Universidad Nacional Abierta y a Distancia

Vicerrectoría Académica y de Investigación

Curso: Big Data Código: 202016911

Tarea 4 # Almacenamiento y Consultas de Datos en Big Data

Jhon Héctor Pinzon Rodríguez

Número del grupo: 202016911_87

Frank Rodríguez Achury

Tutor

2025

Contenido

Introducción.

El crecimiento exponencial de los datos en entornos digitales ha impulsado la necesidad de adoptar tecnologías de almacenamiento y procesamiento capaces de gestionar información con altos niveles de volumen, variedad y velocidad. En este contexto, las bases de datos NoSQL han adquirido un papel fundamental en la arquitectura de sistemas modernos orientados al análisis de grandes volúmenes de datos, permitiendo superar las limitaciones de los modelos relacionales tradicionales.

El presente documento desarrolla un análisis comparativo de los principales tipos de bases de datos NoSQL clave-valor, documentales, orientadas a columnas y grafos destacando sus características, ventajas, limitaciones y casos de uso más relevantes dentro del ecosistema del Big Data.

Adicionalmente, se implementa un caso práctico utilizando MongoDB, en el cual se diseña y construye una base de datos documental para un catálogo de comercio electrónico. Sobre este entorno se ejecutan consultas básicas, consultas avanzadas con operadores y un conjunto de operaciones de agregación para el cálculo de estadísticas, permitiendo evaluar la utilidad de este modelo en escenarios reales de análisis y gestión de datos.

El trabajo integra conceptos teóricos con ejercicios prácticos de aplicación, fortaleciendo la comprensión del estudiante sobre los modelos NoSQL y su pertinencia en soluciones tecnológicas contemporáneas. Asimismo, demuestra la importancia de seleccionar adecuadamente el tipo de base de datos según las necesidades del proyecto, considerando aspectos como escalabilidad, flexibilidad esquemática, patrones de acceso y requerimientos de consistencia.

Objetivo General.

Analizar los modelos de bases de datos NoSQL y aplicar los conceptos estudiados mediante el diseño e implementación de una base de datos en MongoDB para un caso práctico de comercio electrónico, con el propósito de comprender su estructura, funcionamiento, ventajas, limitaciones y aplicabilidad dentro de entornos de Big Data.

Objetivos Específicos.

1. Identificar las características, ventajas y limitaciones de los principales modelos de bases de datos NoSQL clave-valor, documentales, orientadas a columnas y grafos a partir de una revisión teórica y comparativa.

2. Diseñar una base de datos documental en MongoDB adaptada a un entorno de comercio electrónico, definiendo colecciones, documentos y relaciones implícitas entre los datos.

3. Implementar el modelo de datos diseñado mediante la creación de colecciones, inserción de documentos y construcción de consultas utilizando operadores básicos, avanzados y pipelines de agregación.

4. Analizar los resultados obtenidos a partir de las consultas ejecutadas en MongoDB, evaluando su eficiencia, pertinencia y utilidad para el procesamiento de información en escenarios reales.

5. Integrar los hallazgos teóricos y prácticos en un documento académico que permita evidenciar el aprendizaje y la aplicabilidad de los modelos NoSQL en sistemas modernos de gestión y análisis de datos.

Fase 1. Investigación: Realizar un ensayo comparando los diferentes tipos de bases de datos NoSQL (Clave - valor, documentos, columnas, grafos). Discuta las ventajas e inconvenientes de cada tipo, así como sus casos de uso más apropiados.

Bases de Datos NoSQL: Un Análisis Comparativo desde la Perspectiva de la Ciencia de Datos y la Ingeniería de Big Data.

La explosión del volumen, variedad y velocidad de los datos en la era digital ha precipitado una transformación fundamental en los paradigmas de almacenamiento y procesamiento de información. Las bases de datos relacionales tradicionales, diseñadas bajo los principios ACID (Atomicity, Consistency, Isolation, Durability) y fundamentadas en el modelo relacional de Codd han demostrado ser insuficientes para atender las demandas contemporáneas de escalabilidad horizontal, flexibilidad esquemática y procesamiento distribuido que caracterizan al ecosistema del Big Data (Gutiérrez Hernández & Ibarra Limas, 2024). En este contexto, las bases de datos NoSQL emergen no como un reemplazo absoluto, sino como una respuesta arquitectónica especializada que privilegia la disponibilidad, la tolerancia a particiones y la capacidad de procesamiento masivo por encima de la consistencia inmediata.

El teorema CAP formulado por Eric Brewer en el año 2000 y posteriormente formalizado por Gilbert y Lynch (2002), establece que ningún sistema distribuido puede garantizar simultáneamente consistencia, disponibilidad y tolerancia a particiones, obligando a los arquitectos de datos a realizar compromisos estratégicos según las necesidades específicas de cada aplicación (Carrascal & Varona, 2024). Como señalan Gutiérrez Hernández e Ibarra Limas (2024), las bases de datos NoSQL representan diversas implementaciones de estos compromisos, materializadas en cuatro modelos fundamentales: clave#valor, documentos, columnas anchas y grafos. Cada uno de estos paradigmas responde a necesidades específicas dentro del espectro de

la ciencia de datos moderna, desde el almacenamiento de sesiones de usuario hasta la analítica de redes sociales complejas y la gestión de flujos de datos de Internet de las Cosas (IoT).

La relevancia de comprender estas arquitecturas trasciende el ámbito puramente técnico, constituyendo un requisito fundamental para investigadores y profesionales que desarrollan sistemas de Machine Learning, plataformas de analítica en tiempo real y aplicaciones de inteligencia artificial que operan sobre volúmenes masivos de datos heterogéneos. Según Cattell (2011), la elección apropiada del sistema de almacenamiento puede impactar dramáticamente el rendimiento, la escalabilidad y los costos operativos de las aplicaciones de Big Data. Este ensayo examina comparativamente los cuatro modelos NoSQL principales, analizando sus fundamentos arquitectónicos, sus ventajas y limitaciones desde una perspectiva ingenieril, y sus aplicaciones específicas en el contexto de la investigación aplicada y la ciencia de datos contemporánea.

Bases de Datos Clave#Valor: Simplicidad y Rendimiento Extremo

El modelo clave#valor representa la arquitectura más elemental dentro del ecosistema NoSQL, fundamentándose en una estructura de diccionario distribuido donde cada elemento de información se almacena y recupera mediante un identificador único. Miranda, Oña, Mendoza y Chango (2023) realizaron una evaluación comparativa exhaustiva de bases de datos NoSQL clave#valor en entornos de creación de aplicaciones, demostrando que esta simplicidad arquitectónica se traduce en ventajas operativas significativas, particularmente en escenarios que demandan latencias mínimas y throughput máximo. Sistemas como Redis y Amazon DynamoDB ejemplifican esta filosofía, ofreciendo operaciones de lectura y escritura con complejidad temporal O (1) y capacidades de escalamiento horizontal prácticamente ilimitadas.

