Los Autores

Como se observa en la Figura 5, la estructura lógica de un base de datos se expresa gráficamente mediante un diagrama entidad relación, mismo que está conformado de los siguientes componentes:

Figura 5. Modelo Entidad-Relación



Fuente: Los Autores

Rectángulos: aquí se detallan las entidades.

Elipses: donde se encuentran los atributos.

Rombos: se representan las relaciones entre las entidades.

Líneas: unen los atributos con las entidades y las entidades con sus respectivas relaciones.

2.2.4. Procesamiento de Datos en PostgreSQL

Al momento de realizar peticiones a la base de datos, el gestor de PostgreSQL no permite dividir una consulta en varios núcleos del procesador (conocido también como consulta en paralelo), por tal razón, si se tiene una consulta muy compleja que se va a ejecutar varias veces, lo óptimo es contar con un procesador rápido, el contar con varios núcleos no aporta ningún beneficio. (Chávez, 2020)

2.3. Bases de datos NoSQL

Permiten resolver problemas de escalabilidad y rendimiento en grandes volúmenes de datos, usando nuevos entornos de datos distribuidos y escalables de forma horizontal (Moreno, Quintero, & Rueda, 2016). Este tipo de bases de datos trabajan con el teorema de CAP (Consistencia, Disponibilidad, Tolerancia al particionamiento). Las características principales que presenta una base de datos no relacional son:

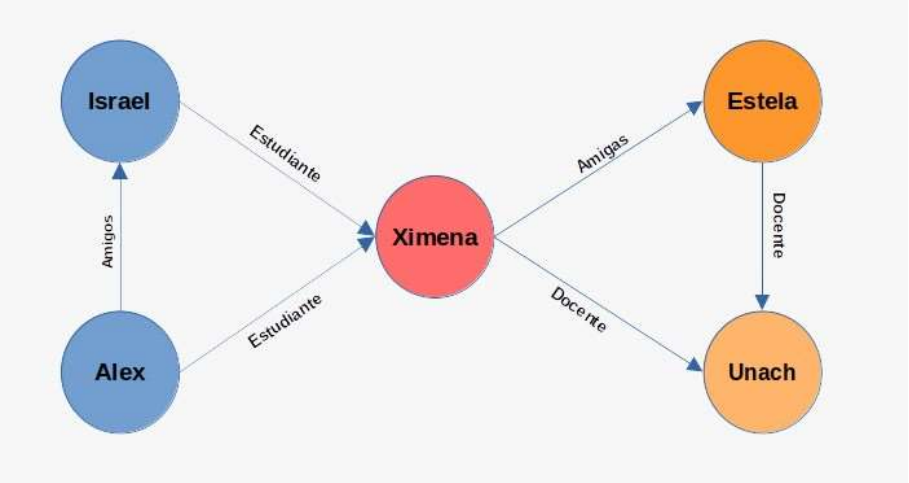
a) El esquema (estructura) de los datos no está necesariamente predefinido.
b) Se permite que el valor de un atributo sea multivaluado (arreglo de valores).
c) Suele haber redundancia de datos (debido a la poca normalización).

Una forma de clasificar las bases de datos NoSQL es por su forma de almacenar los datos, mismas que son: clave/valor, columna, grafo y documento.

2.3.1. Neo4j

Es una base de datos de gráficos de código abierto implementada en Java. Los desarrolladores describen Neo4J como una base de datos totalmente transaccional y un motor Java persistente donde se almacena estructuras en forma de gráficos en lugar de tablas. Neo4J utiliza el almacenamiento de gráficos nativos que brinda la libertad de administrar y almacenar datos de manera altamente disciplinada. Neo4j es considerada la base de datos gráfica más popular y utilizada en todo el mundo, utilizada en áreas como salud, gobierno, producción automotriz, área militar y otras, siendo una referencia importante en esta área. (Neo4j, 2020)

Figura 6. Estructura de un Grafo



Fuente: Los Autores

Estas bases de datos se apoyan en la rama de las matemáticas de la teoría de grafos desarrollada por Leonhard Euler en el siglo XVIII. Los vértices representan las entidades, mientras que las aristas representan las relaciones y propiedades de las mismas. (Fernandes & Bernardino, 2018)

2.3.2. Arquitectura Neo4j

High Available Cluster

Trata de garantizar al máximo la accesibilidad a los datos, aunque varios nodos caigan. En esta arquitectura un clúster está compuesto de una instancia principal (máster) y cero o más instancias secundarias (slave). Las instancias tanto principal como secundaria, tienen una copia completa de toda la base de datos creada con Neo4j. (Constantinov, Poteras, & Mihai, 2016)

Casual Cluster

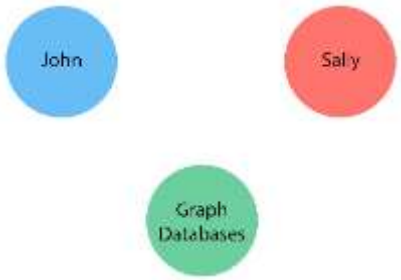
Su objetivo principal es dotar de una mayor escalabilidad a la base de datos y brindar una mayor eficiencia de las operaciones de lectura mediante la reubicación de los nodos, que pueden estar distribuidos en diferentes espacios geográficos.

2.3.3. Diseño de Base de datos en Neo4j

Nodos

Las primeras entidades identificadas en el dominio son los nodos, siendo una de las dos unidades fundamentales que forman un gráfico. Los nodos se utilizan a menudo para representar entidades, pero también pueden representar otros componentes de dominio, según el caso de uso. Los nodos pueden contener propiedades que contengan pares de datos de nombre-valor. (Neo4j, 2020)

Figura 7. Modelo de un Grafo, Nodos

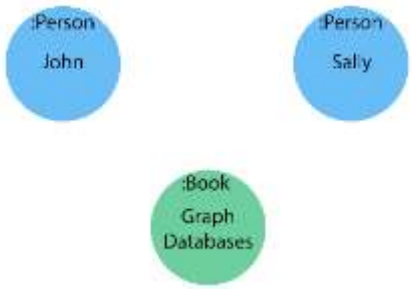


Fuente: (Neo4j, 2020)

Etiqueta

Es una construcción de gráfico con nombre, que se utiliza para agrupar nodos en conjuntos. Todos los nodos etiquetados con la misma etiqueta pertenecen al mismo conjunto. (Neo4j, 2020)

Figura 8. Modelo de un Grafo, Etiquetas



Fuente: (Neo4j, 2020)

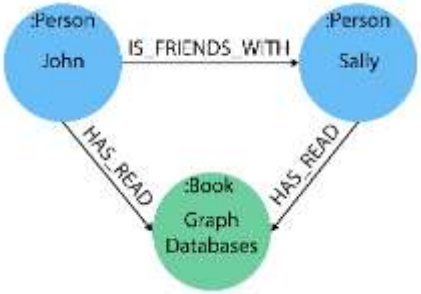
Relación

Conecta dos nodos y permite encontrar nodos de datos relacionados. Tiene un nodo de origen y un nodo de destino que muestra la dirección de la flecha. Aunque debe almacenar una relación en una dirección particular, Neo4j tiene el mismo rendimiento transversal en cualquier dirección, por lo que puede consultar la relación sin especificar la dirección. (Neo4j, 2020)

Relaciones entre los nodos:

John es amigo de Sally
Sally es amiga de John
John ha leído Bases de datos de gráficos
Sally ha leído Bases de datos de gráficos

Figura 9. Modelo de un Grafo, Relaciones



Fuente: (Neo4j, 2020)

2.3.3.1. Estudio de Problema

En el marco de la Teoría General de Sistemas, el análisis de sistemas tiene como objetivo general la comprensión de los sistemas complejos para abordar su modificación de forma que se mejore el funcionamiento interno para hacerlo más eficiente, para modificar sus metas, etc. (Peñalvo, 2018)

2.3.3.2. Identificación de Requisitos

Se define como una aplicación disciplinada de principios científicos y técnicas para desarrollar, comunicar y gestionar requerimientos. Así mismo, definen la Ingeniería de requisitos como todas las actividades relacionadas con la identificación y documentación de las necesidades de clientes y usuarios. (Medina, Pineda, & Tellez, 2019)

2.3.3.3. Modelado del Negocio

La realización del modelado del negocio, durante las fases tempranas del desarrollo, contribuye a lograr una adecuada comprensión del problema y de su dominio, lo cual facilita la identificación, análisis y especificación de los requisitos de la solución. (Rojas, Martinez, & Sanchez, 2018)

2.3.4. Procesamiento de grafos en Neo4j

Cada nodo tiene una referencia directa a su nodo adyacente, el tiempo de una consulta no depende del tamaño total de la base de datos sino al área de búsqueda del grafo.

Hay ficheros para nodos, relaciones y propiedades. Al estar las propiedades de cada nodo relación almacenada en un fichero diferente, el almacenamiento de nodos y relaciones se preocupa solo de la estructura del grafo. (Lopez et al., 2016)

2.3.4.1. Algoritmos integrados a la librería Neo

Algoritmo de Centralidad

Determina la importancia de los distintos nodos dentro de una red o grafo. (Neo4j, 2020)

2.3.4.2. Algoritmos de detección de Comunidad

- **Louvain:** algoritmo para detectar comunidades en redes, depende de una heurística para maximizar la modularidad.

- **Label Propagation:** algoritmo para encontrar comunidades en tiempo casi lineal las detecta utilizando la estructura de red como guía no requiere ni una función objetivo predefinida ni información previa. (Romeo, Martínez, & Rico, 2018)
- **Weakly Connected Components**: encuentra conjuntos de nodos conectados donde cada nodo es accesible desde cualquier otro nodo en el mismo conjunto.
- **Strongly Connected Components:** encuentra grupos de nodos donde cada nodo es directamente accesible desde cualquier otro nodo del grupo.
- **Triangle Count / Clustering Coefficient:** algoritmo de gráfico determina la cantidad de triángulos que pasan por cada nodo. (Neo4j, 2020)

2.3.4.3. Algoritmo de búsqueda de ruta

Llegar a un destino en tiempo y forma, puede requerir que el algoritmo de enrutamiento que es el encargado de escoger las rutas y las estructuras de datos, cumpla con ciertas propiedades que aseguren la eficiencia de su trabajo. Dichas propiedades son: corrección, estabilidad, robustez, equitatividad, sencillez y optimalidad. El algoritmo de la ruta más corta calcula una topología sin bucles con el nodo como punto de partida y examinando a su vez la información que posee sobre nodos adyacentes. (Riaño, Toro, & Rico-Bautista, 2018)

- **Minimum Weight Spanning Tree (MST):** subgrafo con los nodos del grafo original y las relaciones que conectan todos los nodos con el peso mínimo.
- **Single source shortest path:** calcula la ruta más corta entre un par de nodos.
- **All pairs shortest path:** este algoritmo encuentra las rutas más cortas entre todos los pares de nodos en el gráfico.
- **Shortest path:** algoritmo que resuelve o busca la ruta más corta entre todos los nodos de un grafo.

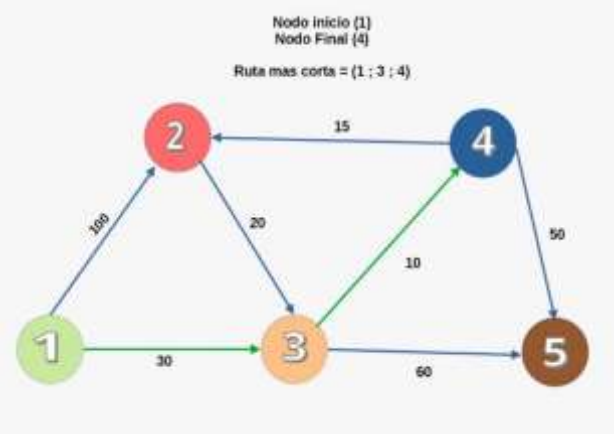
2.3.4.4. Algoritmo de Dijkstra

Permite calcular la ruta más corta desde un vértice inicial a todos los demás vértices a partir de un gráfico orientado y ponderado. La razón es encontrar un subgrafo en donde se enumere los vértices de menor a mayor (ascendente), de su distancia mínima desde el vértice inicial o de partida. (Riaño, Toro, & Rico-Bautista, 2018)

El algoritmo de Dijkstra empieza calculando la distancia para cada uno de los arcos tomando un vértice como base. Desde el vértice con el peso más pequeño, la distancia para cada arco se

determina de nuevo y se realiza este proceso las n veces posibles, hasta que todos los vértices se hayan atravesado. De esta forma la ruta más corta es la que presente menor costo cuando recorra todos los vértices. (Cardona, Castrillón, & Tinoco, 2017)

Figura 10. Estructura de un grafo dirigido



Fuente: Los Autores

Este algoritmo realiza O(n2) operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértices. (Riaño, Toro, & Rico-Bautista, 2018)

2.3.5. Herramienta Cypher

Es un lenguaje de consulta declarativa para gráficos de propiedades. Cypher proporciona capacidades para consultar y modificar datos, así como para especificar definiciones de esquema.

El modelo de datos de Neo4j que utiliza Cypher es el de gráficos de propiedades. El modelo comprende nodos representando entidades (como personas, cuentas bancarias, departamentos, etc.), y relaciones (sinónimo de bordes), representando las conexiones o relaciones entre las entidades. (Francis et al., 2018)

2.4. Open Street Map (OSM)

Conjunto de datos abiertos de código libre que puede ser usado para cualquier propósito. Utiliza nodos (punto con coordenadas), segmentos (enlace dirigido entre nodos) y formas (lista ordenada de segmentos) para representar una red de transporte. Cualquiera de estos objetos puede contener etiquetas. Las etiquetas son pares clave/valor que denotan el tipo y las propiedades del objeto. Básicamente, OSM es una forma de un gráfico de propiedades dirigido disperso, pero no almacenado como tal en una forma nativa. Los nodos OSM representan nodos

del gráfico y las formas agrupan una relación entre esos nodos. Los datos sin procesar de OSM se almacenan en el formato basado en XML. (Open Street Map Wiki, 2020)

OpenStreetMap utiliza una estructura de datos topológicos:

- Los nodos son puntos con una posición geográfica.
- Las formas son listas de nodos, que representan una polilínea o un polígono.
- Las relaciones son grupos de nodos, formas y otras relaciones a las que se pueden asignar ciertas propiedades.
- Las propiedades se pueden asignar a nodos, formas o relaciones y constar de pares .name = value.

2.5. pgRouting

En los (Workshops, 2020) se detalla que es una extensión para PostgreSQL/PostGIS que añade funcionalidades para análisis de redes y planificación de rutas. Las ventajas del enfoque de enrutamiento de bases de datos son:

- Muchos clientes pueden modificar datos y atributos, como QGIS y uDig a través de JDBC, ODBC o directamente mediante PL/pgSQL. Los clientes pueden ser PC o dispositivos móviles.
- Los cambios de datos se pueden reflejar instantáneamente a través del motor de enrutamiento. No hay necesidad de precalculación.
- El parámetro “cost” se puede calcular dinámicamente a través de SQL y su valor puede provenir de varios campos o tablas.

2.6. PostGIS

Es un extensor de base de datos espacial para la base de datos relacional de objetos PostgreSQL. Agrega compatibilidad con objetos geográficos que permiten ejecutar consultas de localización en SQL. (PostGIS, 2021)

2.7. Osm2pgrouting

Es una herramienta de línea de comandos que importa datos de OpenStreetMap en una base de datos pgRouting. Crea automáticamente la topología de red de enrutamiento y crea tablas para tipos de entidades y clases de carretera. (pgRouting, 2021)

2.8. Anaconda

Es un entorno completo de herramientas, desarrolladas en lenguaje Python de código libre y abierto, utilizado esencialmente en la ciencia de datos y aprendizaje autónomo de máquinas, funciona como un gestor de entorno y a su vez como gestor de paquetes Python. (Anaconda Inc, 2021)

2.9. OSMnx

Es un paquete de Python que permite descargar geometrías espaciales, modelar, proyectar, visualizar y analizar redes de calles del mundo real desde las API de OpenStreetMap. Los usuarios pueden descargar y modelar redes urbanas para caminar, conducir o andar en bicicleta con una sola línea de código Python, luego analizarlas y visualizarlas fácilmente. (Boeing, 2017)

2.10. NeoMap

Debido a que Neo4j cuenta con un browser incorporado, el mismo no es una herramienta óptima para visualizar nodos que tengan atributos geoespaciales, por ello NeoMap constituye una poderosa herramienta adicional instalable, que ayuda a visualizar y manejar grandes cantidades de datos geoespaciales, a su vez se integra con el lenguaje de consultas Cypher, para análisis complejos. (Scifo, 2019)

CAPÍTULO III

3. METODOLOGÍA

3.1. Enfoque

Esta investigación tiene un enfoque cuantitativo porque a través de mediciones se buscará cuantificar el tiempo de ejecución de las operaciones CRUD (Crear, Leer, Actualizar, Borrar) a la base de datos, adicional a esto se analizará el comportamiento y rendimiento de cada modelo de base de datos (PostgreSQL y Neo4j), a través de la ejecución del algoritmo de la ruta más corta.

3.1.1. Identificación de las variables

Variables Dependientes:

Eficiencia en el cálculo de la ruta más corta.

Variables Independiente:

- Base de Datos Relacional (PostgreSQL)
- Base de Datos No Relacional (Neo4j)

3.1.3. Operacionalización de las Variables

Tabla 3. Operacionalización de las Variables

Variable	Tipo	Descripción	Dimensiones	Indicadores
Base de Datos Relacional	Independiente	Se basan en el modelo relacional, una forma intuitiva y directa de representar datos en tablas. (Rubio, Vega, & Reyes, 2020)	Análisis	Análisis Cuantitativo
Base de Datos No Relacional	Independiente	Una base de datos no relacional es aquella que no usa el esquema tabular de filas y columnas, en su lugar usan un modelo de almacenamiento que está optimizado para los requisitos específicos del tipo de datos que se almacena. (Moreno, Quintero, & Rueda, 2016)	Análisis	Análisis Cuantitativo
Eficiencia en el cálculo de la Ruta más corta	Dependiente	Algoritmo de Dijkstra. También llamado algoritmo de caminos mínimos es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. (Cardona, Castrillón, & Tinoco, 2017)	Complejidad	Tiempo de Carga Rendimiento Tamaño en Disco

Fuente: Los Autores

3.2. Tipo y Diseño de la Investigación

3.2.1. Investigación Cuasi Experimental

Es cuasi experimental porque se obtendrá los resultados de los tiempos de respuesta en la ejecución de operaciones CRUD (Crear, Leer, Actualizar, Borrar) y el tiempo de ejecución del algoritmo de Dijkstra en cada modelo de base de datos, el modelo entidad relación perteneciente a la base de datos relacional PostgreSQL y el modelo orientado a grafos en la base de datos Neo4j.

3.2.2. Investigación Analítica

Es de tipo analítico debido a que las variables independientes serán la base de datos relacional (PostgreSQL) y la base de datos no relacional (Neo4j), esencialmente relacionadas con la variable de eficiencia en el cálculo de la ruta más corta (variable dependiente). Con ello se pretende comparar, que modelo de base de datos se desenvuelve de forma más eficiente al ejecutar el algoritmo de Dijkstra en un escenario Big Data.

3.2.3. Investigación Bibliográfica

Permite la recopilación y búsqueda de estudios e investigaciones anteriores, sobre el desempeño de las bases de datos NoSQL. Adicional se estudia las arquitecturas de las bases de datos relacional y no relacional. A partir de esta información se desarrollará una guía práctica para la implementación de un ambiente Big Data en Neo4j.

3.3. Unidad de Análisis

Tabla 4. Unidad de Análisis

Población	Muestra	Unidad de Análisis	Variables
Base de datos relacional (PostgreSQL) y no relacional (Neo4j)	Número de consultas	PostgreSQL y Neo4j	Tiempo de respuesta CPU Memoria RAM Disco Duro
Algoritmo de Dijkstra	Complejidad	pgRouting y Cypher	Eficiencia en el cálculo de la ruta

Fuente: Los Autores

3.4. Población y Muestra

Para el desarrollo de esta investigación se tendrá como población y muestra el total de los registros obtenidos del conjunto de datos abiertos del proyecto OpenStreetMap (OSM). Además, debido a que la cantidad de datos utilizados no supera los 100000 nodos, se puede considerar como una población finita.

3.4.1. Población

Para la investigación se establece una población total de 62.817 datos. Dichos valores representan cada una de las intersecciones o nodos de las calles de la ciudad de Quito.

3.4.2. Muestra

Se optó por un muestreo no probabilístico por conveniencia. Esta técnica es una de las más recursivas para ahorrar tiempo, dado que no se emplea mucho esfuerzo o se utiliza algún método especial para encontrar los elementos que serán parte de la muestra, la selección de los datos queda a criterio del investigador. (Parra & Vázquez, 2017)

Total, de la población = 62.817 nodos

Número de pruebas = 10

Intervalo de selección de datos = Aleatorio

3.5. Técnicas de Recolección de Datos

La técnica que se aplicó para la recopilación de datos es la documental.

3.5.1. Técnica Documental

A través de esta técnica se procedió al análisis de fuentes bibliográficas relacionados al tema de investigación. Los trabajos escritos por diferentes autores fueron analizados y comparados de manera minuciosa con el fin de tener una idea clara y concisa de lo que se pretende realizar en la investigación. Los documentos utilizados para este estudio fueron obtenidos de fuentes oficiales. Para esta investigación se analizó artículos de revistas científicas, conferencias, libros y páginas oficiales en la web.

3.6. Técnicas de análisis e interpretación de resultados

3.6.1. Herramientas Utilizadas

Tabla 5. Herramientas y versiones utilizadas

Herramientas	Versión
PgAdmin Desktop	4.21
Base datos PostgreSQL	11
PostGIS	2.5.3
pgRouting	2.6.2
Neo4j Desktop	1.3.3
Base de datos Neo4j	4.1.3 Enterprise
Lenguaje de consulta Cypher	N/D
Graph Data Science Library	1.4.0
APOC	4.1.0.2

Fuente: Los Autores

Para la obtención del tiempo de ejecución del algoritmo de Dijkstra en PostgreSQL se utiliza el runtime que viene incluido en el núcleo de esta base de datos, los datos que arroja este gestor son: el tiempo de ejecución expresado en milisegundos (ms) y la cantidad de columnas afectadas. En el caso de Ne4j los tiempos de respuesta son proporcionados en milisegundos, mediante el lenguaje de consulta Cypher que viene integrado en el núcleo de esta base de datos, adicionalmente proporciona el número total de nodos recorridos una vez finalizado la ejecución del algoritmo de Dijkstra.

CAPÍTULO IV

4. RESULTADOS Y DISCUSIÓN

4.1. Características del equipo con el cual se realizó las pruebas de rendimiento

Para realizar las pruebas de rendimiento de la base de datos relacional (PostgreSQL) y no relacional (Neo4j), se utilizó un computador portátil con las siguientes características:

Tabla 6. Características del Equipo

Característica	Detalle
Procesador	Intel (R) Core (TM) i5-7200U CPU @ 2.50GHz 2.70GHz
Memoria RAM	8,00 GB
Sistema operativo	Windows 10 Home
Tipo de sistema	64 bits

Fuente: Los Autores

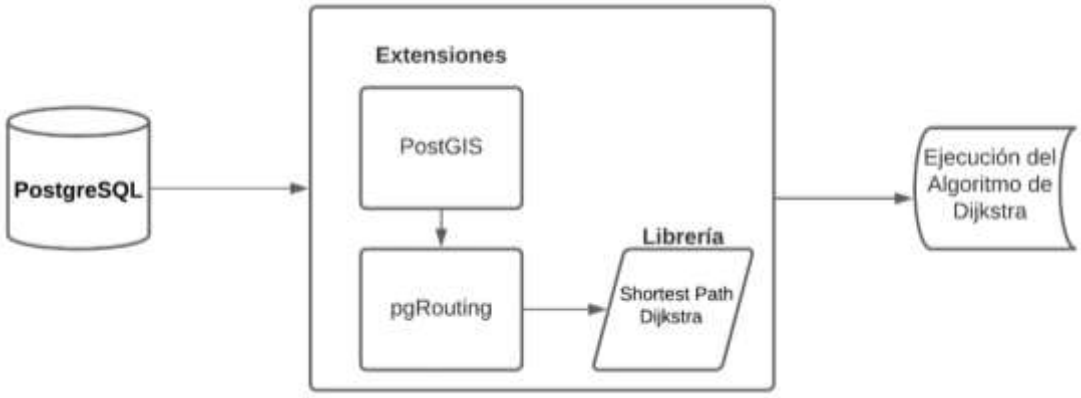
4.2. Configuraciones Adicionales

Debido a que los datos obtenidos de OpenStreetMap, no son un formato compatible para Neo4j, se utilizaron herramientas adicionales (Anaconda, OSMnx) para obtener un archivo con extensión (.graphml) compatible con Neo4j. Anaconda en conjunto con el gestor de paquetes Python llamado OSMnx, utilizan la API (Interfaz de Programación de Aplicaciones) de OpenStreetMap para descargar mapas geoespaciales en formato (.graphml).

4.3. Extensiones adicionales para el cálculo de la ruta más corta

Una vez cargada la base de datos en el gestor PostgreSQL se crea la extensión PostGIS que habilita el soporte para trabajar con objetos geográficos localizados en el espacio. A su vez, se crea la extensión pgRouting, la misma que añade funcionalidades para análisis de redes y planificación de rutas permitiendo realizar cálculos de rutas óptimas desde el propio gestor de base de datos.

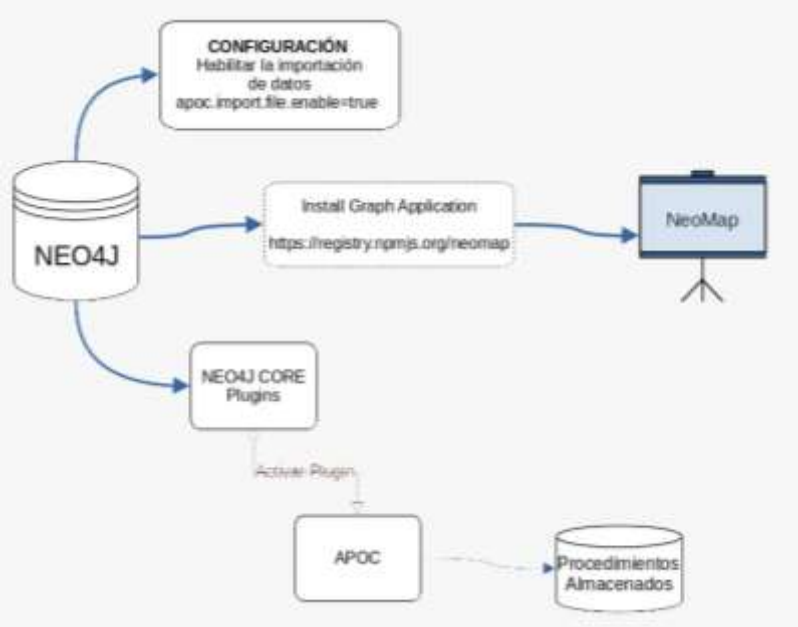
Figura 11. Extensiones añadidas en PostgreSQL



Fuente: Los Autores

Una vez creada la base de datos en Neoj4 se debe habilitar la importación de archivos dentro de su configuración, mediante la línea de código "apoc.import.file.enable=true". Posteriormente instalar el plugin APOC en la base de datos creada, este último contiene todos los procedimientos almacenados y algoritmos necesarios para el cálculo de la ruta más corta, adicional, se usó el plugin NeoMap, el cual servirá para visualizar el trazo de la ruta más corta de un punto a otro.

Figura 12. Configuraciones en Neo4j



Fuente: Los Autores

4.4. Resultados

En esta investigación se realizó 10 pruebas tanto en PostgreSQL como en Neo4j. Dichas pruebas fueron realizadas en frío (arranque del computador en cero) y en caliente (uso del computador con varios procesos ejecutándose a la par).

En la Tabla 7, se muestran los valores de arranque del computador antes de iniciar con los procesos del cálculo de la ruta más corta en cada una de las bases de datos.

Tabla 7. Estado inicial de los recursos del computador

Recursos	**Recursos del computador en caliente**		**Recursos del computador en frío**	
	PostgreSQL	Neo4j	PostgreSQL	Neo4j
CPU inicial	12%	12%	1%	1%
Memoria RAM inicial	87%	87%	30%	30%
Uso inicial del Disco Duro	18%	18%	5%	5%

Fuente: Los Autores

Una vez determinado los valores del arranque del computador, se procede a la aplicación del algoritmo de la ruta más corta tanto en PostgreSQL como en Neo4j.

4.4.1. Pruebas en Caliente aplicado a los gestores PostgreSQL y Neo4j

Tabla 8. Resultado de las pruebas en caliente PostgreSQL y Neo4j

N°	**Nodo Inicio**	**Nodo Fin**	**Tiempo (ms)**	
			PostgreSQL	**Neo4j**
1	7526	447	1316	107
2	39350	4170	513	129
3	16218	21127	384	147
4	6386	17378	585	249
5	17779	458	345	122
6	17958	12036	668	128
7	52509	58510	389	91
8	10450	5346	424	70
9	50315	51071	350	93
10	18037	73251	315	85

Fuente: Los Autores

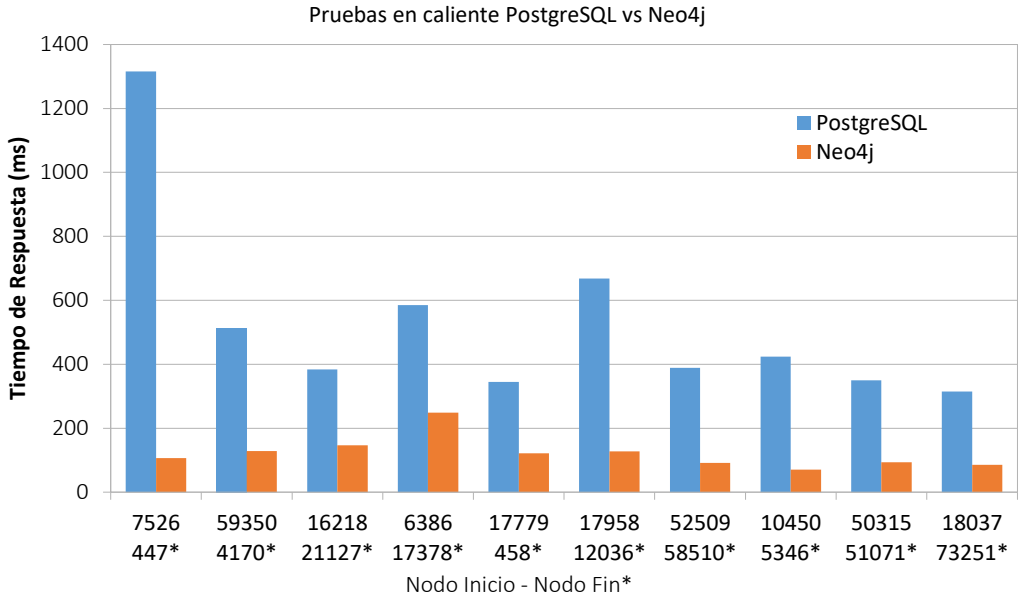
La Tabla 8, muestra las pruebas con diferentes rutas recorridas, tomando un nodo inicial como punto de partida y un nodo final como punto de llegada, obteniendo los resultados del tiempo de respuesta. Estos valores son expresados en milisegundos (ms).

4.4.1.1. Tiempos de Respuesta en Milisegundos (ms)

La Figura 13, muestra el tiempo de respuesta de cada gestor de base de datos al aplicar el algoritmo de la ruta más corta. Los tiempos de respuesta de Neo4j son mucho más rápidos en

comparación a PostgreSQL. Cabe mencionar que se utilizaron los mismos datos o puntos para la resolución del algoritmo de Dijkstra.

Figura 13. Tiempos de respuesta en milisegundos de las pruebas en caliente

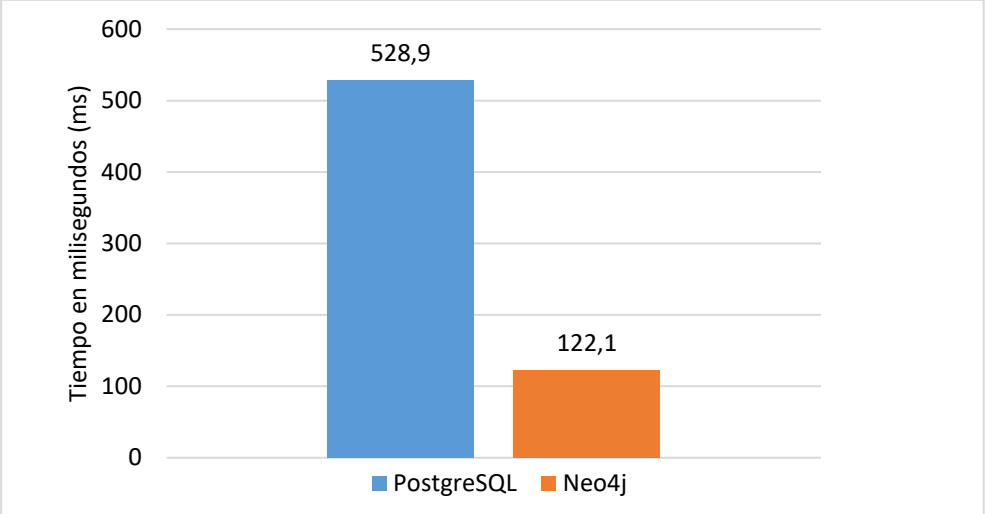


Fuente: Los Autores

4.4.1.2. Tiempo promedio de respuesta en Milisegundos

En la Figura 14, las pruebas en caliente arrojan como resultado que el gestor de base de datos PostgreSQL alcanzó un tiempo de respuesta promedio de 528.9 milisegundos, mientras tanto el gestor de base de datos Neo4j obtuvo un tiempo de respuesta promedio de 122.1 milisegundos. En consecuencia, la diferencia en el tiempo de respuesta entre los dos gestores es de 406.8 milisegundos. Es decir, Neo4j resuelve el algoritmo de la ruta más corta cuatro veces más rápido en comparación de PostgreSQL.

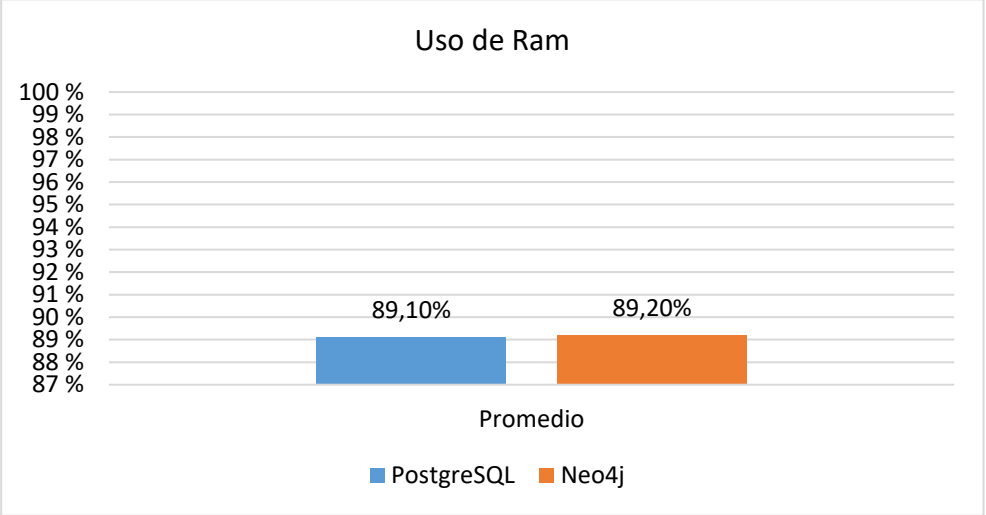
Figura 14. Tiempo promedio de respuesta de las pruebas en caliente



Fuente: Los Autores

4.4.1.3. Uso de memoria RAM

Figura 15. Uso de la memoria RAM en las pruebas en caliente

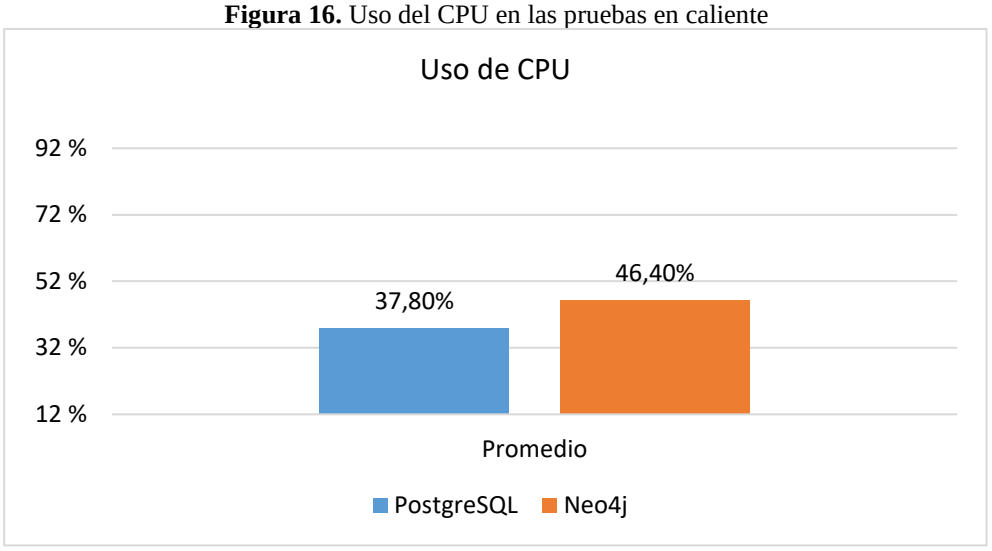


Fuente: Los Autores

Durante las pruebas en caliente, partiendo del uso de la memoria RAM en 87%, el uso promedio de la memoria RAM se incrementa un 2,1% en PostgreSQL y un 2,2% en Neo4j. La diferencia es de 0,1% a favor del gestor de base de datos PostgreSQL (Figura 15).

4.4.1.4. Uso de CPU

Figura 16. Uso del CPU en las pruebas en caliente

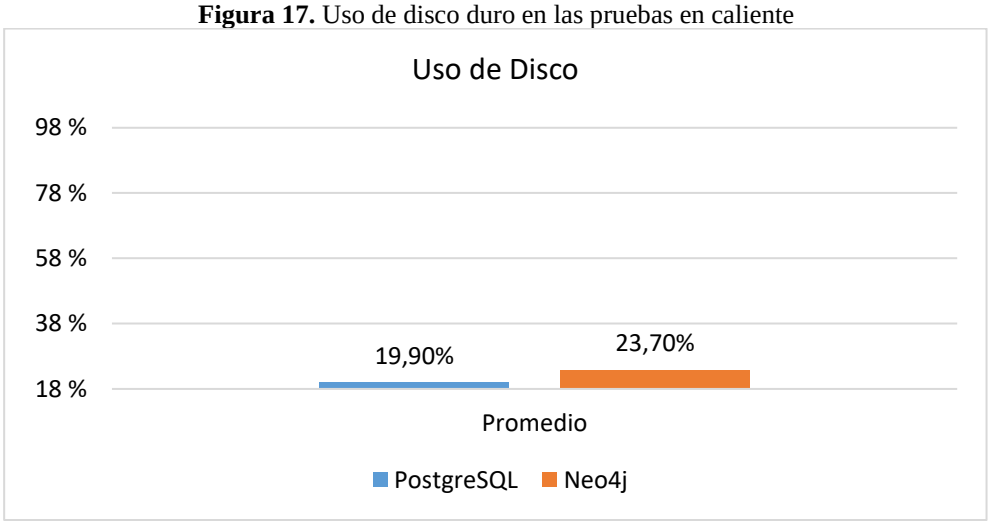


Fuente: Los Autores

El uso de CPU tiene un estado inicial de 12%. Al momento de aplicar el algoritmo de ruta más corta en PostgreSQL incrementa el porcentaje en un 25.8%. Neo4j tiene un incremento de 34,4%. En consecuencia, existe un consumo mayor de recursos de CPU por parte de Neo4j, con un 8,6% en comparación del otro gestor.

4.4.1.5. Uso de Disco Duro

Figura 17. Uso de disco duro en las pruebas en caliente



Fuente: Los Autores

El estado inicial del disco duro se encuentra en un 18%. PostgreSQL presenta un incremento de 1,90% con respecto al estado inicial. El porcentaje del uso promedio de disco en Neo4j presenta un incremento de 5,70%. La diferencia entre los gestores es de 3.80%, existiendo una mayor incidencia de consumo en disco duro por parte del gestor Neo4j.

4.4.2. Pruebas en Frío aplicado a los gestores PostgreSQL y Neo4j

En la Tabla 9, se expone los tiempos de respuesta obtenidos al aplicar el algoritmo de la ruta más corta en PostgreSQL y en Neo4j. Los valores presentados se expresan en la unidad de tiempo milisegundos (ms).