Desde la perspectiva de la ingeniería de datos, las bases clave#valor destacan por su capacidad para implementar cachés distribuidas de alto rendimiento, almacenar sesiones de usuario en aplicaciones web de gran escala y gestionar contadores en tiempo real para sistemas de métricas y telemetría (Carrascal & Varona, 2024). La investigación desarrollada por DeCandia et al. (2007) sobre Amazon Dynamo demostró cómo este modelo puede alcanzar disponibilidad del 99.9995% mediante técnicas de replicación eventual y hashing consistente. En el contexto de la ciencia de datos, Miranda et al. (2023) destacan que este modelo es aprovechado para el almacenamiento de embeddings vectoriales en aplicaciones de procesamiento de lenguaje natural, donde la velocidad de recuperación de representaciones densas es crítica para la inferencia en tiempo real. Asimismo, en arquitecturas de Machine Learning distribuido, las bases clave#valor sirven como almacenamiento intermedio para parámetros de modelos y gradientes durante el entrenamiento paralelo, minimizando la latencia de comunicación entre nodos computacionales.

No obstante, esta simplicidad arquitectónica conlleva limitaciones inherentes que restringen su aplicabilidad en contextos analíticos complejos. La ausencia de capacidades de consulta estructurada, la imposibilidad de realizar operaciones relacionales y la carencia de indexación secundaria hacen que estos sistemas sean inadecuados para análisis exploratorios o consultas ad#hoc sobre conjuntos de datos multidimensionales. Como documenta Stonebraker (2010) en su análisis crítico de los sistemas NoSQL, el modelo clave#valor sacrifica expresividad semántica en favor de rendimiento bruto, resultando óptimo únicamente cuando los patrones de acceso están perfectamente definidos y la estructura de las consultas es predecible y uniforme. Esta observación es corroborada por Miranda et al. (2023), quienes enfatizan que la evaluación del tipo de base de datos debe considerar no solo el rendimiento bruto, sino también la complejidad de los patrones de consulta requeridos por la aplicación.

Bases de Datos de Documentos: Flexibilidad Esquemática para Datos Semiestructurados

Las bases de datos orientadas a documentos representan una evolución arquitectónica que preserva el rendimiento del modelo NoSQL mientras introduce capacidades de consulta significativamente más sofisticadas. En este paradigma, la información se organiza en documentos autocontenidos, típicamente codificados en formato JSON o BSON, que pueden contener estructuras jerárquicas arbitrariamente complejas sin requerir un esquema predefinido. Sarasa (2016) proporciona una introducción comprehensiva a las bases de datos NoSQL utilizando MongoDB como caso de estudio, destacando cómo estas implementaciones ofrecen lenguajes de consulta expresivos que permiten filtrado, proyección, agregación y transformación de datos con una sintaxis declarativa comparable a SQL.

La flexibilidad esquemática inherente a las bases documentales resulta particularmente valiosa en el contexto de la investigación en ciencia de datos, donde los modelos conceptuales evolucionan iterativamente durante las fases de exploración y experimentación. Según Sarasa (2016), esta característica facilita el desarrollo ágil de aplicaciones analíticas, permitiendo a los científicos de datos incorporar nuevos atributos y dimensiones sin incurrir en costosas migraciones de esquema o períodos de inactividad del sistema. Chodorow y Dirolf (2010) argumentan que MongoDB específicamente fue diseñado para abordar las limitaciones de los sistemas relacionales en entornos donde la estructura de datos es variable o está en constante evolución. En proyectos de Internet de las Cosas, donde los sensores heterogéneos generan estructuras de datos variables, las bases documentales eliminan la complejidad de normalizar información diversa en esquemas rígidos.

Desde una perspectiva de arquitectura de datos moderna, Carrascal y Varona (2024) señalan que las bases documentales son especialmente adecuadas para implementar sistemas de

gestión de contenido, plataformas de comercio electrónico con catálogos de productos altamente variables y aplicaciones móviles que requieren sincronización offline. En el ámbito del Machine Learning, estos sistemas facilitan el almacenamiento de experimentos y metadatos de modelos, donde cada ejecución puede generar conjuntos de hiperparámetros y métricas de evaluación con estructuras heterogéneas. La capacidad de realizar agregaciones complejas directamente en la base de datos permite implementar pipelines de feature engineering sin necesidad de extraer datos a sistemas externos de procesamiento, como documentan Han, Haihong, Le y Du (2011) en su análisis comparativo de sistemas NoSQL para aplicaciones de Big Data.

Sin embargo, las limitaciones del modelo documental se manifiestan en escenarios que requieren relaciones complejas entre entidades o transacciones distribuidas que abarcan múltiples documentos. Aunque versiones recientes de MongoDB han incorporado soporte transaccional ACID multi#documento, como documenta Sarasa (2016), estas operaciones implican penalizaciones significativas de rendimiento y no alcanzan la robustez de los sistemas relacionales tradicionales. Adicionalmente, la ausencia de restricciones de integridad referencial a nivel de sistema transfiere la responsabilidad de mantener la consistencia semántica a la capa de aplicación, incrementando la complejidad del código y el riesgo de anomalías de datos. Gutiérrez Hernández e Ibarra Limas (2024) advierten que esta característica requiere disciplina arquitectónica rigurosa para evitar inconsistencias en proyectos de Big Data de gran escala.

Bases de Datos de Columnas Anchas: Optimización para Analítica Distribuida

El modelo de columnas anchas, también denominado column#family o wide#column stores, representa una arquitectura especializada diseñada explícitamente para cargas de trabajo analíticas sobre conjuntos de datos masivos distribuidos. A diferencia de las bases de datos relacionales tradicionales que organizan físicamente los datos por filas, este paradigma agrupa las

columnas, permitiendo una compresión más eficiente y acceso selectivo a subconjuntos específicos de atributos sin necesidad de leer registros completos. Lakshman y Malik (2010) describen en detalle la arquitectura de Apache Cassandra, mientras que Chang et al. (2008) documentan los fundamentos de Google Bigtable dos de las implementaciones más influyentes de este modelo que proporcionan capacidades de escritura extremadamente rápidas y escalamiento lineal mediante arquitecturas peer#to#peer sin puntos únicos de falla.

Gutiérrez Hernández e Ibarra Limas (2024) destacan que las bases de columnas anchas resultan óptimas para aplicaciones que requieren ingestión continua de grandes volúmenes de datos con patrones de lectura predecibles y orientados a columnas específicas. En el contexto de la analítica de Big Data, sistemas como Apache Cassandra permiten implementar arquitecturas lambda y kappa que combinan procesamiento batch y streaming, almacenando tanto datos históricos como flujos en tiempo real en una infraestructura unificada. Carrascal y Varona (2024) enfatizan que la capacidad de distribuir automáticamente los datos entre múltiples nodos mediante particionamiento consistente garantiza que la adición de nuevos servidores resulte en mejoras proporcionales de capacidad y throughput, una característica fundamental para proyectos escalables de ciencia de datos.

En el ámbito de la ciencia de datos aplicada, las bases columnares son especialmente valiosas para almacenar series temporales de alta frecuencia generadas por sensores IoT, métricas de telemetría de aplicaciones distribuidas y registros de eventos de aplicaciones web. Picher Vera, Martínez María#Dolores y Bernal García (2018) documentan cómo la integración de estas bases de datos con herramientas de visualización como Power BI permite a los analistas universitarios y profesionales explorar grandes volúmenes de datos de manera interactiva. La investigación en sistemas de recomendación aprovecha este modelo para almacenar matrices dispersas de

interacciones usuario#artículo, donde la estructura columnar permite recuperar eficientemente todos los elementos con los que un usuario ha interactuado o todos los usuarios que han consumido un contenido específico. Adicionalmente, en aplicaciones de Machine Learning que requieren feature stores distribuidos, las bases columnares facilitan el almacenamiento y recuperación rápida de características computadas offline para inferencia en tiempo real, como demuestra el trabajo de Kreps, Narkhede y Rao (2011) sobre Apache Kafka y sistemas de streaming.