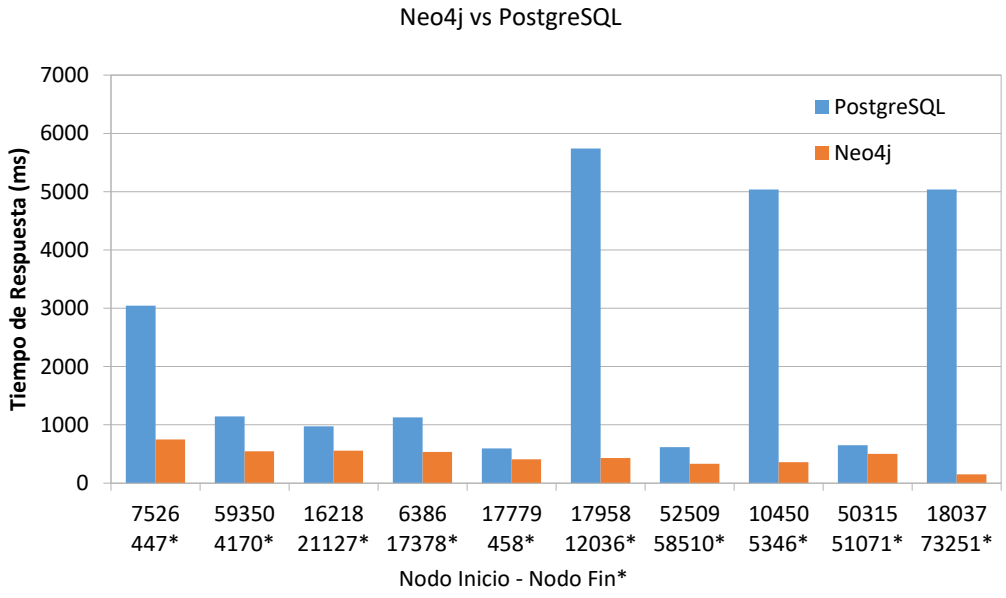
Tabla 9. Resultado de las pruebas en frío PostgreSQL y Neo4j

N°	Nodo Inicio	Nodo Fin	Tiempo (ms)	
			PostgreSQL	Neo4j
1	7526	447	3045	740
2	39350	4170	1143	545
3	16218	21127	976	556
4	6386	17378	1130	538
5	17779	458	598	407
6	17958	12036	5759	431
7	52509	58510	617	331
8	10450	5346	5039	358
9	50315	51071	653	503
10	18037	73251	5039	153

Fuente: Los Autores

4.4.2.1. Tiempos de respuesta en Milisegundos (ms)

Figura 18. Tiempos de respuesta en milisegundos de las pruebas en frío



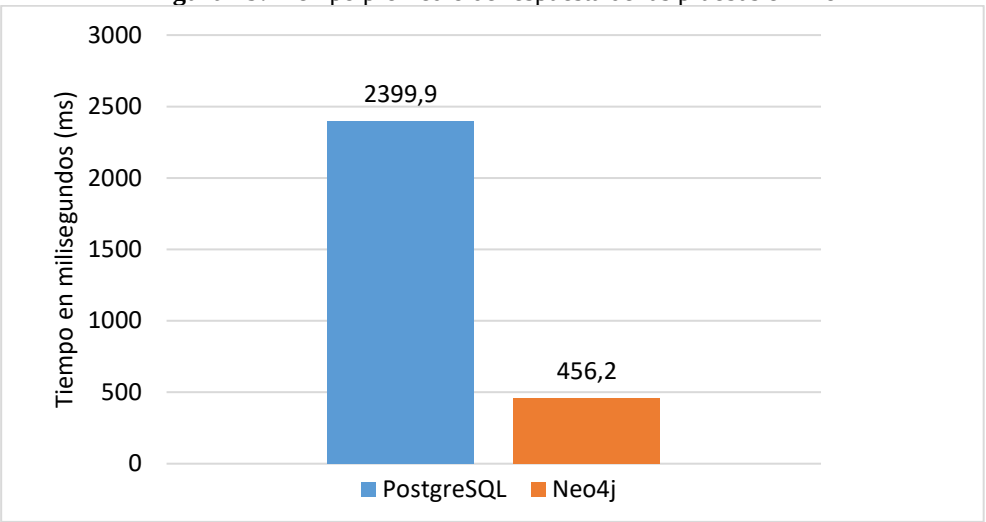
Fuente: Los Autores

En la Figura 18, se presenta los tiempos obtenidos en los dos gestores de bases de datos al aplicar el algoritmo de Dijkstra. El nodo inicial y el final son los mismos para cada gestor. A

simple vista se aprecia que el gestor de bases de datos Neo4j presenta tiempos de resolución del algoritmo mucho más rápidos que PostgreSQL.

4.4.2.2. Tiempo promedio de respuesta en Milisegundos

Figura 19. Tiempo promedio de respuesta de las pruebas en frío



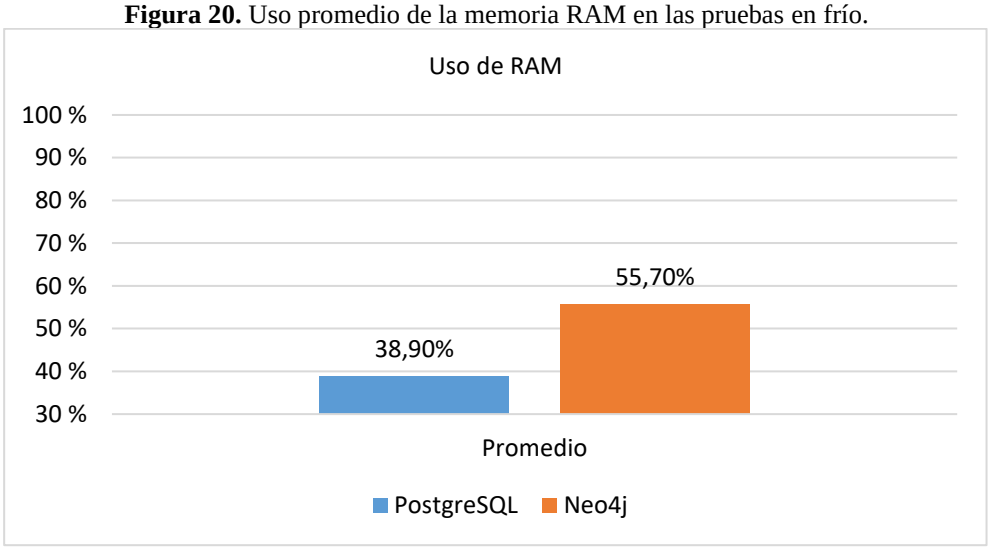
Fuente: Los Autores

Como se aprecia en la Figura 19, existe una diferencia de 1943,7ms entre las bases de datos relacional PostgreSQL y no relacional Neo4j. Mediante este resultado se afirma que Neo4j resuelve el algoritmo de la ruta más corta, aproximadamente, 4 veces más rápido que PostgreSQL en lo que concierne a las pruebas en frío.

4.4.2.3. Uso de Memoria RAM

El computador tiene una memoria RAM inicial del 30%. PostgreSQL presenta un incremento en memoria RAM de 8,90%. Neo4j aumenta el consumo de la memoria en un 25,70%. En consecuencia, existe una diferencia de 16.8% entre los dos gestores, siendo Neo4j el gestor que más recursos de memoria RAM consume al momento de la ejecución del algoritmo de ruta más corta (Figura 20).

Figura 20. Uso promedio de la memoria RAM en las pruebas en frío.

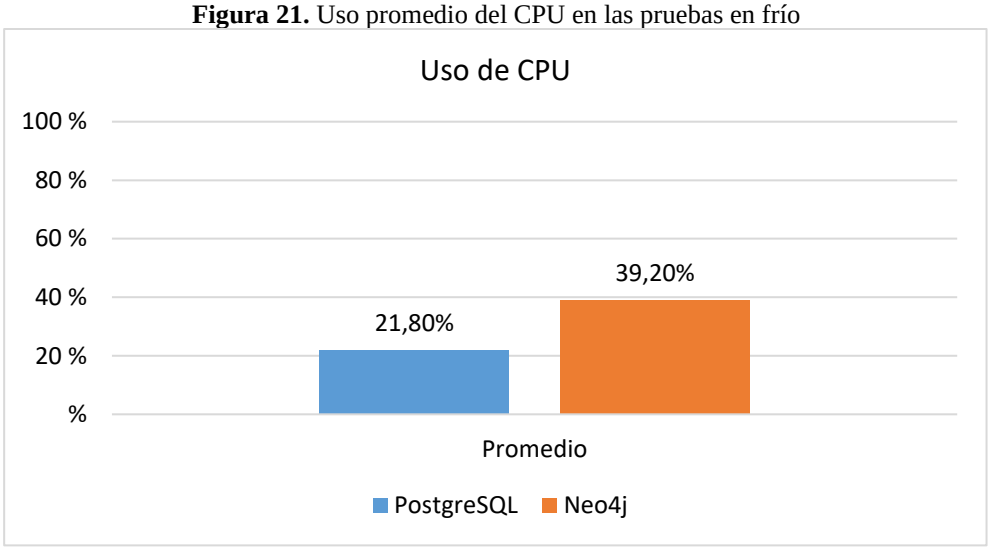


Fuente: Los Autores

4.4.2.4. Uso de CPU

El consumo del CPU tiene un valor inicial del 1%. Existe un incremento del CPU en un 20,80% con respecto a su estado inicial en PostgreSQL, a su vez, Neo4j presenta un incremento de 38,20%. Finalmente, se afirma que Neo4j presenta un mayor consumo de recursos de CPU al momento de la ejecución del algoritmo de la ruta más corta (Figura 21).

Figura 21. Uso promedio del CPU en las pruebas en frío



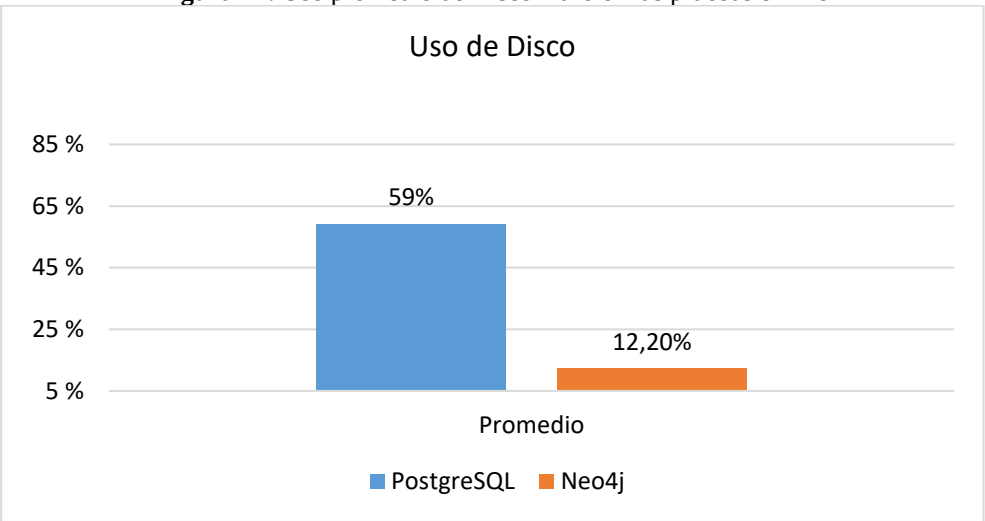
Fuente: Los Autores

4.4.2.5. Uso de Disco Duro

El estado inicial del disco duro es del 5%. Tras aplicar el algoritmo de la ruta más corta en PostgreSQL, se aprecia un incremento en disco duro del 54%. Neo4j presenta un incremento

de 7,20%. Con una diferencia de 46.8% Neo4j es el gestor de base de datos que menor uso de disco duro presenta.

Figura 22. Uso promedio de Disco Duro en las pruebas en frío



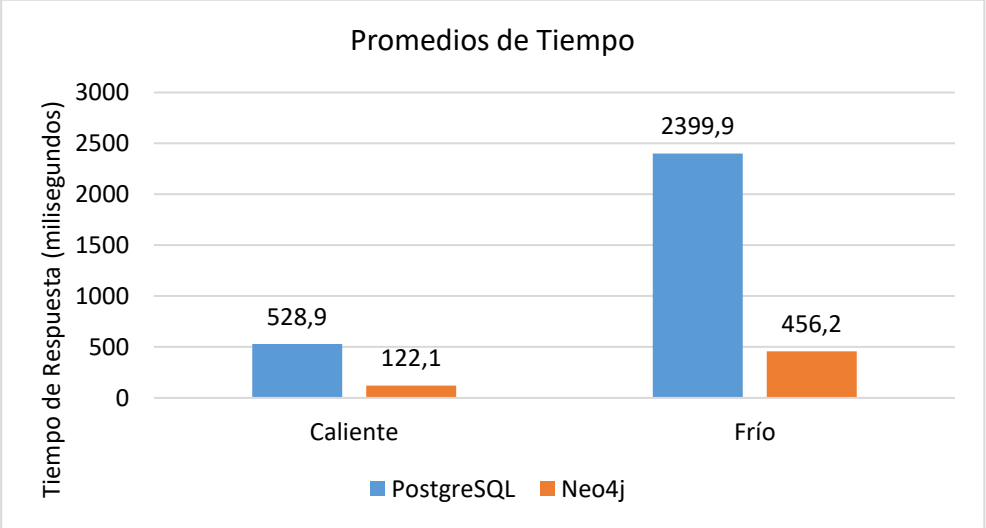
Fuente: Los Autores

4.4.3. Eficiencia en el cálculo de ruta más corta Neo4j vs PostgreSQL

La Figura 23, presenta una marcada diferencia de tiempos de respuesta entre los dos gestores. Los datos estadísticos evidencian un rendimiento superior del gestor Neo4j, obteniendo una ventaja de 406.8ms por cada recorrido de ruta que se realizó para las pruebas en caliente. Por otra parte, las pruebas en frío muestran una superioridad de 1943.7ms para cada trazo de ruta realizado en Neo4j.

En base a los datos presentados, se afirma que el tiempo de respuesta de Neo4j tanto para las pruebas en caliente como en frío es muy superior a PostgreSQL, debido a la rapidez con la que resuelve el algoritmo de Dijkstra.

Figura 23. Tiempo promedio de respuesta de las pruebas en caliente vs frío



Fuente: Los Autores

4.4.4. Operaciones CRUD

En la Tabla 10, se puede apreciar los tiempos obtenidos una vez realizado las operaciones de CRUD en los dos gestores de base de datos. Dichos tiempos están expresados en milisegundos. Cabe mencionar que tanto en PostgreSQL como en Neo4j fue empleado un único nodo de prueba.

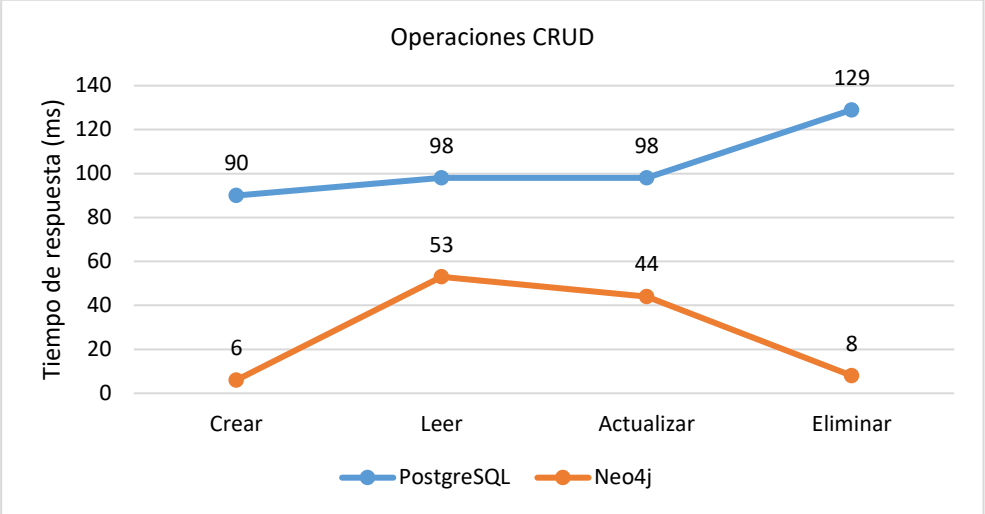
Tabla 10. Tiempos de respuesta de las Operaciones CRUD

Gestores de base de datos	Tiempo de Respuesta (ms)			
	Crear	**Leer**	**Actualizar**	**Eliminar**
PostgreSQL	90	98	98	129
Neo4j	6	53	44	8

Fuente: Los Autores

Al momento de realizar las operaciones CRUD existen variaciones de tiempo completamente diferentes entre los gestores. PostgreSQL presenta una variación en los tiempos de forma ascendente llegando a su pico máximo en la operación eliminar con un tiempo de 129ms. Por otra parte, Neo4j presenta un acenso hasta la operación (Leer) con un tiempo de 53ms, posterior a ello empieza el descenso llegando a un tiempo final de 8ms con la operación (Eliminar). El tiempo que demora Ne4oj en realizar las operaciones CRUD es 111ms, dando a notar la enorme optimización del modelo NoSQL orientado a grafos.

Figura 24. Gráfico estadístico de los tiempos de respuesta de las operaciones CRUD



Fuente: Los Autores

4.5. Discusión

El presente proyecto de investigación se basa en el análisis comparativo entre las bases de datos no relacional Neo4j y relacional PostgreSQL, teniendo como propósito identificar cuál de los gestores presenta un mejor rendimiento en el tiempo de respuesta, al aplicar el algoritmo de ruta más corta o también conocido como algoritmo de Dijkstra.

La investigación de (Fernandes & Bernardino, 2018) concluye que las bases de datos orientadas a gráficos brindan más rendimiento, flexibilidad y agilidad que las bases de datos no relacionales. Siendo Neo4j una de las bases de datos más potentes preferidas por los usuarios por su sencillez y por su potente lenguaje de consulta llamado Cypher. A su vez, (Moreno, Quintero, & Rueda, 2016) en su análisis entre MongoDB y Oracle sostiene que las bases de datos NoSQL, son más rápidas que las bases de datos SQL, en operaciones de inserción, actualización y borrado; debido a que las bases de datos relacionales realizan primero la verificación e integración de los datos, más sin embargo las BD relacionales tienen mejor desempeño en operaciones de consultas.

(Martinez & Aizemberg, 2015) en su investigación denominada: “Bases de datos de grafos con manejo de datos espaciales” realizan un análisis comparativo entre las bases de datos orientado a grafos Neo4j y ArangoDB, tomando en cuenta solo la operación de lectura y midiendo el tiempo de respuesta. El número de pruebas realizadas es igual a 4. Neo4j presenta un tiempo promedio de respuesta de 0.69 segundos, por su parte ArangoDB alcanza un tiempo promedio igual a 2.23 segundos, notándose una superioridad marcada a favor de Neo4j.

Partiendo de la afirmación donde (Moreno, Quintero, & Rueda, 2016) detallan que las bases de datos relacionales tienen mejor desempeño en operaciones de consultas. En esta investigación en base a los resultados obtenidos mediante el uso de operaciones CRUD muestra que la base de datos Neo4j a pesar de ser una base de datos no relacional el tiempo de respuesta en operaciones de lectura da un valor aproximado de 54% más rápido, en comparación a la base relacional PostgreSQL. (Figura 24)

Los resultados obtenidos en esta investigación, tienen una similitud con la investigación de (Martinez & Aizemberg, 2015) en la rapidez del manejo de datos espaciales por parte de Neo4j y los bajos tiempos de consulta frente a otras bases de datos.

Para esta investigación se optó por realizar pruebas en caliente, mientras existen otros procesos ejecutándose a la par, simulando una saturación de los recursos del computador con la finalidad de observar la variación en el tiempo de respuesta al resolver el algoritmo. De igual manera, se realizó pruebas en frío con la finalidad de eliminar el caché de la memoria y observar el desempeño de los gestores al resolver el algoritmo de ruta más corta.

En conclusión, las pruebas en caliente arrojaron un resultado aproximado del 77% más rápido, en la resolución del algoritmo de la ruta más corta a favor de la base de datos no relacional Neo4j. En las pruebas en frío, se probó que el gestor de base de datos Neo4j presenta un nivel de respuesta óptimo, aproximadamente 81 % más rápido que PostgreSQL al momento de resolver el algoritmo de la ruta más corta, tomando en cuenta que cada prueba se realizó con un número total de 62.817 nodos.

CAPÍTULO V

5. CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- Se estudió las diferentes arquitecturas de bases de datos. PostgreSQL presenta una arquitectura cliente/servidor y usa multiprocesos para garantizar la estabilidad del sistema, dado que la investigación simula un ambiente Big Data con datos geoespaciales, PostgreSQL debe incorporar herramientas adicionales, PostGIS convierte al sistema de administración de bases de datos en una base de datos espacial y pgRouting es una extensión para PostgreSQL que añade funcionalidades para análisis de redes y planificación de rutas. Por el contrario, Neo4j presenta dos arquitecturas; la primera, agrupación de clúster de alta disponibilidad, que garantiza al máximo la accesibilidad a los datos y la segunda, agrupación casual, que tiene como objetivo principal dotar de una mayor escalabilidad a la base de datos y brindar una mayor eficiencia en las operaciones de lectura. Para el tratamiento de grandes volúmenes de datos Neo4j no necesita ninguna herramienta adicional debido a que cuenta con sus propias librerías incorporadas dentro de su núcleo.
- Se elaboró una guía práctica paso a paso para la implementación de un ambiente Big Data del gestor de base de datos Neo4j. Dotando al lector de una herramienta de consulta de primera mano, a la cual pueda recurrir en futuras investigaciones. Con esta guía se pretende dar una idea más detallada del proceso para la creación de un escenario con datos masivos geoespaciales.
- Los datos cualificados utilizados en esta investigación constan aproximadamente de 62.817 nodos. Las pruebas de optimización en caliente realizadas a los gestores muestran una superioridad muy marcada debido a la rapidez de respuesta de Neo4j, con un tiempo promedio de ejecución del algoritmo Dijkstra de 122,1ms frente a 528,9ms de PostgreSQL. La diferencia de tiempos entre los gestores es de 406.8ms que representa un 77% más óptimo a favor de Neo4j. Para las pruebas de optimización en frío, Neo4j mantiene una tendencia mayor a las pruebas en caliente, con un tiempo promedio de respuesta de 456,2ms frente a 2399,9ms de PostgreSQL. La diferencia de tiempos entre dichos gestores es de 1943,7ms que representa un 81% de mayor eficiencia a favor de Neo4j.

- En la actualidad, Neo4j está siendo utilizada para el tratamiento de redes de transporte públicos, análisis de clima, redes sociales, etc. Basándonos en investigaciones anteriores y en esta investigación se ratifica a Neo4j como una opción viable para el tratamiento de datos masivos y resolución de algoritmos complejos.

5.2. RECOMENDACIONES

- Realizar un estudio más profundo acerca de las arquitecturas de PostgreSQL y Neo4j, dado que esta investigación tuvo un enfoque piloto con simulación de un escenario real. De ser el caso de emplear esta investigación en proyectos reales basarse en la documentación oficial.
- Se recomienda el uso de la guía práctica indexada en esta investigación, dado que Neo4j viene a ser un gestor orientado a grafos muy potente y por el incremento inimaginable de datos tiene una proyección muy buena a corto tiempo en todo el mundo. Además, la guía propuesta reúne herramientas utilizadas en varias investigaciones y sobre todo se centra en la documentación que se encuentra en la página oficial de (Neo4j, 2020).
- En esta investigación se pudo afirmar que Neo4j es el gestor más óptimo en tiempo de respuesta a la ejecución del algoritmo de ruta más corta, en comparación del gestor de base de datos relacional PostgreSQL. Como menciona (Martinez & Aizemberg, 2015) Neo4j brinda un soporte mucho más avanzado para el manejo de datos geográficos, posibilitando realizar consultas complejas de manera eficiente. En tal virtud, se recomienda el uso de la base de datos Neo4j con todas sus prestaciones y servicios.
- Este proyecto de investigación tuvo un enfoque marcado hacia dos gestores de bases de datos relacional y no relacional, pero, en el mercado existen un sin número de bases de datos como es el caso de Oracle Database 19c, que es de tipo relacional y ofrece rendimiento, escalabilidad, fiabilidad y seguridad de los datos. Por otro lado, está ArangoDB, que es una base de datos no relacional orientado a grafos proporcionando mucha flexibilidad y, además, admite fragmentación. Mediante este antecedente, se recomienda para futuras investigaciones incluir estos gestores, con el fin de analizar el desempeño y rendimiento que ofrecen.

BIBLIOGRAFÍA

Anaconda Inc. (2021). *Anaconda*. Obtenido de https://www.anaconda.com

Boeing, G. (04 de Julio de 2017). OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. *65*, 126-139. Computers, Environment and Urban Systems 65.

Camargo, J., Camargo, J., & Joyanes, L. (2015). Conociendo Big Data. *Facultad de Ingeniería, 24*(38), 69. Obtenido de https://www.redalyc.org/articulo.oa?id=413940775006

Cardona, M., Castrillón, O., & Tinoco, H. (2017). Determinación del Método Óptimo de Operaciones de Ensamble Bimanual con el Algoritmo de Dijkstra (o de Caminos Mínimos). *Información Tecnológica, 28*(4), 125-134. Obtenido de http://dx.doi.org/10.4067/S0718-07642017000400015

Castillo, J., Garcés, J., Navas, M., & Jácome, D. (2017). Base de Datos NoSQL: MongoDB vs. Cassandra en operaciones CRUD (Create, Read, Update, Delete). *Revista Publicando, 4*(11), 79-107. Obtenido de https://revistapublicando.org/revista/index.php/crv/article/view/398

Chávez, J. (2020). *CLIENTE PSQL DE POSTGRESQL*. Venezuela: IEASS, Editores.

Constantinov, C., Poteras, C., & Mihai, M. (2016). Performing real-time social recommendations on a highly-available graph database cluster. *2016 17th International Carpathian Control Conference (ICCC)*, 116-121.

Durán, J. W., Tandazo, E. J., Morales, M. R., & Morales, S. (2019). Rendimiento de bases de datos Columnares. *INGENIUS*, 47-58.

Fernandes, D., & Bernardino, J. (2018). Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. *7th International Conference on Data Science, Technology and Applications*, (pág. 377). Portugal. doi:10.5220/0006910203730380

Francis, N., Green, A., Guagliardo, P., Libkin, L., Plantikow, S., Lindaaker, T., . . . Selmer, P. (2018). Cypher: An Evolving Query Language for Property Graphs. *SIGMOD*, 1433-1445.

Gonzáles, J. (2015). *Utilización de las bases de datos relacionales en el sistema de gestión y almacenamiento de datos.* Ediciones Paraninfo SA.

Lopez, D., Montenegro, C., Toledano, O., & Vazquez, A. (2016). Método para generar la red social de usuarios del jabber.

Martinez, F., & Aizemberg, A. (2015). Bases de datos de grafos con manejo de datos espaciales. *1º Simposio Argentino de Grandes Datos, 44*.

Medina, J., Pineda, E., & Tellez, F. R. (1 de Abril de 2019). Requerimientos de software: prototipado, software heredado y analisis de documentos. *37(2)*. (U. d. Norte, Ed.) Ingenieria y Desarrollo.

Migani, S., Vera, C., & Lund, M. (2018). NoSQL: modelos de datos y sistemas de gestión de bases de datos. *XX Workshop de Investigadores en Ciencias de la Computación*, 225-228.

Moreno, F., Quintero, J., & Rueda, R. (2016). Una comparación de rendimiento entre Oracle y MongoDB. *Ciencia e Ingeniería Neogranadina*, *26*(1). Obtenido de https://www.redalyc.org/pdf/911/91145342002.pdf

Neo4j. (2020). *Neo Technology*. Obtenido de https://neo4j.com/

Open Street Map Wiki. (2020). *Open Street MAP*. Obtenido de https://wiki.openstreetmap.org/wiki/Portal:Press

Ordoñez, M., Ríos, J., & Castillo, F. (2017). *Administración de Bases de datos con PostgreSQL* (Vol. 19). 3Ciencias.

Parra, L. Y., & Vázquez, M. G. (2017). Probabilidad y Estadística. *Muestreo probabilístico y no probabilístico*.

Peñalvo, F. J. (2018). *Ciencia de la Computación e Inteligencia Artificial.* Salanabca, España.

pgRouting. (2021). *Import OSM data into pgRouting Database*. Obtenido de https://pgrouting.org/docs/tools/osm2pgrouting.html

PostGIS. (2021). *Acerca de PostGIS*. Obtenido de https://postgis.net/

PostgreSQL, G. d. (2020). *PostgreSQL*. Obtenido de https://www.postgresql.org/

Riaño, E., Toro, G., & Rico-Bautista, D. (2018). ÁRBOL DE CAMINOS MÍNIMOS: ENRUTAMIENTO, ALGORITMOS. *Revista Colombiana de Tecnologías de Avanzada, 1*(31). doi: https://doi.org/10.24054/16927257.v31.n31.2018.2780

Robles, D., Sánchez, M., Serrano, R., Adárraga, B., & Heredia, D. (2015). ¿Qué características tienen los esquemas SQL? *Investigación y Desarrollo en TIC, 6*(1), 40-44.

Rojas, O., Martinez, N., & Sanchez, M. (13 de Junio de 2018). Revisión sobre directrices prácticas para la calidad del modelado. (U. d. Informáticas, Ed.) La Habana, Cuba: Revista Cubana de Ciencias Informáticas .

Romeo, E., Martínez, G., & Rico, D. (2018). Árbol de caminos mínimos: Enrutamiento, Algoritmos Aproximados y Complejidad. *Revista Colombiana de Tecnologías de Avanzada, 1*(31), 16-17. doi:10.24054/16927257

Rubio, F., Vega, P., & Reyes, R. (2020). NoSQL contra SQL en la Administración de datos masivos: Un estudio empírico. *NnE Engineering*, 42. doi:10.18502/keg.v5i1.5917

Scifo, E. (16 de Noviembre de 2019). *Neo4j Dveloper Blog*. Obtenido de https://medium.com/neo4j/introducing-neomap-a-neo4j-desktop-application-for-spatial-data-3e14aad59db2

Torres, O., Sabater, S., Bravo, L., Martin, D., & García, M. (Enero-Marzo de 2019). Detección de anomalías en grandes volúmenes de datos. *Resvista Faculta de Ingeniería, 28*(50), 62-67. doi:https://doi.org/10.19053/01211129.v28.n50.2019.8793

Valverde, V., Portalanza, N., & Mora, P. (2019). Análisis descriptivo de base de datos relacional y no relacional. *Revista Atlante: Cuadernos de Educación y Desarrollo*, 3. Obtenido de https://www.eumed.net/rev/atlante/2019/06/base-datos-relacional.html

Vega, G., Villalobos, M., Acuña, L., & Oviedo, R. (2019). Una comparación de rendimiento entre bases de datos NoSQL: MongoDB y ArangoDB. *Tecnología En Marcha, 32*(3), 5-15. Obtenido de https://doi.org/10.18845/tm.v32i6.4223

Workshops. (2020). *Workshops de pgRouting de los FOSS4G*. Obtenido de https://workshop.pgrouting.org/

ANEXOS

ANEXO I

Script para el cálculo del algoritmo de la ruta más corta sin visualización de mapa en PostgreSQL.

```
SELECT * FROM pgr_dijkstra('SELECT gid as id,
        source,
        target,
        length_m AS cost
       FROM public.ways',
   79012, 35280,
   directed := false)
```

Donde:

source >> nodo inicial

target >> nodo final

length_m >> distancia o costo de la ruta en metros

directed = false >> sentido de circulación de la ruta.

Script para el cálculo del algoritmo de la ruta más corta con visualización de mapa en PostgreSQL.

```
WITH ruta as (SELECT * FROM pgr_dijkstra(
   'SELECT gid as id,
        source,
        target,
        length_m AS cost
       FROM public.ways',
   79012, 35280,
   directed := false))
SELECT ruta.*, b.the_geom
FROM ruta
LEFT JOIN public.ways b ON ruta.edge = b.gid ;
```

ANEXO II

Script para el cálculo de la ruta más corta en Neo4j.

```
MATCH (startNode:Node {osmid: "1258864665"})
MATCH (endNode:Node {osmid: "1209700380" })
CALL gds.alpha.shortestPath.stream(
        "projected_graph",
       {
         startNode: startNode,
         endNode: endNode,
         relationshipWeightProperty: "length",
          writeProperty: 'sssp'
       }
)
YIELD nodeId, cost
RETURN gds.util.asNode(nodeId).osmid AS name, cost
```

Donde:

startNode= nodo inicial

endNode= nodo final

Nota: Neo4j trabaja con propiedades, por esta razón osmid viene a ser los valores que toman los nodos inicio y fin para el cálculo de la ruta más corta, siendo los mismos puntos de PostgreSQL, pero con diferente nomenclatura.

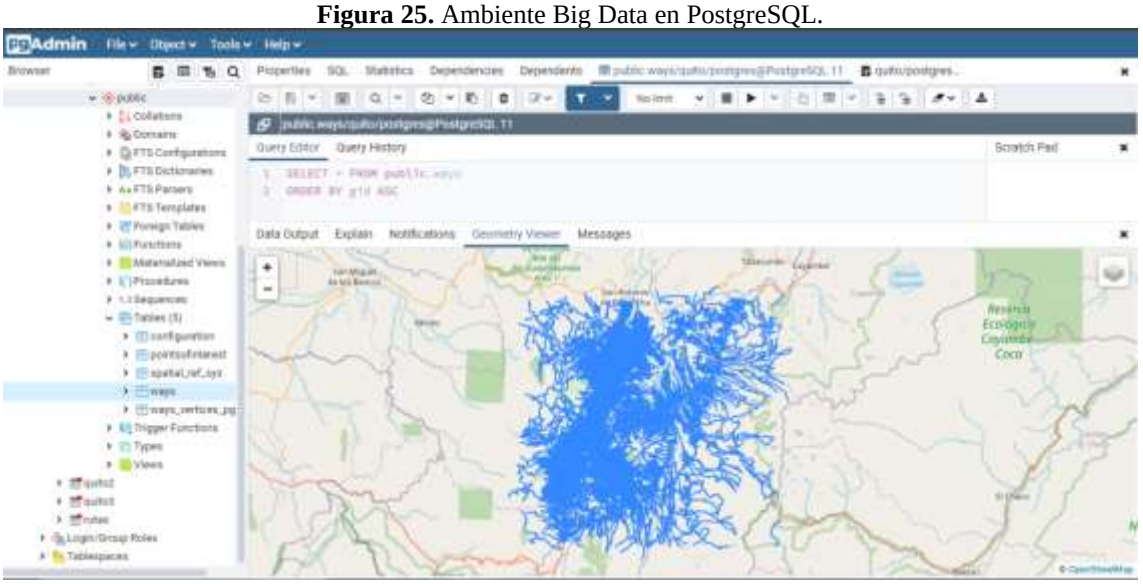
Script para la visualización de la ruta a través del plugin Neomap

```
MATCH (startNode:Node {osmid: "1258864665"})
MATCH (endNode:Node {osmid: "1209700380" })
CALL gds.alpha.shortestPath.stream(
       "projected_graph",
       {
          startNode: startNode,
          endNode: endNode,
          relationshipWeightProperty: "length"
       }
)
YIELD nodeId, cost
WITH gds.util.asNode(nodeId) AS node
RETURN node.x AS longitude, node.y AS latitude
```

ANEXO III

Ambiente Big Data creado en PostgreSQL con las rutas viales de la ciudad de Quito

Figura 25. Ambiente Big Data en PostgreSQL.

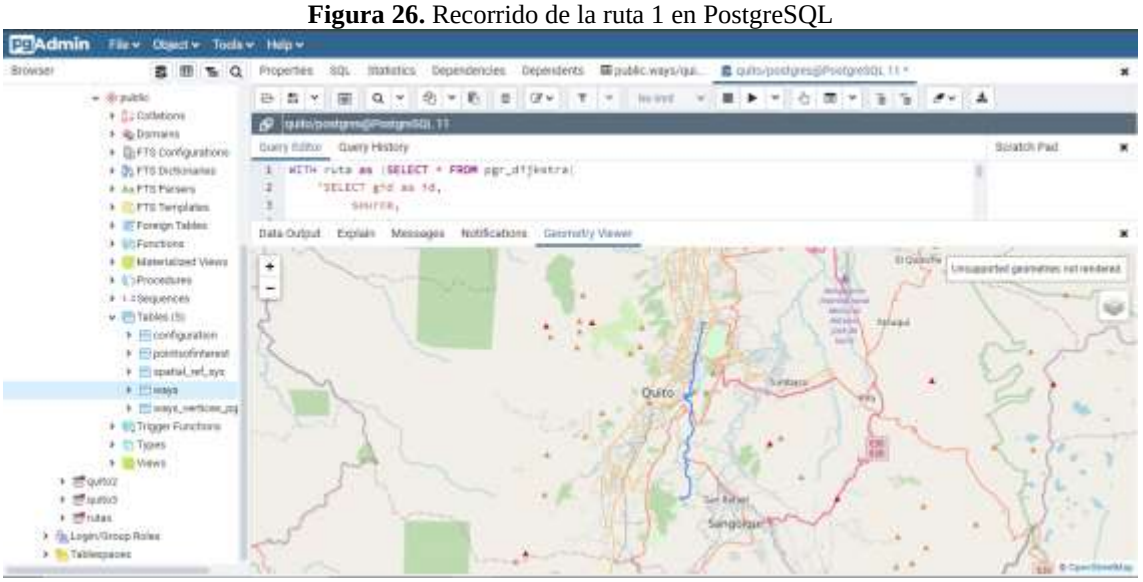


Fuente: Los Autores

Para la creación del ambiente Big Data en PostgreSQL, se utilizó un total 62.817 registros que fueron obtenidos de OpenStreetMap.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 1)

Figura 26. Recorrido de la ruta 1 en PostgreSQL

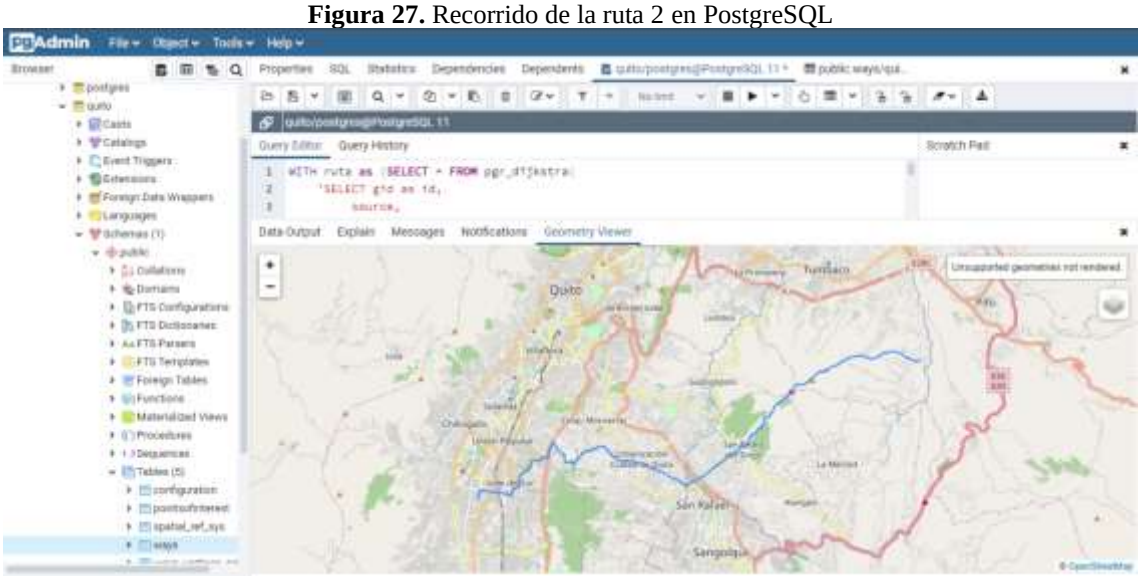


Fuente: Los Autores

En la prueba 1, se utilizaron los nodos (7526, 447) como inicial y final respectivamente obteniendo un costo total de la ruta de 20994.08m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 2)