No obstante, la arquitectura columnar presenta limitaciones significativas para consultas exploratorias complejas que involucran múltiples familias de columnas o requieren joins entre diferentes tablas. La ausencia de capacidades relacionales nativas y la necesidad de desnormalizar datos para optimizar patrones de acceso específicos pueden resultar en redundancia sustancial y complejidad operativa. Carrascal y Varona (2024) señalan que, aunque Cassandra ofrece CQL (Cassandra Query Language) con sintaxis similar a SQL, las restricciones arquitectónicas subyacentes limitan severamente el tipo de consultas soportadas, requiriendo que los patrones de acceso sean definidos durante el diseño del esquema mediante la cuidadosa selección de claves de particionamiento y clustering. Esta observación es consistente con los hallazgos de Abadi, Boncz y Harizopoulos (2009) en su análisis de los compromisos arquitectónicos entre sistemas orientados a filas y columnas.

Bases de Datos de Grafos: Modelado Nativo de Relaciones Complejas

Las bases de datos de grafos representan un paradigma fundamentalmente diferente, diseñado específicamente para almacenar y consultar eficientemente estructuras de datos donde las relaciones entre entidades son tan importantes como las entidades mismas. A diferencia de los modelos previamente discutidos, que tratan las relaciones como características secundarias

implementadas mediante referencias o desnormalización, las bases de grafos materializan explícitamente las conexiones como objetos de primera clase con propiedades y semántica propias. Robinson, Webber y Eifrem (2015) proporcionan un análisis exhaustivo de este modelo, mientras que Angles y Gutierrez (2008) formalizan sus fundamentos teóricos desde la perspectiva de la teoría de grafos. Neo4j, Amazon Neptune y JanusGraph ejemplifican este enfoque, proporcionando lenguajes de consulta declarativos como Cypher y Gremlin que permiten expresar naturalmente travesías complejas y patrones de conexión.

La arquitectura de grafos resulta particularmente valiosa en dominios donde las relaciones multi#hop son fundamentales para el análisis, incluyendo redes sociales, sistemas de detección de fraude, grafos de conocimiento y motores de recomendación basados en filtrado colaborativo. Carrascal y Varona (2024) documentan que la capacidad de realizar traversals de múltiples niveles con rendimiento constante, independientemente de la profundidad de la búsqueda, constituye una ventaja decisiva sobre las bases de datos relacionales, donde cada nivel de relación requiere un join adicional con costo computacional creciente. Vicknair et al. (2010) realizaron un estudio comparativo demostrando que las consultas de travesía en Neo4j pueden ser órdenes de magnitud más rápidas que las equivalentes en MySQL cuando la profundidad de la relación excede tres niveles. En el contexto de la ciencia de datos, las bases de grafos facilitan la implementación de algoritmos de centralidad, detección de comunidades y análisis de caminos críticos directamente en la capa de almacenamiento.

Las aplicaciones contemporáneas de inteligencia artificial aprovechan las bases de grafos para implementar Graph Neural Networks (GNNs), donde la estructura topológica de los datos es esencial para el aprendizaje de representaciones. Zhou et al. (2020) proporcionan una revisión comprehensiva de las redes neuronales de grafos y sus aplicaciones en diversos dominios

científicos. Los sistemas de gestión de conocimiento semántico, incluyendo ontologías biomédicas y grafos industriales de conocimiento, requieren la expresividad semántica que únicamente las bases de grafos pueden proporcionar eficientemente, como documenta el trabajo de Ehrlinger y Wöß (2016) sobre grafos de conocimiento. Adicionalmente, en aplicaciones de análisis de dependencias de software, rastreo de transacciones distribuidas y gestión de identidades y accesos, el modelo de grafo permite consultas naturales que serían prohibitivamente complejas en arquitecturas relacionales o documentales.

Sin embargo, las bases de grafos presentan desafíos significativos en términos de escalamiento horizontal y distribución de datos. La naturaleza interconectada de los grafos complica fundamentalmente el particionamiento, dado que cualquier división arbitraria del conjunto de datos inevitablemente resulta en aristas que atraviesan fronteras de particiones, generando tráfico de red adicional durante las consultas. Gutiérrez Hernández e Ibarra Limas (2024) advierten que, aunque sistemas como JanusGraph implementan estrategias de particionamiento heurístico y cachés distribuidos, el rendimiento óptimo de las bases de grafos generalmente se alcanza en configuraciones de memoria que permiten mantener el grafo completo en RAM. Adicionalmente, para cargas de trabajo que no involucran traversals complejos o donde las relaciones son escasas y predecibles, la sobrecarga arquitectónica de las bases de grafos puede resultar en rendimiento inferior comparado con alternativas más simples, como demuestran los experimentos de rendimiento de Domínguez#Sal et al. (2010).

El análisis comparativo de los cuatro paradigmas NoSQL fundamentales revela que no existe una arquitectura universalmente superior, sino más bien una colección de herramientas especializadas, cada una optimizada para patrones de acceso, estructuras de datos y requisitos no funcionales específicos. El concepto de persistencia políglota, ampliamente discutido por

Sadalage y Fowler (2012) y adoptado en arquitecturas de microservicios contemporáneas, reconoce esta realidad al proponer que las aplicaciones modernas deben emplear múltiples tecnologías de almacenamiento, seleccionando la más apropiada para cada subsistema o fase del pipeline de datos. Carrascal y Varona (2024) enfatizan que esta aproximación arquitectónica representa un cambio paradigmático fundamental en el diseño de sistemas de información empresariales y científicos.

Las bases clave#valor ofrecen rendimiento extremo y simplicidad operativa para casos de uso con patrones de acceso predecibles y estructuras de datos planas, como documentan Miranda et al. (2023) en su evaluación comparativa. Las bases documentales proporcionan el equilibrio óptimo entre flexibilidad esquemática y capacidad de consulta para aplicaciones que manejan datos semiestructurados heterogéneos, tal como demuestra Sarasa (2016) en su análisis de MongoDB. Las bases columnares destacan en escenarios analíticos distribuidos que requieren ingestión masiva de datos y consultas orientadas a agregaciones sobre subconjuntos específicos de atributos, siendo particularmente relevantes para proyectos de Big Data universitarios e industriales según documenten Gutiérrez Hernández e Ibarra Limas (2024) y Picher Vera et al. (2018). Finalmente, las bases de grafos resultan indispensables cuando las relaciones complejas constituyen el núcleo del modelo conceptual y los algoritmos de análisis requieren traversals eficientes de múltiples niveles.

Para investigadores y profesionales en ciencia de datos e ingeniería de Big Data, la comprensión profunda de estos paradigmas trasciende el conocimiento puramente técnico, constituyendo una competencia estratégica que habilita el diseño de arquitecturas de datos resilientes, escalables y eficientes. La integración de estos sistemas con herramientas modernas de visualización y análisis, como Power BI documentado por Contreras García y Cabrera Lozano

(2024) y Jiménez (2021), amplifica significativamente su valor en contextos académicos y profesionales. La elección informada entre estos modelos, fundamentada en el análisis riguroso de requisitos funcionales, patrones de acceso anticipados y restricciones de escalabilidad, determina en última instancia la viabilidad y el rendimiento de los sistemas analíticos modernos. La evolución continua del ecosistema NoSQL, con la incorporación progresiva de capacidades transaccionales, indexación avanzada y procesamiento analítico integrado, promete ampliar aún más el espectro de aplicaciones donde estos sistemas constituyen la opción arquitectónica preferente, consolidando la persistencia políglota como paradigma dominante en la ingeniería de datos contemporánea.