Figura 27. Recorrido de la ruta 2 en PostgreSQL

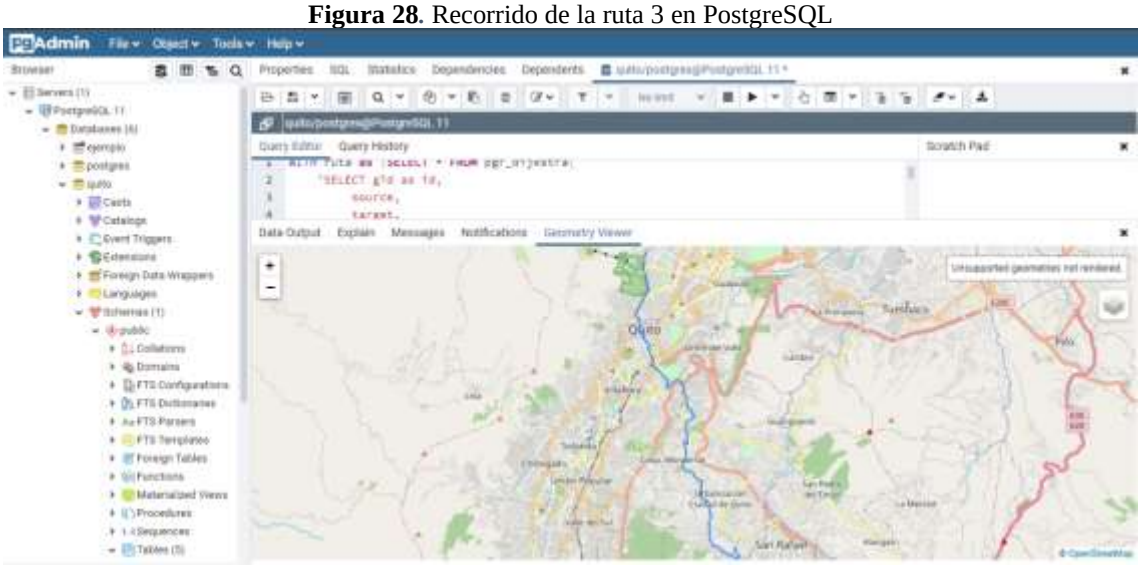


Fuente: Los Autores

En la prueba 2, los nodos utilizados fueron (39350, 4170) como inicial y final respectivamente obteniendo un costo total de la ruta de 28517.32m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 3)

Figura 28. Recorrido de la ruta 3 en PostgreSQL

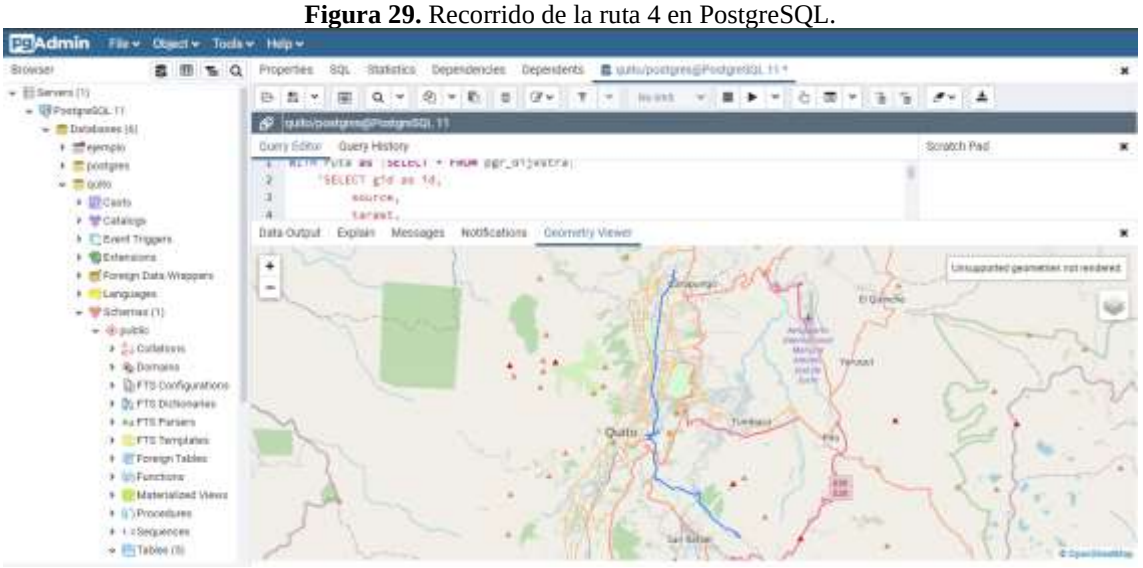


Fuente: Los Autores

Los nodos (16218, 21127) fueron utilizados para la prueba 3, obteniendo un costo total de la ruta de 19759.35m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 4)

Figura 29. Recorrido de la ruta 4 en PostgreSQL.

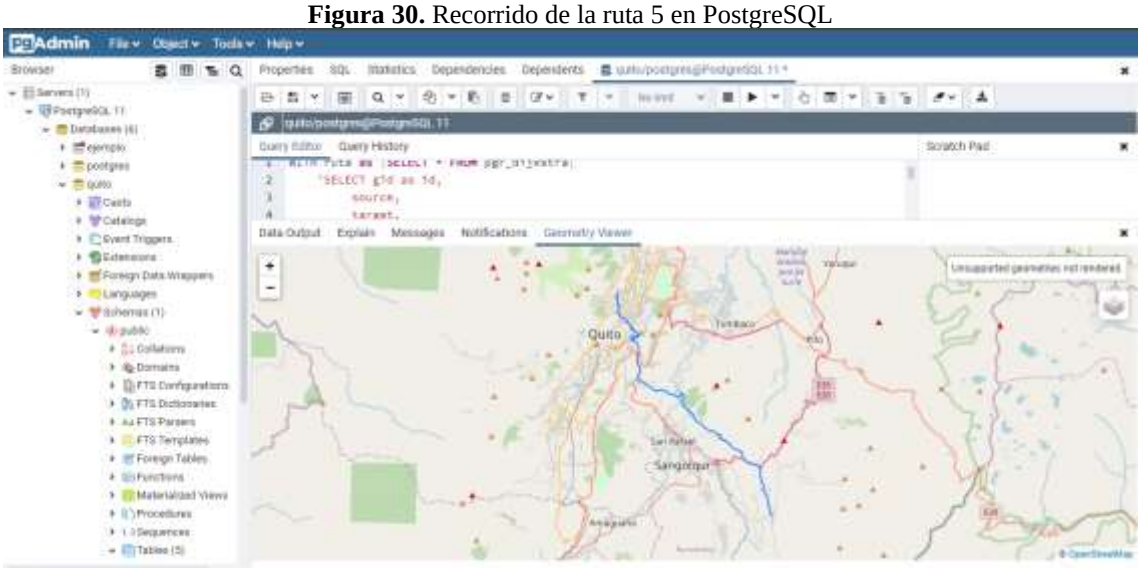


Fuente: Los Autores

La ruta 4, presenta un costo total de 33820.20m, teniendo a los nodos (6386, 17378) como inicial y final respectivamente.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 5)

Figura 30. Recorrido de la ruta 5 en PostgreSQL

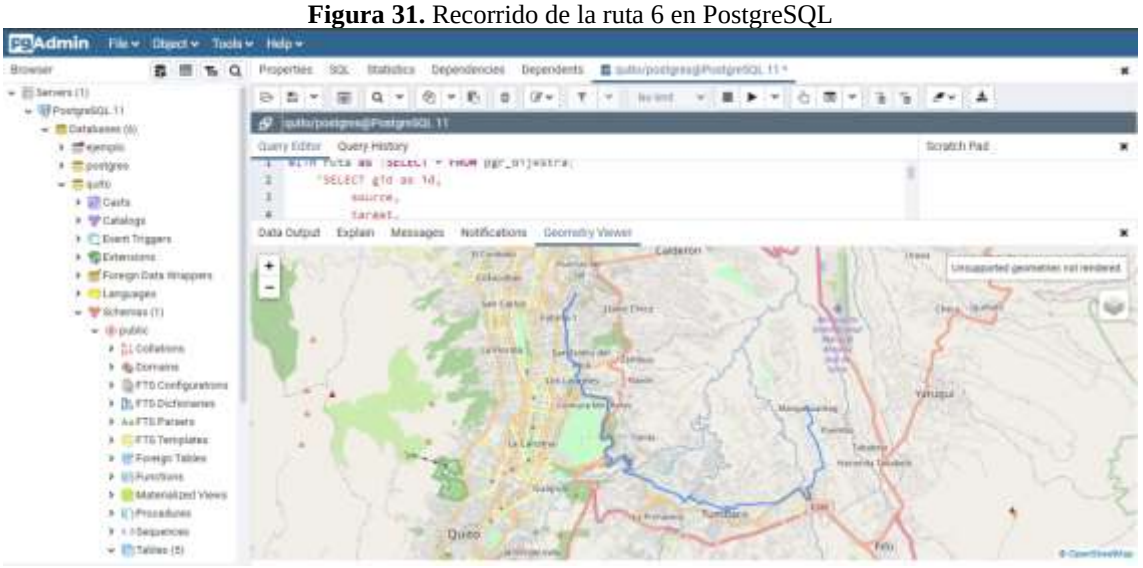


Fuente: Los Autores

Los nodos (17779, 458) fueron utilizados para la ruta 5. El costo total es igual a 32349.58m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 6)

Figura 31. Recorrido de la ruta 6 en PostgreSQL

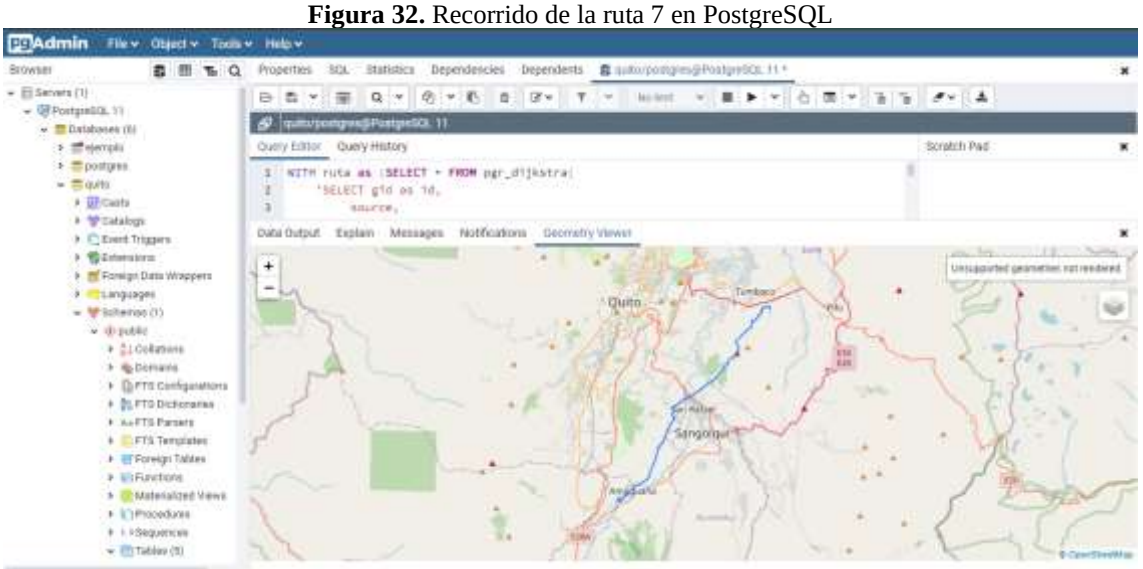


Fuente: Los Autores

En la prueba 6, se utilizaron los nodos (17958, 12036) como inicial y final. Se obtuvo un costo total de la ruta igual a 28896.76m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 7)

Figura 32. Recorrido de la ruta 7 en PostgreSQL

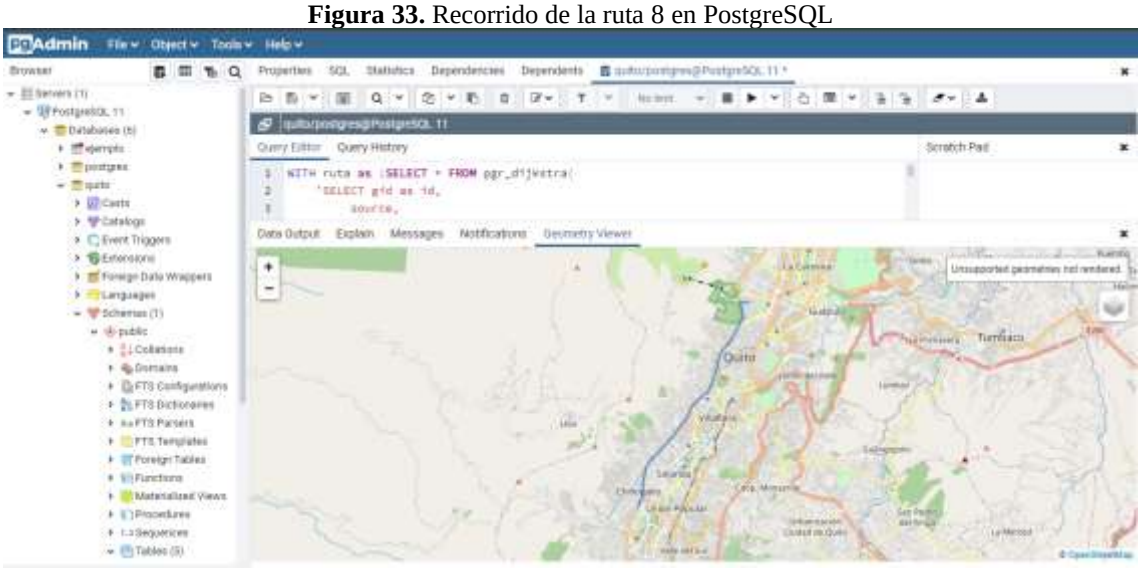


Fuente: Los Autores

Los nodos (52509, 58510) fueron utilizados en la ruta 7. El costo total de la ruta es de 27762.61m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 8)

Figura 33. Recorrido de la ruta 8 en PostgreSQL

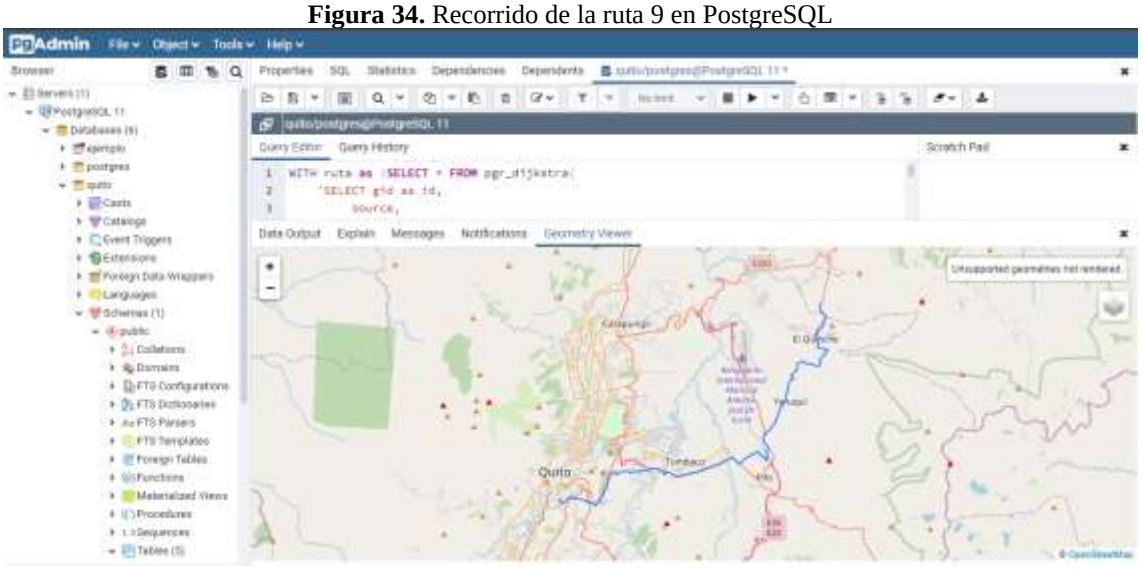


Fuente: Los Autores

En la ruta 8, los nodos (10450, 5346) fueron utilizados como inicial y final, generando un costo total de la ruta de 11512.98m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 9)

Figura 34. Recorrido de la ruta 9 en PostgreSQL

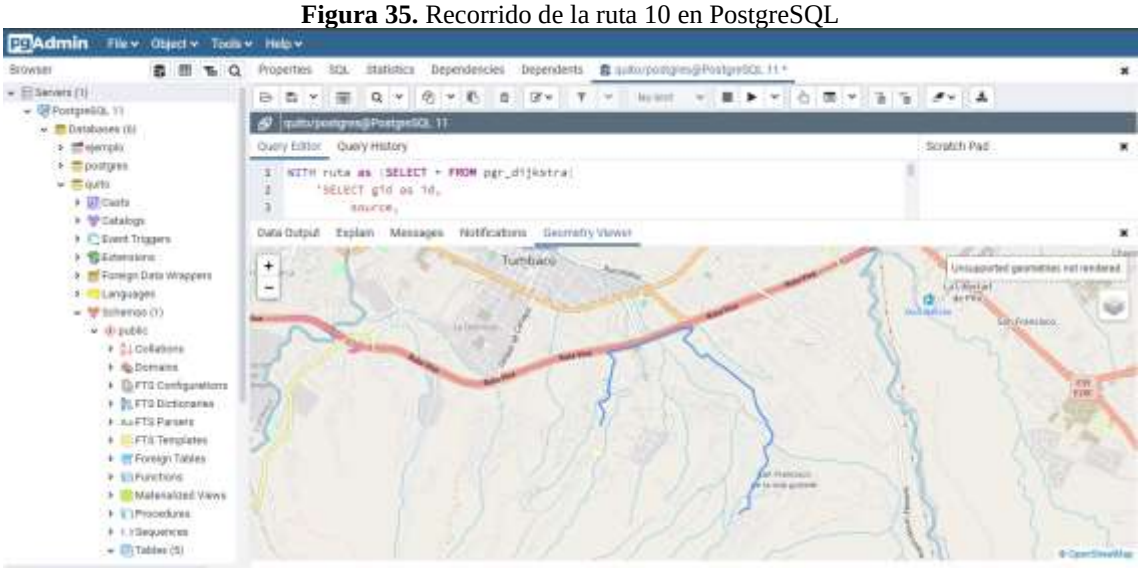


Fuente: Los Autores

En la prueba 9, los nodos utilizados son (50315, 51071) como inicial y final. El costo total de la ruta es igual a 45289.05m.

Resultado de la ejecución del algoritmo de Dijkstra (Prueba 10)

Figura 35. Recorrido de la ruta 10 en PostgreSQL



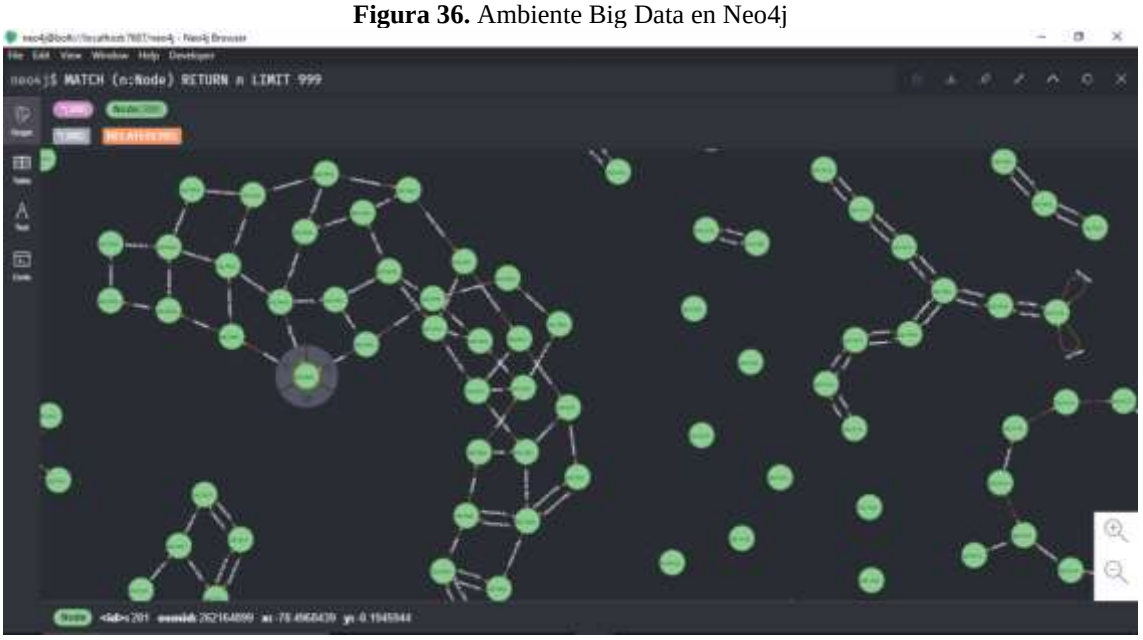
Fuente: Los Autores

El costo total de la ruta 10 es igual a 4906.51m, teniendo a los nodos (18037, 73251) como inicial y final.

ANEXO IV

Ambiente Big Data creado en Neo4j con las redes viales de la ciudad de Quito

Figura 36. Ambiente Big Data en Neo4j



Fuente: Los Autores

Los datos utilizados para la creación del ambiente Big Data en Neo4j es igual a 62.817 nodos. Para la visualización de cada trazo de ruta se utilizó el plugin Neomap en Neo4j.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 1)

Figura 37. Recorrido de la ruta 1 en Neo4j

Fuente: Los Autores

En la ruta 1, el costo total es de 21165.99m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 2)

Figura 38. Recorrido de la ruta 2 en Neo4j

Fuente: Los Autores

En la ruta 2, se obtuvo un costo total de 28671.97m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 3)

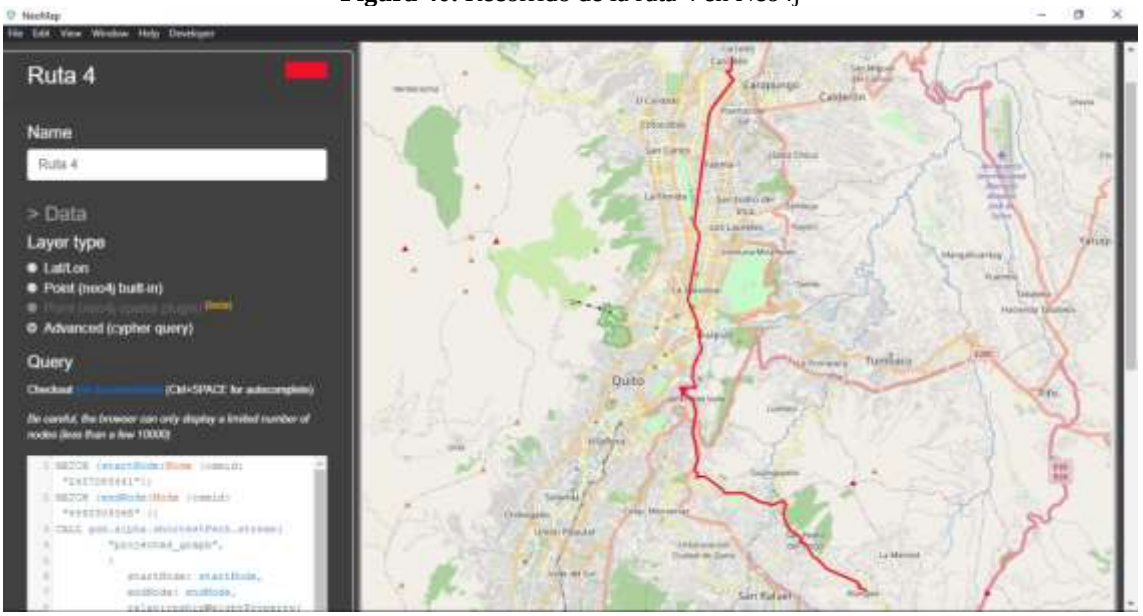
Figura 39. Recorrido de la ruta 3 en Neo4j

Fuente: Los Autores

En la ruta 3, se obtuvo un costo total de 20195.94m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 4)

Figura 40. Recorrido de la ruta 4 en Neo4j

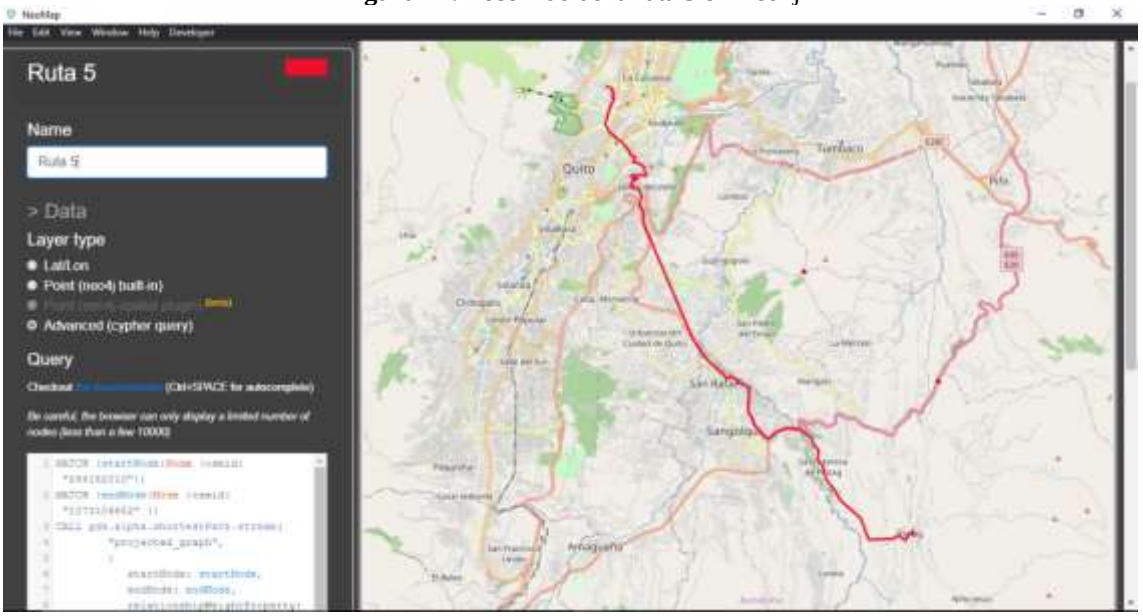


Fuente: Los Autores

En la ruta 4, se obtuvo un costo total de 34209.71m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 5)

Figura 41. Recorrido de la ruta 5 en Neo4j

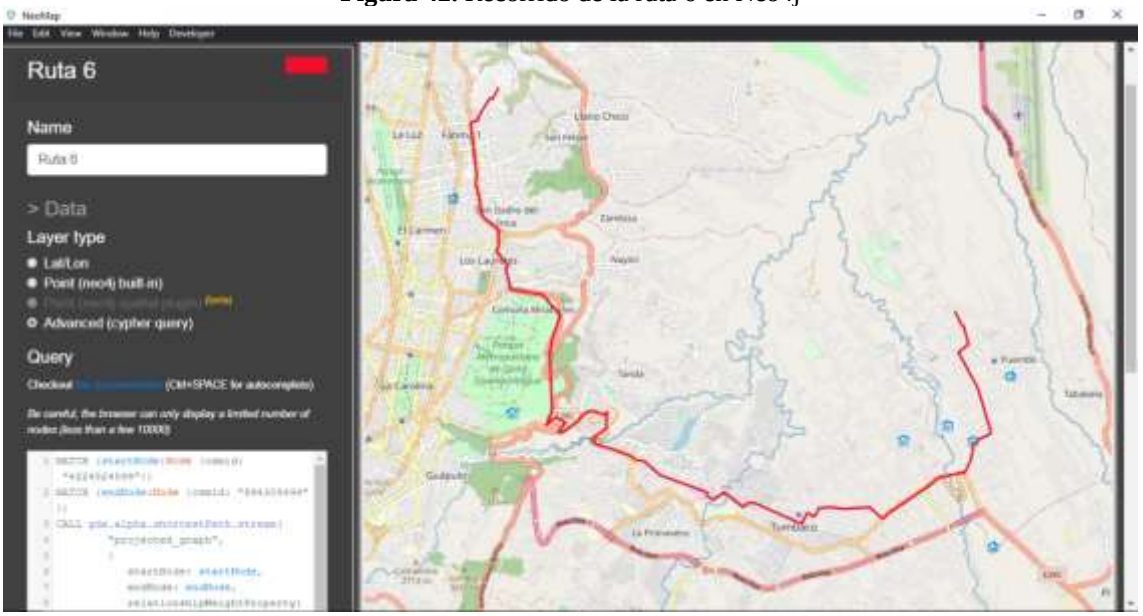


Fuente: Los Autores

En la ruta 5, se obtuvo un costo total de 35626.77m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 6)

Figura 42. Recorrido de la ruta 6 en Neo4j



Fuente: Los Autores

En la ruta 6, se obtuvo un costo total de 29912.04m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 7)

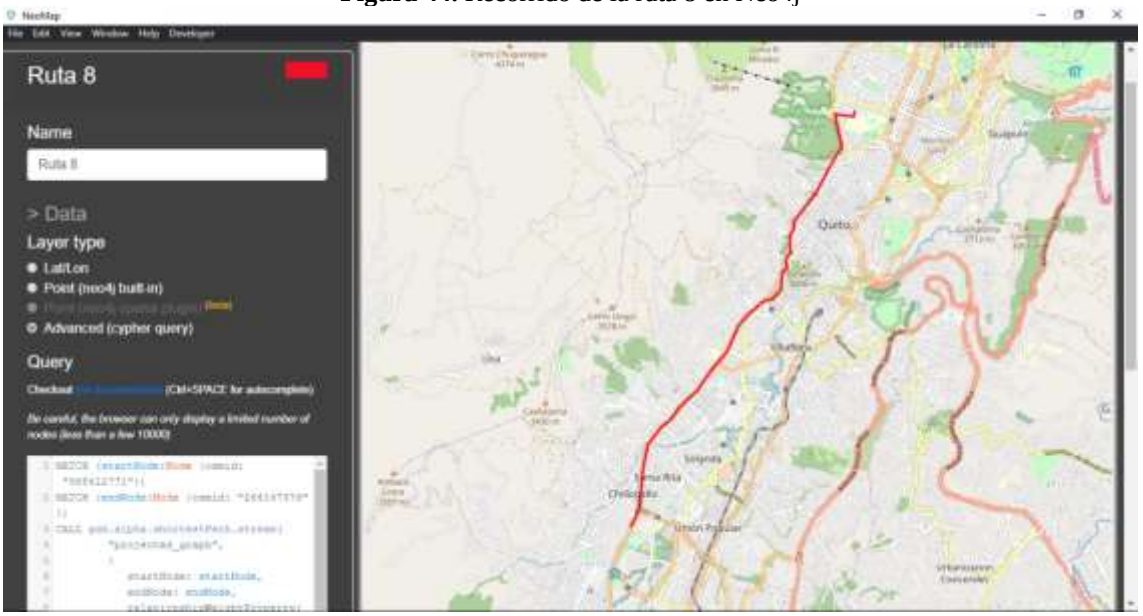
Figura 43. Recorrido de la ruta 7 en Neo4j

Fuente: Los Autores

En la ruta 7, se obtuvo un costo total de 27874.71m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 8)

Figura 44. Recorrido de la ruta 8 en Neo4j



Fuente: Los Autores

En la ruta 8, el costo total es de 11873.31m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 9)

Figura 45. Recorrido de la ruta 9 en Neo4j

Fuente: Los Autores

En la ruta 9, se obtuvo un costo total de 45663.56m.

Resultado de la ejecución del algoritmo Dijkstra en Neo4j (Prueba 10)

Figura 46. Recorrido de la ruta 10 en Neo4j

Fuente: Los Autores

En la ruta 10, se obtuvo un costo total de 5142.35m.

GUÍA DE IMPLEMENTACIÓN DE UN AMBIENTE BIG DATA EN NEO4J

UNIVERSIDAD NACIONAL DE CHIMBORAZO

"GUÍA DE IMPLEMENTACIÓN DE UN AMBIENTE BIG DATA EN NEO4J"

Autores:

Alex Tuapanta
Jhonatan Montenegro

Docente:

Ing. Ximena Quintana López, PhD.

Facultad de Ingeniería
Escuela de Sistemas y Computación

Índice

Acerca del Manual

- ✓ Propósito

El presente documento tiene como objetivo ser una guía básica para la operación e implementación del ambiente Big Data en Neo4j usando data sets de OpenStreetMap; permitiendo al lector del mismo adquirir las destrezas y conocimientos indispensables para el manejo adecuado de la base de datos. Este documento sirve como una herramienta de consulta de primera mano en caso de cualquier duda que presente el lector.

- ✓ Herramientas previas necesarias
  - ➢ Tener instalado Java SE 8 o Java SE 11
  - ➢ Entorno de Data Science Anaconda
  - ➢ Paquete osmnx

Instalación de Neo4j

La instalación de Neo4j es un proceso bastante sencillo dependiendo el sistema operativo elegido. En esta guía se detallan los pasos tanto para Ubuntu 20.04 como para Windows 10 con las siguientes especificaciones.

Características de los equipos

Ubuntu

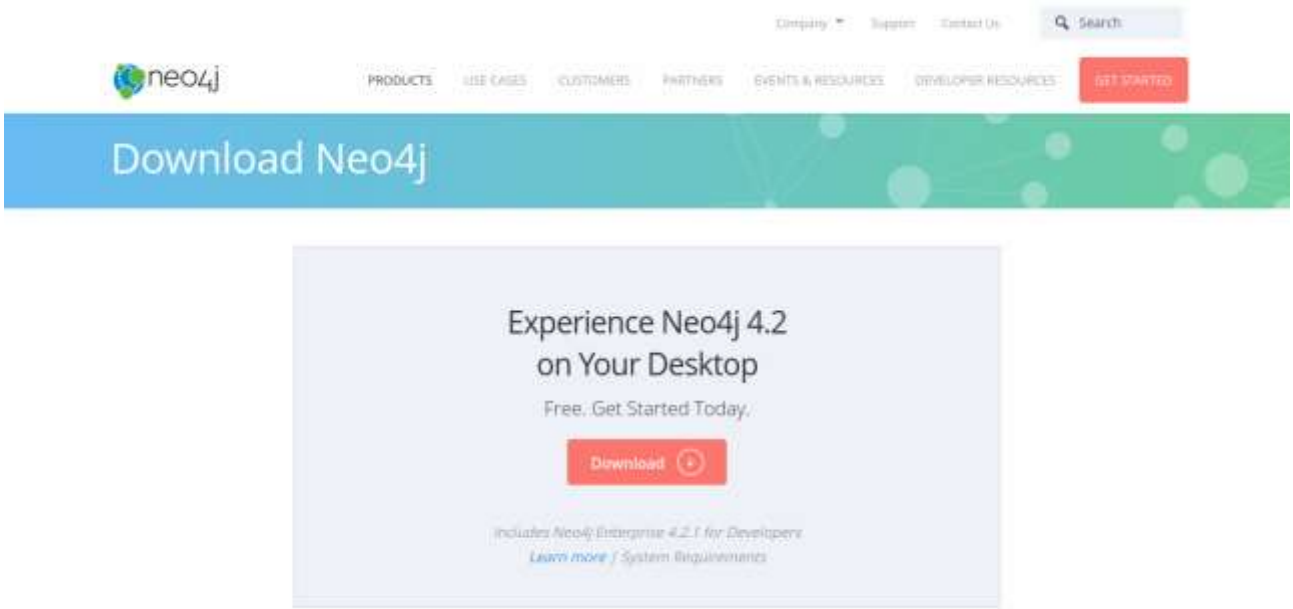
Procesador	Intel® Core™ i5-3210M CPU @ 2.50GHz × 4
Memoria Ram	6 GB
Sistema Operativo	Ubuntu 20.04

Windows

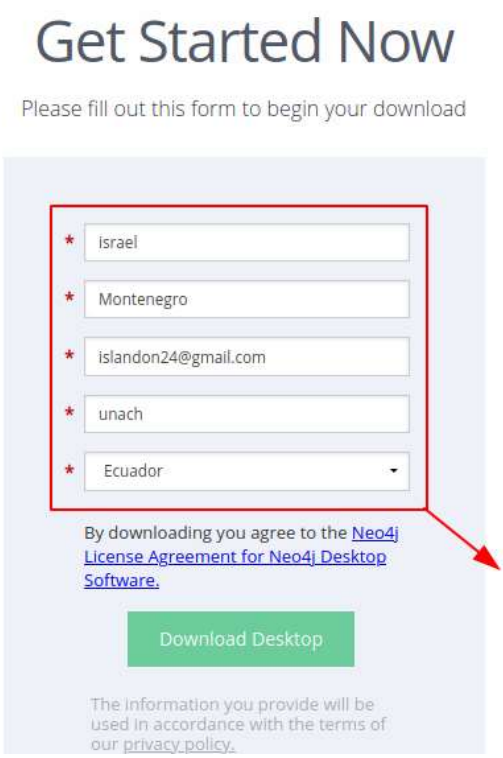
Procesador	Intel® Core™ i5-7200U CPU @ 2.50GHz 2.70GHz
Memoria Ram	8 GB
Sistema Operativo	Windows 10 Home

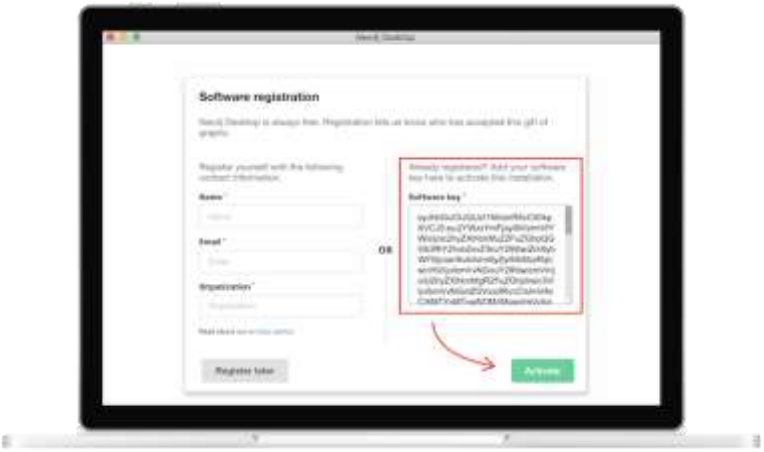
Paso 1

Descargar el instalador de la página oficial de Neo4j, dependiendo el sistema operativo será un archivo con extensión diferente:
Windows: archivo con extensión (.exe)
Ubuntu y derivados: archivo con extensión (.Appimage)



Paso 2





Para acceder a la descarga y a la llave de activación de la aplicación, complete el formulario. posteriormente aparecerá una nueva pantalla con una clave de activación para Neo4j, copiar y guardar la clave de activación para su posterior uso.

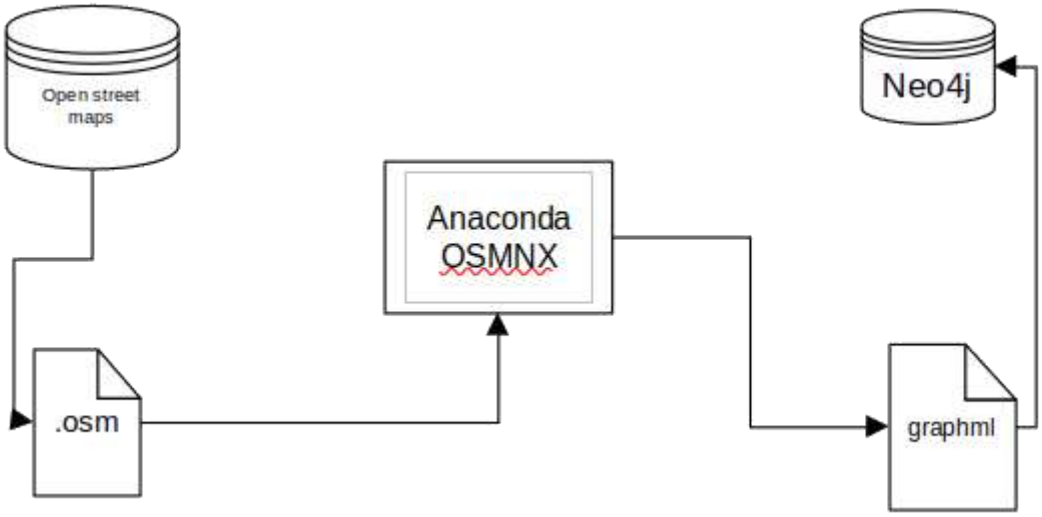
Una vez descargado, ejecutar el archivo, aceptar los acuerdos de licencia y pegar la clave de activación de Neo4j.

Nota: En Linux para la ejecución del archivo descargado siga la siguiente secuencia:

- ✓ Abrir la terminal de Linux
- ✓ Navegar al directorio donde se descargó el archivo
- ✓ Ejecutar chmod +x “Nombre del archivo. appimge ”

Esquema de Datos

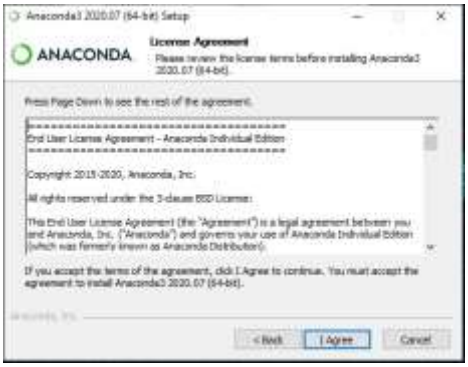
Para la descarga del conjunto de datos de Open Street Maps se utiliza el entorno de anaconda, en conjunto con el editor de código Spyder.