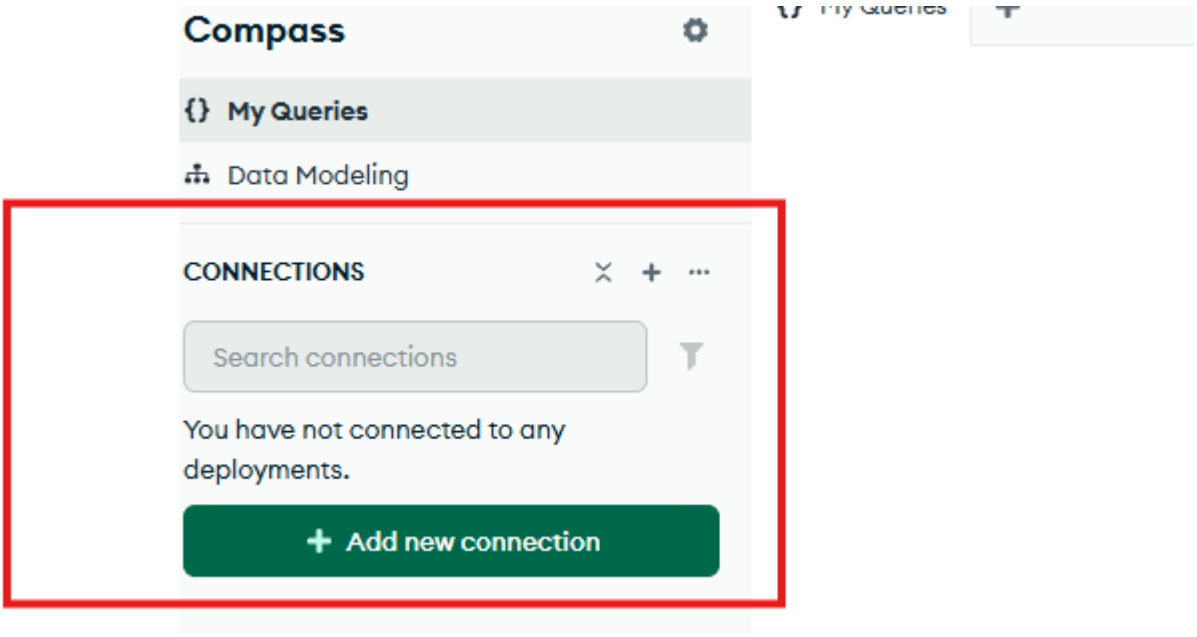
Modelo NoSQL	Descripción Arquitectónica	Ventajas Técnicas	Limitaciones / Desventajas	Casos de Uso Recomendados	Ejemplos Prácticos
Clave –Valor	Basado en un diccionario distribuido donde cada valor se almacena bajo una llave única. No existe esquema ni estructura fija.	# Máxima velocidad de lectura y escritura. # Escalabilidad horizontal simple. # Latencias extremadamente bajas. # Ideal para almacenamiento de datos simples y acceso directo.	# No soporta consultas complejas. # No maneja relaciones ni estructuras anidadas. # No ofrece indexación secundaria en la mayoría de las implementaciones. # No adecuado para análisis exploratorio.	# Caches distribuidas. # Contadores. # Sesiones de usuario. # Configuración en tiempo real. # Sistemas de autenticación.	**1**. Redis para almacenar sesiones de usuarios y tokens JWT en aplicaciones web de alto tráfico. 2. Contadores de vistas, likes o métricas en redes sociales en tiempo real (e.g., DynamoDB). 3. Almacenamiento rápido de configuraciones para microservicios distribuidos.
Documentos	Datos almacenados en documentos JSON/BSON, con estructura jerárquica flexible. Cada documento es autocontenido.	# Flexibilidad esquemática. # Permite datos heterogéneos y anidados. # Soporte para consultas complejas, agregaciones y pipelines. # Intuitivo para desarrolladores.	# Relaciones complejas difíciles de manejar. # Transacciones costosas en rendimiento. # No hay integridad referencial nativa. # Puede existir duplicación de datos.	# Comercio electrónico. # IoT. # Aplicaciones móviles. # Catálogos dinámicos. # Sistemas de contenido.	1. MongoDB gestionando catálogos de productos con atributos variables según la categoría (e.g., electrodomésticos, ropa, servicios digitales). 2. Almacenamiento de datos IoT con estructuras cambiantes según el sensor. 3. Registro de perfiles de usuario con datos anidados: dirección, listas, preferencias, historial.

		# Escalabilidad horizontal mediante sharding.			
Columnas Anchas (Wide-Column)	Datos almacenados por familias de columnas, optimizadas para lectura selectiva y escritura distribuida masiva.	# Ideal para analítica a gran escala. # Altísima eficiencia en ingestión continua de datos. # Acceso rápido a subconjuntos de columnas. # Escala lineal con nodos adicionales. # Excelente compresión.	# Diseños dependientes de patrones de acceso (schema-on-read). # Consultas complejas limitadas. # Carece de joins tradicionales. # Requiere desnormalización.	# Series temporales. # Telemetría. # Logs distribuidos. # Sensores IoT. # Analítica a gran escala.	1. Apache Cassandra almacenando millones de registros de telemetría de servidores (CPU, RAM, tráfico). 2. Bigtable gestionando series temporales de sensores IoT de alta frecuencia. 3. Logs masivos de aplicaciones para análisis posterior en plataformas Big Data.
Grafos	Modelo basado en nodos y relaciones (aristas). Ideal para estructuras altamente interconectadas. Ofrece modelos relacionales nativos.	# Consultas multi#nivel optimizadas. # Alta expresividad semántica. # Ideal para relaciones complejas. # Permite análisis mediante algoritmos avanzados (PageRank, comunidades).	# Difícil escalamiento horizontal real. # Alto uso de memoria. # Complejo particionamiento del grafo. # No eficiente para datos tabulares simples.	# Redes sociales. # Sistemas de recomendación. # Detección de fraude. # Análisis de redes de conocimiento. # Topologías complejas.	1. Neo4j detectando fraude bancario mediante patrones de transacciones sospechosas entre cuentas. 2. Análisis de redes sociales identificando comunidades y relaciones profundas. 3. Motores de recomendación basados en similitud de nodos (e.g., usuarios que ven productos similares).