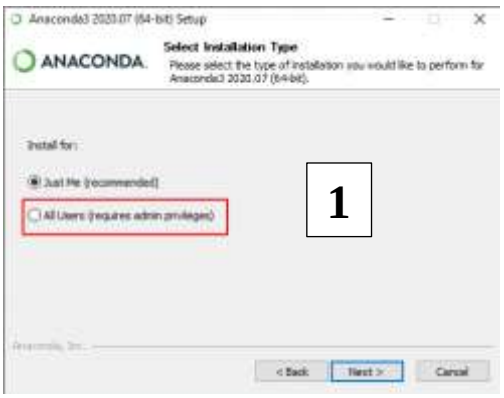
Instalación de Anaconda

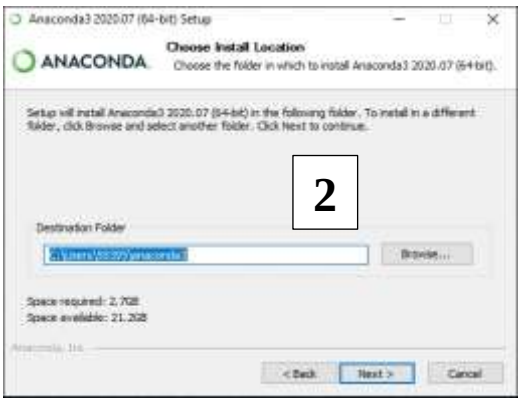
Pasos:

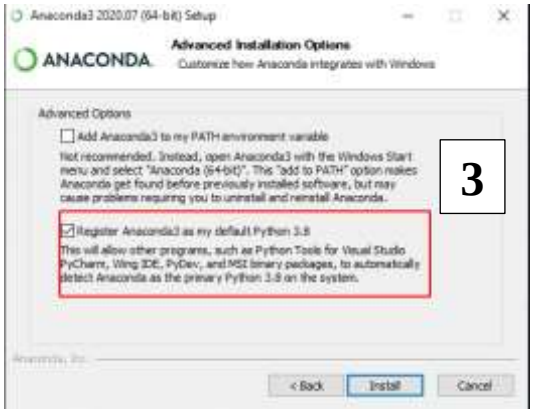
- ✓ Descargar el entorno de Anaconda de la página oficial.
- ✓ Ejecutar el instalador: .exe para Windows y .sh para Linux
- ✓ Seguir los siguientes pasos



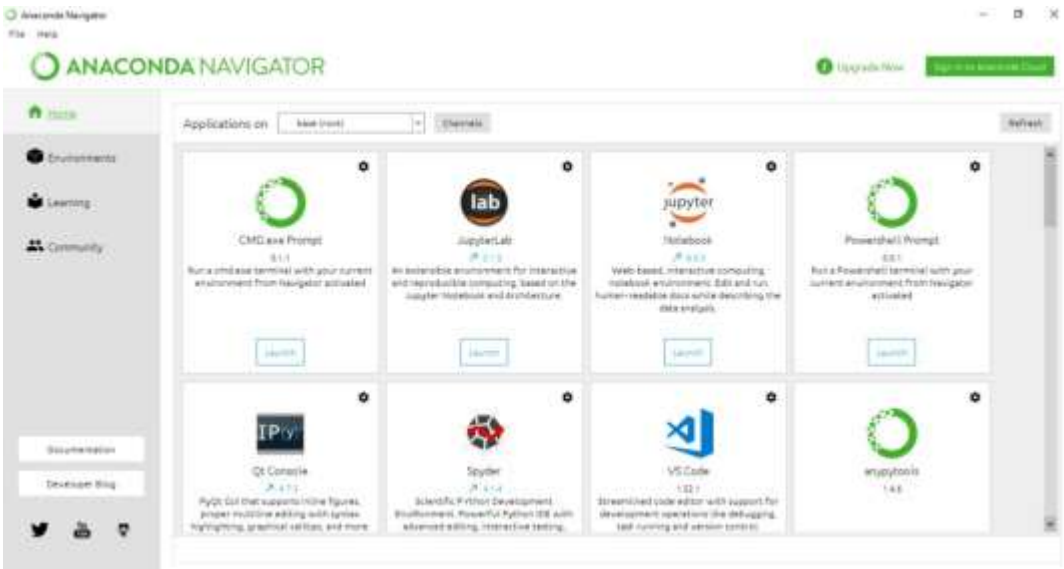
En las dos primeras ventanas del asistente de instalación, presione el botón “Next” de la primera imagen, posteriormente en la segunda ventana aceptar los términos y condiciones del software.





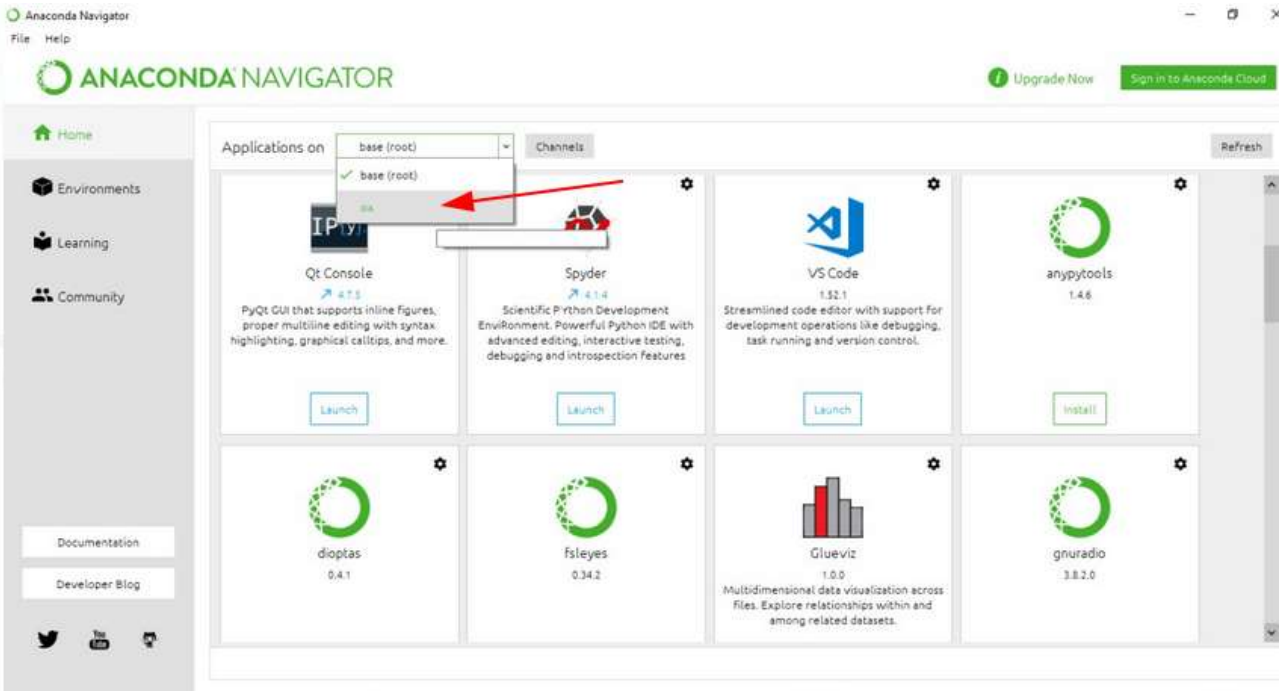


En la ventana número **"1"** importante seleccionar la opción "**All users**" y click en "**Next**"; esto permitirá al entorno funcionar con privilegios en los usuarios creados en la PC. En caso de no estar en la cuenta de administrador, la ventana siguiente **"2"** mostrará la ruta de instalación en el ordenador, click en "**Next**". Finalmente, en la imagen **"3"** seleccionar la opción "Registrar Anaconda" y click en "Install".



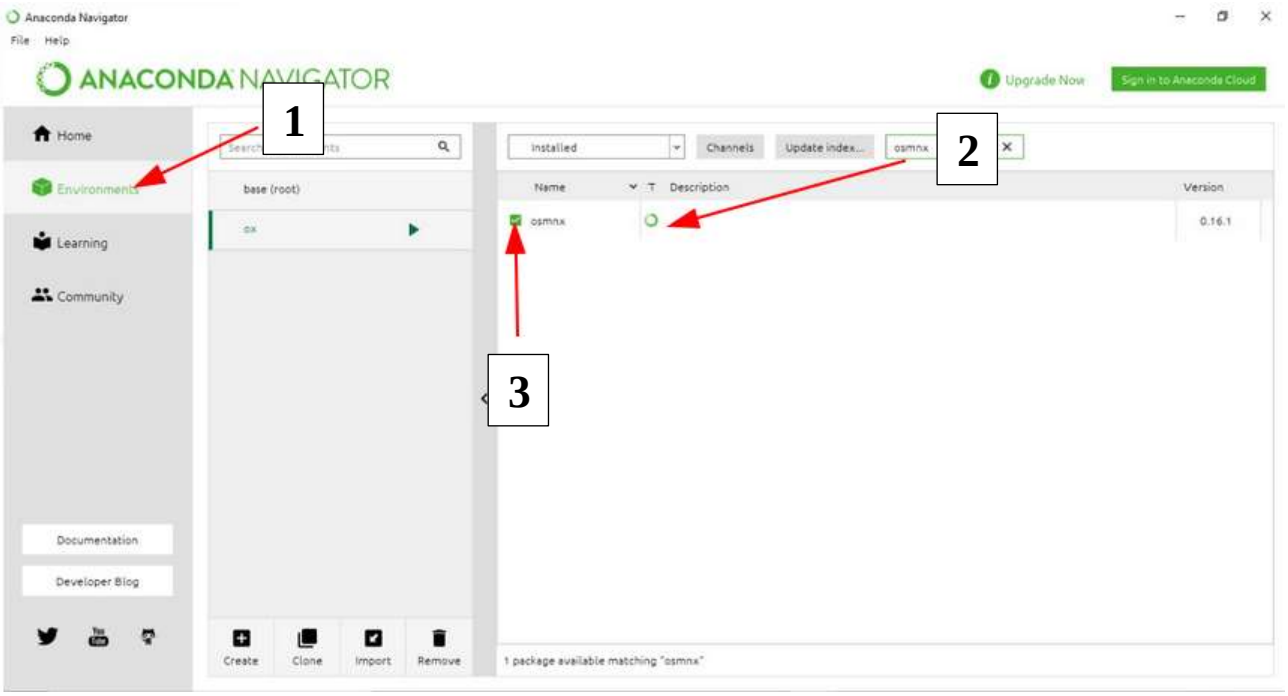
Preparación del Entorno de Anaconda

Paso 1



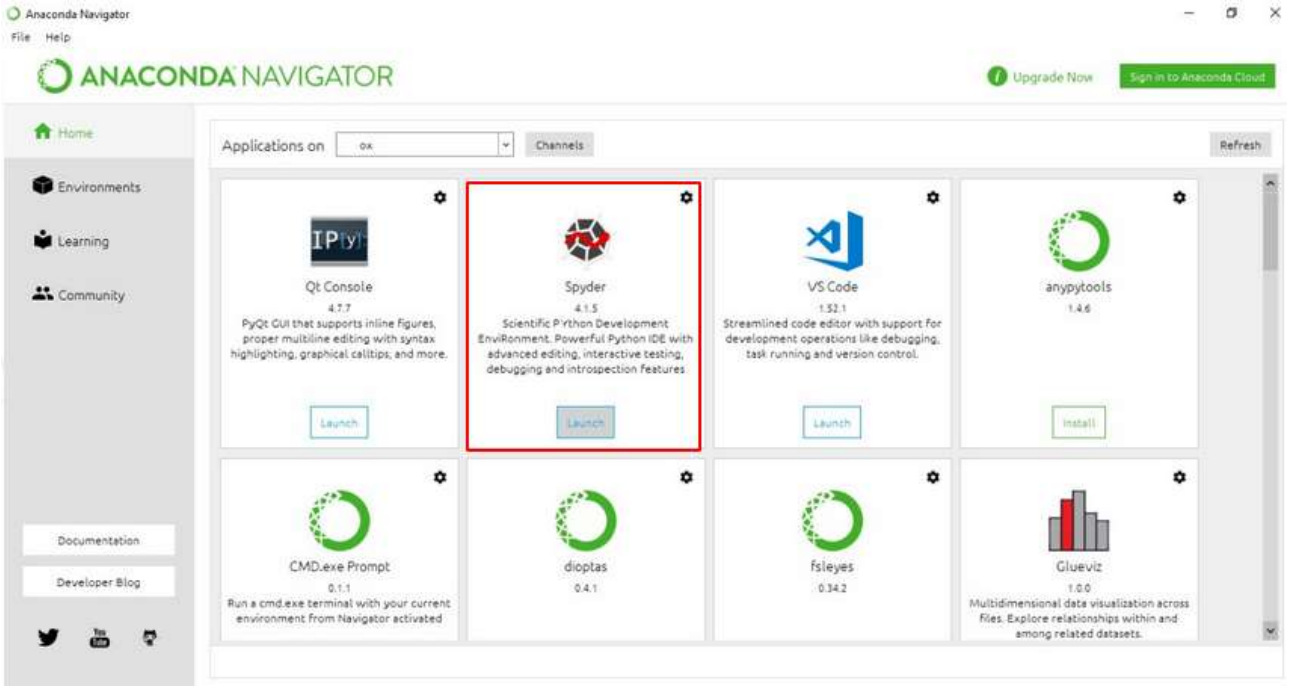
Una vez instalado el entorno, ejecutar el lanzador de la aplicación “Anaconda Navigator”, al abrirse el entorno seleccionar en el casillero de “Aplications on” la opción de “ox” como muestra la imagen.

Paso 2



Posteriormente seleccionar en el menú lateral la opción **Enviroments**, luego en el cuadro de búsqueda de la flecha 2, escribir el nombre del paquete **osmnx**, y finalmente activarlo marcando el recuadro.

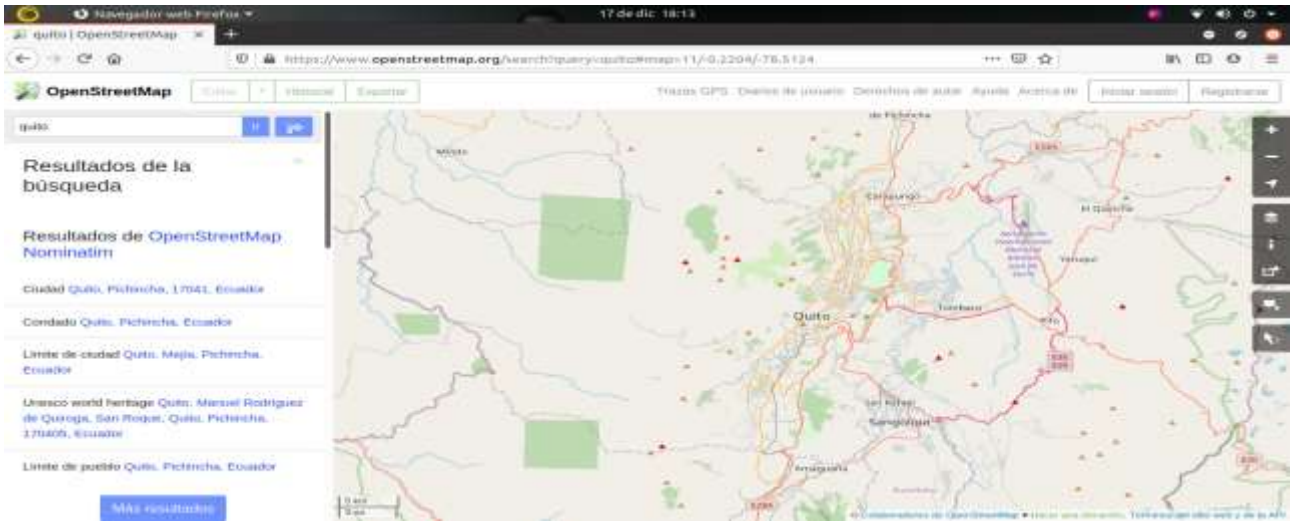
Paso 3



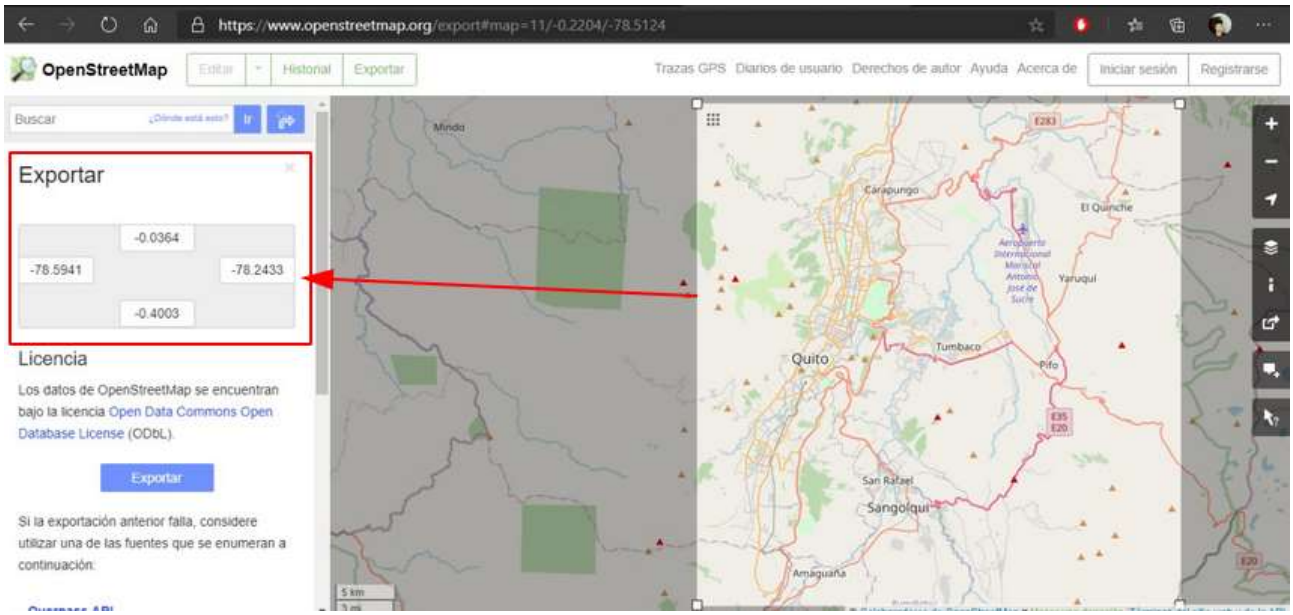
Una vez que este activado el paquete de python osmnx, proceder a instalar el editor Spyder, click en "Install", esto activara todas las herramientas necesarias para la obtención de datos desde OpenStreetMap.

Descarga de Datos

Para la preparación de datos; OpenStreetMap brinda una solución bastante rápida y eficaz. Mediante su herramienta web permite obtener y marcar un área en específico de cualquier parte del mundo, según sea la necesidad o caso de estudio.