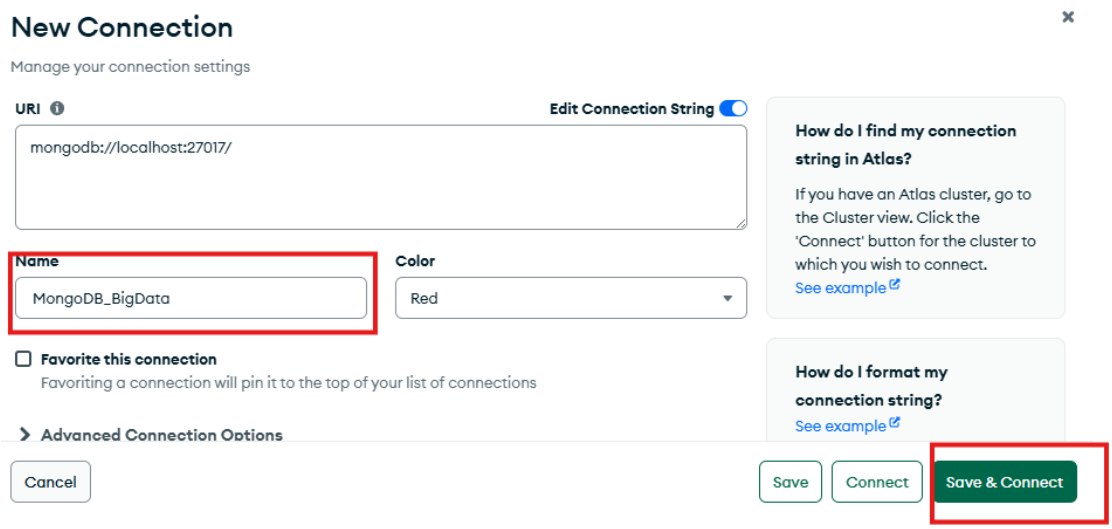
Fase 2. Mongo DB:

1. Diseño de la base de datos.

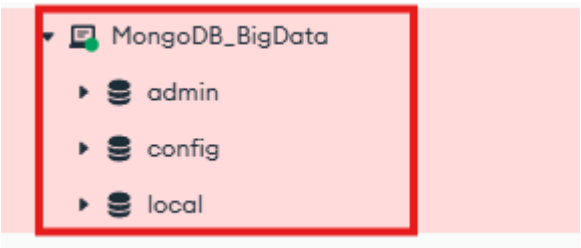
Se creo la conexión dentro de mongo compas,



luego se procedió a darle un nombre a la conexión. (**MongoDB_BigData)**



Damos un nombre a la conexión y luego guardamos y conectamos.



Se nos va a crear la conexión y luego podremos dentro de la consola de Mongosh crear nuestra BD.

```
> use catalogo_comercio_electronico
< switched to db catalogo_comercio_electronico
```

Creación de la BD desde la consola de Mongosh nombrada **(catalogo_comercio_electronico).**

```
> db.createCollection("categorias")

  db.createCollection("productos")

  db.createCollection("usuarios")

  db.createCollection("pedidos")

  db.createCollection("resenas")
< { ok: 1 }
```

Creación de las colecciones dentro de la BD.

Cada documento dentro de la colección **categorias** sigue la siguiente estructura:

Campo (Clave)	**Tipo de Dato (BSON)**	**Requerido**	**Propósito**	**Ejemplo del Documento**
_id	String	Sí	Identificador único de la categoría (clave primaria).	"cat_fywrh23ss"
nombre	String	Sí	Nombre de la categoría legible para el usuario.	"Electrónica"
descripción	String	No	Texto descriptivo de la categoría.	"Dispositivos electrónicos, audio y wearables"
slug	String	Sí	Versión simplificada y segura para URL. Debe ser único.	"electronica"
activa	Boolean	Sí	Indica si la categoría está disponible y visible (true) o inactiva (false).	true
Fecha_creación	String (Se recomienda Date)	Sí	Fecha en la que el documento fue creado.	"2019#01#14"

Cada documento dentro de la colección **pedidos** sigue la siguiente estructura:

Campo (Clave)	Tipo de Dato (BSON)	Propósito	Ejemplo del Documento
_id	String	Identificador único de la orden (clave primaria).	"ord_guqry9i6lk"
usuario_id	String	Referencia al _id del usuario que realizó la orden.	"usr_i26xh26i00"
items	Array (5)	Documento anidado: Lista de productos incluidos en el pedido (5 elementos en el ejemplo).	[Ver Sub#estructura 2.1]
resumen	Object	Documento anidado: Totales de la orden (subtotal, impuestos, total final).	[Ver Sub#estructura 2.2]
metodo_pago	String	Método utilizado para pagar la orden.	"pse"
estado	String	Estado actual de la orden en el proceso de logística.	"entregado"
direccion_envio	Object	Documento anidado: Dirección completa donde se debe entregar el pedido.	[Ver Sub#estructura 2.3]
tracking	Object	Documento anidado: Información de seguimiento y logística.	[Ver Sub#estructura 2.4]
fecha_pedido	String	Fecha en que se creó la orden.	"2023#06#25"

fecha_actualizacion	String	Fecha de la última modificación o cambio de estado del pedido.	"2024#05#13"

Y así sucesivamente cada colección tiene su estructura **documental** dentro de cada **campo**.

2. Implementación en MongoDB:

Inserción de datos.

Inserción de documentos dentro de una la colección llamada **categoría** que está en la BD creado previamente en Mongo Compas.

Consola desde Mongosh

```
>_MONGOSH
> use catalogo_comercio_electronico
< switched to db catalogo_comercio_electronico
> db.categorias.insertOne({

      "_id": "cat_fywrh23ss",

      "nombre": "Electrónica",

      "descripcion": "Dispositivos electrónicos, audio y wearables",

      "slug": "electrónica",

      "activa": true,

      "fecha_creacion": "2019-01-14"

  })
< {
    acknowledged: true,
    insertedId: 'cat_fywrh23ss'
  }
catalogo_comercio_electronico>
```

Con use **catalogo_comercio_electronico** lo que hacemos en dentrar a la BD,

switched to db catalogo_comercio_electronico, nos confirma que acedimos a la BD correctamente

db.categorias.insertOne({
"_id": "cat_fywrh23ss",
"nombre": "Electrónica",
"descripcion": "Dispositivos electrónicos, audio y wearables",
"slug": "electrónica",
"activa": true,

```
  "fecha_creacion": "2019#01#14"
})
```

Con este codigo insertamos dentro de la colección **categorias** insertamos datos de un documento. Como la actividad solicita que se agreguen a cada colección minimo 100 documentos se opta por buscar en linea archivos JSON con datos similares para la prueba https://www.kaggle.com/datasets/aaditshukla/flipkart#fasion#products#dataset

Implementar consultas utilizando el lenguaje de consulta.

Consulta se inserción, como previamente lo habíamos realizado

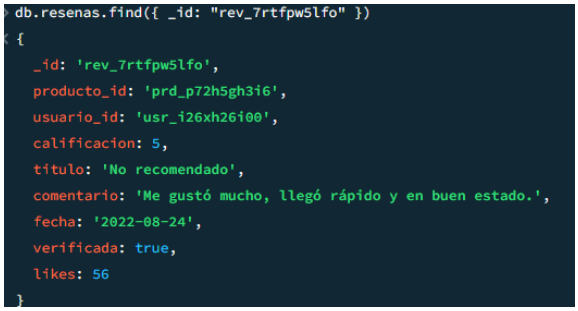
```
> db.productos.insertOne({
  nombre: "Teclado Mecánico",
  precio: 95.50,
  stock: 50,
  activo: true
  })
< {
    acknowledged: true,
    insertedId: ObjectId('69227617e57873f918c8f8b4')
  }
```

Se evidencia la insercion de un documento dentro de la colección productos.