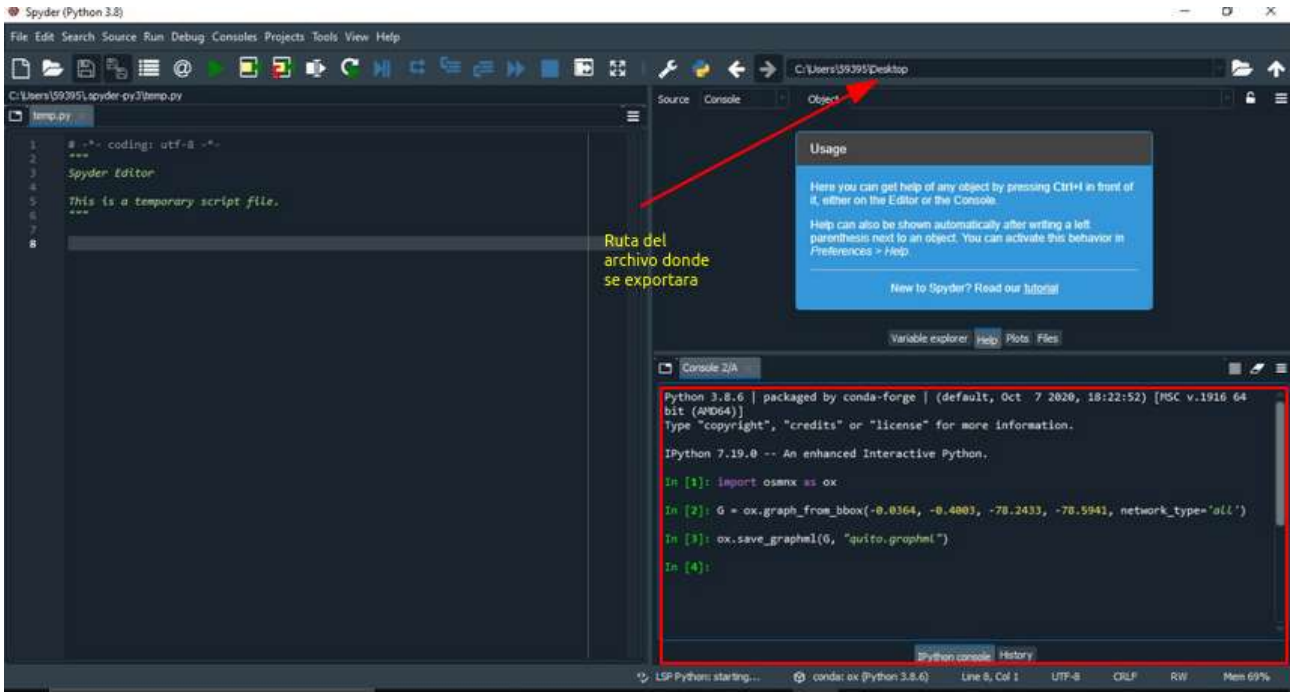


Paso 1



Con la herramienta de exportación se podrá obtener las coordenadas del área interesada, en este caso es la ciudad de Quito. En el panel izquierdo se muestra las coordenadas que forman la zona marcada en el mapa de la derecha. Estas coordenadas representan el Norte, Sur, Este y Oeste respectivamente.

Paso 2



Una vez que se tenga las coordenadas Norte, Sur, Este y Oeste, abrir el programa de Spyder desde Anaconda Navigator, ir al recuadro de la línea de comandos y escribir las siguientes sentencias una por una:

1. import osmnx as **ox**
2. G = ox.graph_from_bbox(**-0.0364, -0.4003, -78.2433, -78.5941**, network_type = ‘all’)
3. ox.save_graphml(G, “quito.graphml”)

Explicación:

La primera línea de comando importa el módulo osmnx y lo guarda en una variable, en este ejemplo es la variable “ox”.

La segunda línea de comando, utiliza la función del paquete osmnx “ox.graph_from_bbox (N, S, E, O, network_type=“all”)” esto ayudará a crear un área en forma de un rectángulo. Los valores obtenidos (N, S, E, O) en el paso 1 reemplazar en la función, network_type=“all” le dice a osmnx que tipo de red vial se descargará, es decir descargará todos los tipos de red vial (caminando, bicicleta, vehículo) de la ciudad o área seleccionada.

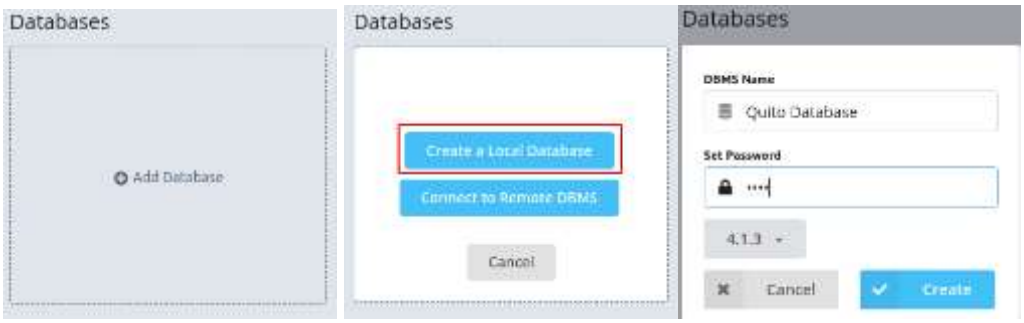
La tercera línea de comando será la encargada de guardar y transformar el archivo de formato osm a graphml, mismo que es compatible con Neo4j, obteniendo así la misma estructura del formato origen.

Paso 3

Verificar el archivo en la ruta seleccionada, debe tener el mismo nombre con extensión. graphml.

Preparación de Neo4j

Paso 1



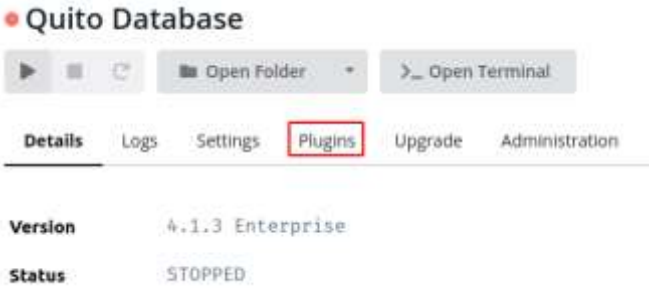
Para crear una base de datos local, click en “Add Database”, luego click en “Create a Local Database”, posteriormente escribir el nombre de la base de datos y establecer una contraseña, finalmente click en “Create”.

Paso 2

Para activar los plugins APOC y Graph Data Science Library necesarios para el manejo de datos y grafos en Neo4j siga la siguiente secuencia.

a) Presionar los tres puntos que aparece junto a la base de datos creada y dar click en “Manage”.

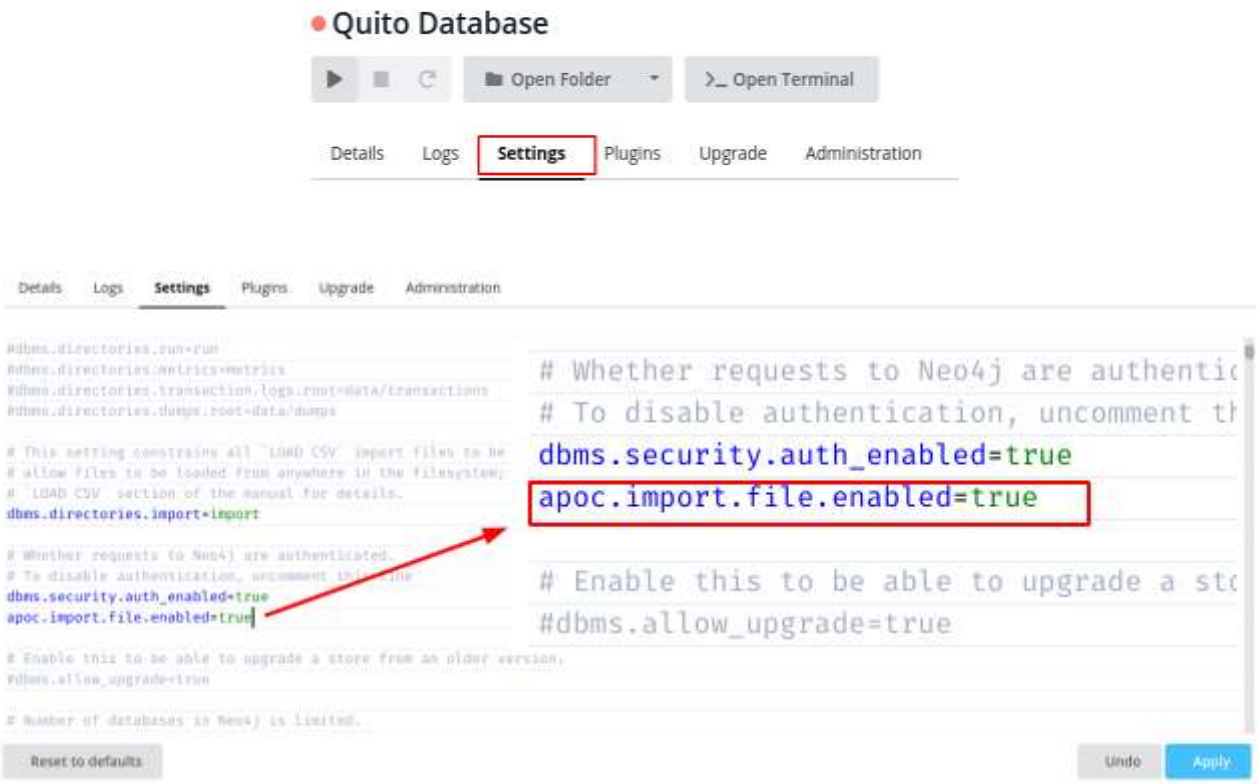
b) Seleccione la opción Plugins.



Paso 3

Se muestra una ventana con los diferentes plugins que cuenta Neo4j, seleccione “Install” en el plugin APOC y en Graph Data Science Library, esperar a que se instale automáticamente.

Paso 4



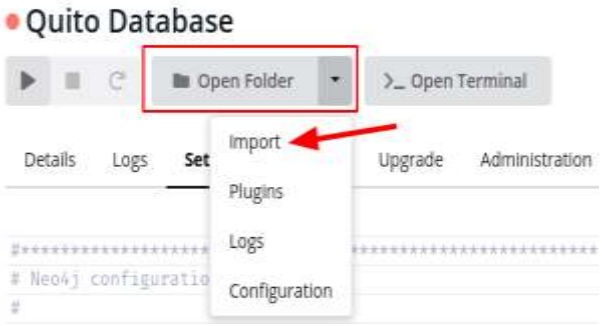
Luego de activar los plugins, se debe activar la importación de datos a través del plugin APOC, presione en la opción Settings y en la parte inferior se mostrará un archivo en texto plano, con todas las configuraciones de Neo4j, aquí añadir la siguiente línea de código “apoc.import.file.enabled=true”, finalmente click en “Apply” para guardar los cambios.

Paso 5

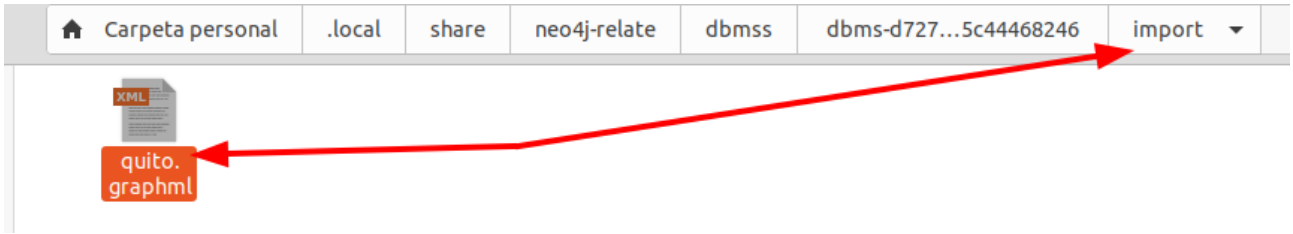
Copiar el archivo quito.graphml dentro de la carpeta import de Neo4j.

a) Dar click en los tres puntos junto a la base de datos y posteriormente presione en "Manage".

b) Presione en la flecha del menú desplegable y seleccione la opción "import".



Una vez dentro de la carpeta import en Neo4j, pegar el archivo con la extensión ".graphml".



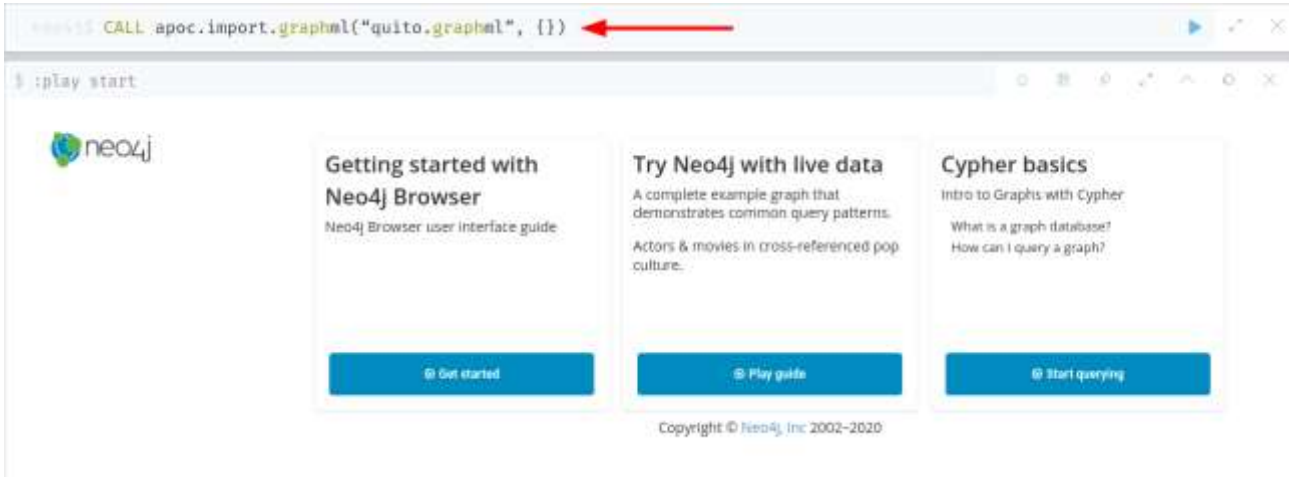
Importar Datos a Neo4j

Paso 1

Click en “Start” para iniciar la base de datos. Presione “Open” para lanzar el browser integrado en Neo4j.

Una vez iniciada la base de datos, en la CLI del browser ejecute el siguiente comando:

- ✓ “CALL apoc.import.graphml(“quito.graphml”, {}) ”

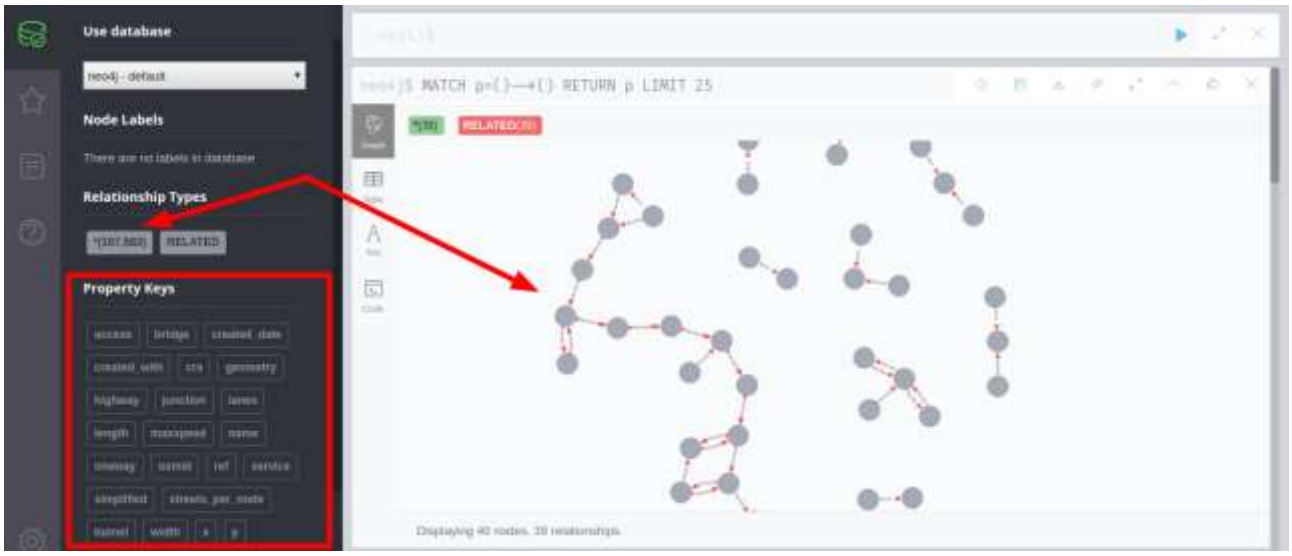


Este comando importa el archivo .graphml y forma los grafos automáticamente en la base de datos.

Paso 2

Al terminar de ejecutar la importación, se muestra un mensaje con todos los datos creados, como la cantidad de nodos, relaciones y propiedades, así también el tiempo que tarda el gestor en cargar todos los datos y crear la estructura.

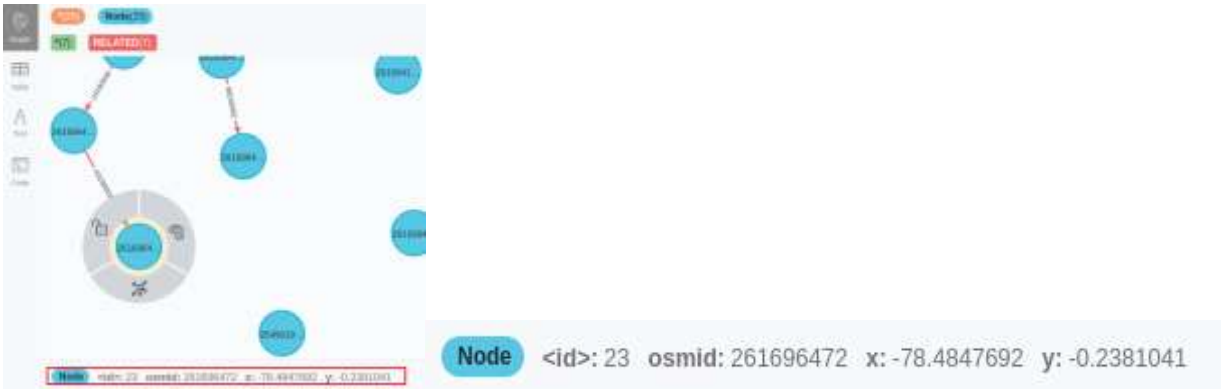
Paso 3



Para visualizar los grafos creados, presione sobre el número de nodos como se muestra en el gráfico. El límite máximo de nodos que muestra Neo4j es de 999. Neo4j tiene un apartado llamado “Property Keys” que también sirve para trabajar con las diferentes propiedades de los grafos.

Paso 4

Finalmente, para poder trabajar con los nodos asigne una etiqueta a todos los nodos, en la CLI de Neo4j ejecutar el comando “MATCH (n) SET n:Node” el cual busca todos los nodos (n) y les asigna un nombre, en este caso el nombre de la etiqueta es Node.



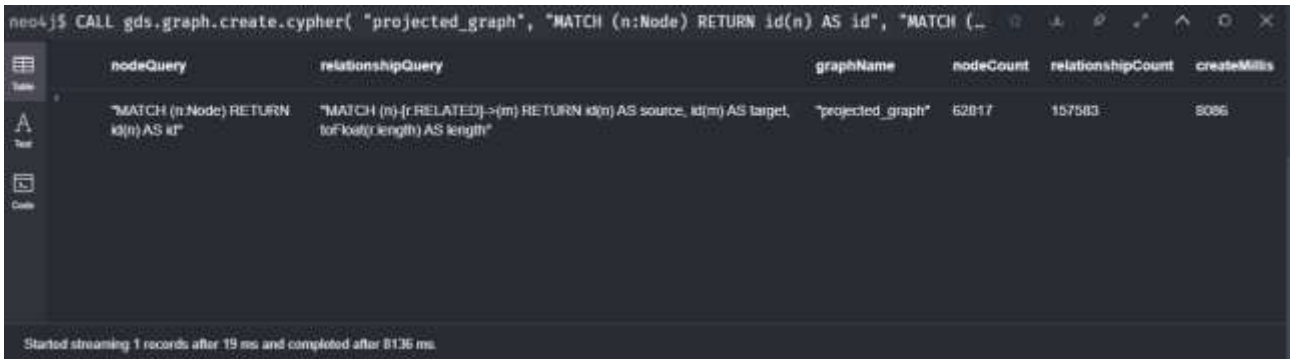
Ejecución del algoritmo de la ruta más corta

Paso 1

Crear un gráfico proyectado de la base de datos. Para ello en la CLI del browser ejecute las siguientes líneas de código.

```
CALL gds.graph.create.cypher(
        "projected_graph",
        "MATCH (n:Node) RETURN id(n) AS id",
        "MATCH (n)-[r:RELATED]→(m) RETURN id(n) AS source, id(m) AS target, toFloat(r.length) AS length"
)
```

En la imagen siguiente se aprecia las propiedades creadas a través del gráfico proyectado de la base de datos.



Paso 2

En la CLI del browser ejecute las siguientes líneas de código. El algoritmo normal muestra toda la ruta recorrida desde el nodo inicial hasta el nodo final.

```
MATCH (startNode:Node {osmid: "7101890259"})
MATCH (endNode:Node {osmid: "304292133" })
CALL gds.alpha.shortestPath.stream(
        "projected_graph",
        {
          startNode: startNode,
          endNode: endNode,
          relationshipWeightProperty: "length",
           writeProperty: 'sssp'
        }
)
YIELD nodeId, cost
RETURN gds.util.asNode(nodeId).osmid AS name, cost
```

Una vez ejecutado el algoritmo normal se genera una tabla con los atributos nombre y costo. El costo total de la ruta se encuentra en la última fila de la tabla.