2.1. Consultas básicas (inserción, selección, actualización y eliminación de documentos)

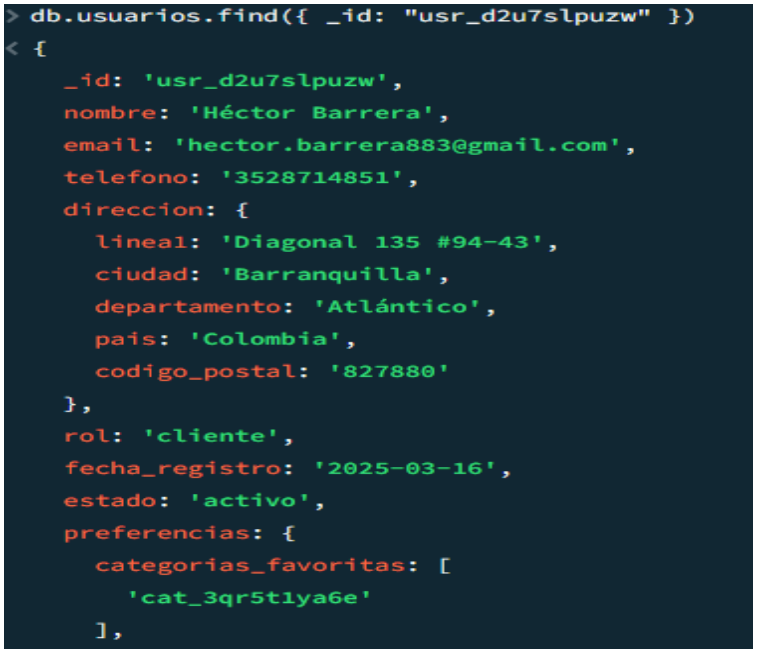
CONSULTA SE “**SELECCIÓN**”

db.categorias.find({ _id: "cat_demo_001" })



Observamos que encuenta el documento buscado dento de la colección.

Daremos 2 ejemplos mas:



db.usuarios.find({ _id: "usr_d2u7slpuzw" })

```
> db.pedidos.find({ _id: "ord_guqry9i61k" })
< {
    _id: 'ord_guqry9i61k',
    usuario_id: 'usr_i26xh26i00',
    items: [
      {
        producto_id: 'prd_ew98bshuu3',
        nombre_snapshot: 'Balón Golty profesional',
        precio_unitario: {
          valor: 405.66,
          moneda: 'COP'
        },
        cantidad: 2
      },
      {
        producto_id: 'prd_o5fx45ihau',
        nombre_snapshot: 'Chocolatina Jet caja surtida',
        precio_unitario: {
          valor: 33.95,
          moneda: 'COP'
```

db.pedidos.find({ _id: "ord_guqry9i61k" })

CONSULTA DE **"ACTUALIZACIÓN"**

```
> db.usuarios.updateOne(
    { _id: "usr_rcgb7361qu" },
    {
      $set: {
        nombre: "Gabriela Gómez R.",
        telefono: "3005552211"
      }
    }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

Este codigo indica que se está buscando el documento cuyo campo **_id** es **"usr_rcgb7361qu"** y se están modificando dos atributos:

- nombre, que pasa a ser **"Gabriela Gómez R.",**
- telefono, que pasa a ser **"3005552211".**

Debajo del comando aparece la respuesta del sistema MongoDB, mostrada como un documento JSON con los siguientes campos:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Esto indica:

- acknowledged: true → MongoDB recibió y ejecutó la operación correctamente.
- matchedCount: 1 → Se encontró exactamente un documento que coincidía con el filtro _id.
- modifiedCount: 1 → Se modificó correctamente el documento encontrado.
- insertedId: null → No se insertó ningún documento (no corresponde a este tipo de operación).
- upsertedCount: 0 → No se creó un **nuevo documento (porque no se usó upsert).**

```
> use catalogo_comercio_electronico
< switched to db catalogo_comercio_electronico
> db.usuarios.updateOne(
    { _id: "usr_i26xh26i00" },
    {
      $set: {
        telefono: "3207778899",
        estado: "inactivo"
      }
    }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

De igual manera en esta captura se observa un codigo con la miasma estructura que el anterior solo que se actualiza el telefono y el estado del usuario de inactivo a activo.

Consulta de **“ELIMINACIÓN”**

```
> db.productos.deleteOne({ _id: "prd_b29co0vs85" })
< {
    acknowledged: true,
    deletedCount: 1
  }
> db.productos.deleteOne({ _id: "prd_u8f32xpuai" })
< {
    acknowledged: true,
    deletedCount: 1
  }
> db.productos.find({ _id: { $in: ["prd_b29co0vs85", "prd_u8f32xpuai"] } })
<
catalogo_comercio_electronico>
```

Se procese a eliminar el producto con id **prd_b29co0vs85** y **prd_u8f32xpuai** de la lsita del archivo dedocumentos de la colección de **productos,** luego se procede a buscarlos pero ya no aparecen como archivos de la colección.

2.2. Consultas con filtros y operadores.

Consulta de Productos con stock menor de 20 con el operador de comparación **$lt**

```
> db.productos.find(
    { stock: { $lt: 20 } },
    { nombre: 1, stock: 1 }
  )
< {
    _id: 'prd_pjmlozn7aa',
    nombre: 'Salsa de ají artesanal',
    stock: 14
  }
  {
    _id: 'prd_09tfor415b',
    nombre: "Portátil 14'' i5 16GB 669",
    stock: 8
  }
  {
    _id: 'prd_pzuzk8lwkx',
    nombre: 'Mouse inalámbrico 150',
    stock: 10
  }
  {
    _id: 'prd_gi7ztccc9h',
    nombre: 'Zapatillas running 653',
    stock: 0
```

La imagen muestra una consulta realizada en la colección **productos** utilizando el método find(). El filtro { stock: { $lt: 20 } } selecciona únicamente los productos con stock menor a 20 unidades, mientras que la proyección { nombre: 1, stock: 1 } indica que solo se deben mostrar esos dos campos. El resultado incluye varios productos con inventarios bajos, como “Salsa de ají artesanal” (14 unidades), “Portátil 14" i5 16GB 669” (8 unidades), “Mouse inalámbrico 150” (10 unidades) y “Zapatillas running 653” (0 unidades). Esto permite identificar rápidamente artículos que requieren reposición.

Consulta de Usuarios registrados después de 2024 con el operador de comparación **$gte**

```
> db.usuarios.find(
    { fecha_registro: { $gte: "2024-01-01" } },
    { nombre: 1, fecha_registro: 1 }
  )
< {
    _id: 'usr_i26xh26i00',
    nombre: 'Carlos Ospina',
    fecha_registro: '2025-03-01'
  }
  {
    _id: 'usr_d2u7slpuzw',
    nombre: 'Héctor Barrera',
    fecha_registro: '2025-03-16'
  }
  {
    _id: 'usr_gczkhw9187',
    nombre: 'Natalia Navarro',
    fecha_registro: '2025-03-01'
  }
```

Pe evidencia una consulta realizada sobre la colección usuarios utilizando el operador de comparación $gte para filtrar aquellos registros cuya fecha de registro es igual o posterior al 1 de enero de 2024. La proyección { nombre: 1, fecha_registro: 1 } permite visualizar únicamente estos dos campos. Como resultado, se listan usuarios recientes, entre ellos Carlos Ospina, Héctor Barrera y Natalia Navarro, todos registrados durante el año 2025. Esta consulta permite identificar usuarios ingresados al sistema en fechas recientes.

Consulta de Productos de varias categorías con el operador de comparación **$in.**

```
> db.productos.find(
    { categoria_id: { $in: ["cat_a7pf17yp91", "cat_wblre8nf3e"] } }
  )
< {
    _id: 'prd_ukoa1nsr18',
    nombre: 'Mochila Wayuu artesanal',
    categoria_id: 'cat_a7pf17yp91',
    marca: 'Artesanías Guajira',
    precio: {
      valor: 687.28,
      moneda: 'COP'
    },
    stock: 204,
    descripcion: 'Perfecto para regalo o colección. Producto original con excelente calidad.',
    especificaciones: {
      origen: 'Hecho en Colombia',
      color: 'beige',
      talla: 'M',
      peso_kg: 0.57
    },
```

Presenta una consulta realizada sobre la colección productos, donde se utiliza el operador $in para filtrar todos los documentos cuyo categoria_id coincide con cualquiera de los valores especificados en el arreglo. En este caso, la consulta recupera los productos pertenecientes a las categorías "cat_a7pf17yp91" y "cat_wblre8nf3e". El resultado muestra artículos que cumplen esta condición, como la *"Mochila Wayuu artesanal"*, incluyendo su información detallada: marca, precio, stock, descripción y especificaciones. Esta operación permite obtener productos asociados a múltiples categorías de manera eficiente.

Consulta de Productos activos con stock mayor a 30 con el operadores logicos **$and**

```
> db.productos.find(
    {
      $and: [
        { activo: true },
        { stock: { $gt: 3 } }
      ]
    }
  )
< {
    _id: ObjectId('69227617e57873f918c8f8b4'),
    nombre: 'Teclado Mecánico',
    precio: 95.5,
    stock: 50,
    activo: true
  }
  {
    _id: 'prd_t2epbbm4yg',
    nombre: 'Café de origen Huila 500g',
    categoria_id: 'cat_8v5rni5fc2',
    marca: 'Juan Valdez',
    precio: {
      valor: 40.52,
      moneda: 'COP'
    },
```

Muestra una consulta realizada sobre la colección **productos**, donde se utiliza el operador lógico $and para combinar dos condiciones: que el producto esté activo (activo: true) y que el stock sea mayor o igual a 3 (stock: { $gte: 3 }). Esta operación permite recuperar únicamente los productos disponibles y con inventario suficiente. El resultado incluye artículos como el *"Teclado Mecánico"* y el *"Café de origen Huila 500g"*, mostrando sus campos principales (nombre, precio, stock y estado). Esta consulta es útil para identificar productos activos que aún cuentan con existencias para la venta.

Consulta de Usuarios cuyo rol sea cliente *o* premium con el operadores logicos **$or**

```
> db.usuarios.find(
    { $or: [ { rol: "cliente" }, { rol: "premium" } ] }
  )
< {
    _id: 'usr_i26xh26i00',
    nombre: 'Carlos Ospina',
    email: 'carlos.ospina607@unadvirtual.edu.co',
    telefono: '3207778899',
    direccion: {
      linea1: 'Carrera 88 #51-5',
      ciudad: 'Armenia',
      departamento: 'Quindío',
      pais: 'Colombia',
      codigo_postal: '681440'
    },
    rol: 'cliente',
    fecha_registro: '2025-03-01',
    estado: 'inactivo',
    preferencias: {
      categorias_favoritas: [
        'cat_1vs9izrttp',
        'cat_qhafftgxwe',
```

Se observa una consulta realizada sobre la colección **usuarios** utilizando el operador lógico $or. La instrucción busca todos los documentos cuyo campo rol sea "cliente" o "premium", permitiendo obtener usuarios que pertenecen a cualquiera de estos dos perfiles. El resultado muestra información detallada de uno de los usuarios coincidentes, incluyendo su nombre, correo, teléfono, dirección completa, fecha de registro, estado y preferencias. Esta consulta es útil para segmentar usuarios según su tipo de rol dentro del sistema.

Consulta de Pedidos que NO estén entregados con el operadores logicos **$ne.**

```
> db.pedidos.find(
    { estado: { $ne: "entregado" } }
  )
< {
    _id: 'ord_7limyfh7xs',
    usuario_id: 'usr_w7079egqer',
    items: [
      {
        producto_id: 'prd_b29co0vs85',
        nombre_snapshot: 'Ruana de lana boyacense',
        precio_unitario: {
          valor: 457.46,
          moneda: 'COP'
        },
        cantidad: 1
      }
    ],
    resumen: {
      subtotal: {
        valor: 457.46,
        moneda: 'COP'
      },
      envio: {
        valor: 9490.96,
```

Se muestra una consulta realizada sobre la colección pedidos, donde se utiliza el operador $ne para filtrar todos los pedidos cuyo estado no sea “entregado”. El resultado incluye información completa del pedido encontrado, como el identificador del usuario que lo realizó, la lista de productos adquiridos, el precio unitario, la cantidad y el resumen del costo total, incluyendo subtotal y envío. Esta consulta permite identificar pedidos pendientes o en proceso dentro del sistema.

2.3. Consultas de agregación para calcular estadísticas (contar, sumar, promediar, etc.).

CONTAR DOCUMENTOS POR CATEGORÍA.

```
> db.productos.aggregate([
    {
      $group: {
        _id: "$categoria_id",
        total_productos: { $sum: 1 }
      }
    }
  ])
< {
    _id: 'cat_wblre8nf3e',
    total_productos: 7
  }
  {
    _id: 'cat_8v5rni5fc2',
    total_productos: 10
  }
  {
    _id: null,
    total_productos: 1
  }
  {
    _id: 'cat_a7pf17yp91',
    total_productos: 16
  }
```

Se muestra una consulta de agregación ejecutada sobre la colección **productos**, donde se utiliza la etapa $group para agrupar los documentos según el campo categoria_id. En la operación, se calcula el número total de productos por categoría usando el acumulador $sum: 1. El resultado lista cada categoría junto con la cantidad de productos asociados, mostrando valores como 7, 10 y 16 productos, así como un grupo con _id: null correspondiente a registros sin categoría asignada. Esta consulta permite obtener una visión general de la distribución de productos por categoría.

CALCULAR EL PROMEDIO DE STOCK POR MARCA.

```
> db.productos.aggregate([
    {
      $group: {
        _id: "$marca",
        promedio_stock: { $avg: "$stock" }
      }
    }
  ])
< {
    _id: 'Golty',
    promedio_stock: 130
  }
  {
    _id: 'Juan Valdez',
    promedio_stock: 177
  }
  {
    _id: 'Penguin Random House',
    promedio_stock: 171
  }
```

Se evidencia una consulta de agregación ejecutada en la colección productos, donde se utiliza la etapa $group para agrupar los documentos por el campo marca. Dentro del agrupamiento se calcula el promedio del stock de cada marca mediante el operador $avg. El resultado presenta varias marcas junto con su respectivo promedio de unidades en inventario, como Golty (130), Juan Valdez (177) y Penguin Random House (171). Esta consulta permite analizar la disponibilidad promedio de productos según su marca.

TOTAL, GASTADO POR USUARIO + CANTIDAD DE PEDIDOS.

```
> db.pedidos.aggregate([
    {
      $group: {
        _id: "$usuario_id",
        gasto_total: { $sum: "$resumen.total.valor" },
        pedidos_realizados: { $sum: 1 }
      }
    },
    { $sort: { gasto_total: -1 } }
  ])
< {
    _id: 'usr_d1xs6jwy91',
    gasto_total: 131888.01,
    pedidos_realizados: 6
  }
  {
    _id: 'usr_z0ngb7awy7',
    gasto_total: 82542.22,
    pedidos_realizados: 4
  }
  {
    _id: 'usr_i26xh26i00',
    gasto_total: 61843.119999999995,
    pedidos_realizados: 4
```

La imagen muestra una consulta de agregación aplicada a la colección pedidos, donde se agrupan los documentos por el campo usuario_id usando $group. En esta etapa se calcula el gasto total por usuario mediante $sum: "$resumen.total.valor" y el número de pedidos realizados con $sum: 1. Posteriormente, los resultados se ordenan de forma descendente por gasto total utilizando $sort. El resultado muestra usuarios con sus valores acumulados, evidenciando, por ejemplo, clientes que han gastado entre $61.843 y $131.888 millones de pesos, así como la cantidad de pedidos asociados a cada uno. Esta consulta permite identificar a los usuarios con mayor actividad y consumo dentro del sistema.

CALIFICACIÓN PROMEDIO POR PRODUCTO (RESEÑAS).

```
> db.resenas.aggregate([
    {
      $group: {
        _id: "$producto_id",
        promedio_calificacion: { $avg: "$calificacion" },
        total_resenas: { $sum: 1 }
      }
    },
    { $sort: { promedio_calificacion: -1 } }
  ])
<
```

La imagen muestra una consulta de agregación aplicada a la colección resenas, donde se agrupan los documentos por el campo producto_id utilizando $group. En esta etapa se calcula la calificación promedio de cada producto mediante el operador $avg y el total de reseñas usando $sum. Finalmente, los resultados se ordenan de forma descendente según el promedio de calificación utilizando $sort. Esta consulta permite identificar cuáles productos tienen mejores valoraciones y cuántas reseñas han recibido en total.

TOP 5 PRODUCTOS CON MÁS RESEÑAS.

```
> db.resenas.aggregate([
    {
      $group: {
        _id: "$producto_id",
        total_resenas: { $sum: 1 }
      }
    },
    { $sort: { total_resenas: -1 } },
    { $limit: 5 }
  ])
<
```

Se muestra una consulta de agregación ejecutada sobre la colección resenas, donde se agrupan los documentos por producto_id mediante $group, calculando el total de reseñas de cada producto con el acumulador $sum. Luego, la etapa $sort ordena los resultados de forma descendente según la cantidad de reseñas, y finalmente $limit reduce la salida a los cinco productos con mayor número de reseñas. Esta consulta permite identificar rápidamente los productos más comentados dentro del sistema.

Enlace al repositorio de código.

https://github.com/jhonpinzon18/Tarea_4_BigData

Conclusiones.

La comparación entre los diferentes modelos de bases de datos NoSQL permitió demostrar que no existe una solución única que se ajuste a todos los escenarios de Big Data. Cada paradigma clave-valor, documentos, columnas anchas y grafos responde a necesidades específicas relacionadas con el rendimiento, la estructura de los datos, la escalabilidad y los patrones de consulta. En consecuencia, la selección adecuada del modelo de almacenamiento debe partir de un análisis detallado del caso de uso, ya que esta decisión impacta directamente en la eficiencia, los costos y la viabilidad técnica de los proyectos de analítica y procesamiento de datos.

La implementación práctica en MongoDB evidenció la capacidad de las bases documentales para gestionar datos heterogéneos y realizar operaciones de consulta y agregación de forma eficiente. El diseño de la base de datos, la inserción de información y la ejecución de consultas avanzadas permitieron comprender cómo este modelo facilita tanto la flexibilidad esquemática como el análisis estadístico mediante pipelines de agregación. Esto reafirma la utilidad de MongoDB como una herramienta robusta para sistemas modernos de comercio electrónico, analítica operativa y aplicaciones donde la adaptabilidad del modelo de datos es un requisito fundamental.

Referencias bibliograficas.

Abadi, D., Boncz, P., & Harizopoulos, S. (2009). Column#oriented database systems. *Proceedings of the VLDB Endowment*, *2*(2), 1664#1665. https://doi.org/10.14778/1687553.1687625

Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, *40*(1), 1#39. https://doi.org/10.1145/1322432.1322433

Carrascal, F. L., & Varona, M. A. (2024). *Bases de datos NoSQL* [Objeto Virtual de Aprendizaje]. Repositorio Institucional Universidad Nacional Abierta y a Distancia (UNAD). https://repository.unad.edu.co/handle/10596/62906

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, *39*(4), 12#27. https://doi.org/10.1145/1978915.1978919

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, *26*(2), Artículo 4. https://doi.org/10.1145/1365815.1365816

Chodorow, K., & Dirolf, M. (2010). *MongoDB: The definitive guide* (1st ed.). O'Reilly Media, Inc.

Contreras García, J. M., & Cabrera Lozano, R. (2024). Visualización de datos con Power BI. *Epsilon: Revista de la Sociedad Andaluza de Educación Matemática "Thales"*, *41*(117), 7#30.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., & Vogels, W. (2007). Dynamo: Amazon's highly available key#value store. *ACM SIGOPS Operating Systems Review*, *41*(6), 205#220. https://doi.org/10.1145/1323293.1294281

Domínguez#Sal, D., Urbón#Bayes, P., Giménez#Vañó, A., Gómez#Villamor, S., Martínez#Bazán, N., & Larriba#Pey, J. L. (2010). Survey of graph database performance on the HPC scalable graph analysis benchmark. En J. X. Yu, M. H. Kim, & R. Unland (Eds.), *Database systems for advanced applications* (pp. 37#48). Springer. https://doi.org/10.1007/978#3#642#16720#1_3

Ehrlinger, L., & Wöß, W. (2016). Towards a definition of knowledge graphs. En M. Martin, M. Cuquet, & E. Folmer (Eds.), *Joint Proceedings of the Posters and Demos Track of 12th International Conference on Semantic Systems # SEMANTiCS2016 and 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16)* (Vol. 48, pp. 1#4). CEUR Workshop Proceedings. http://ceur#ws.org/Vol#1695/paper4.pdf

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition#tolerant web services. *ACM SIGACT News*, *33*(2), 51#59. https://doi.org/10.1145/564585.564601

Gutiérrez Hernández, N. I., & Ibarra Limas, E. (2024). Base de datos para proyectos de Big Data. *Congreso Internacional de Investigación Academia Journals*, *16*(1), 1#7.

Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. En *2011 6th International Conference on Pervasive Computing and Applications* (pp. 363#366). IEEE. https://doi.org/10.1109/ICPCA.2011.6106531

Jiménez, J. H. (2021). *Presentación de datos en Power BI* [Objeto Virtual de Información]. Repositorio Institucional Universidad Nacional Abierta y a Distancia (UNAD). https://repository.unad.edu.co/handle/10596/42071

Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. En *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB)*, Artículo 7, 1#7.

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, *44*(2), 35#40. https://doi.org/10.1145/1773912.1773922

Miranda, A., Oña, P., Mendoza, K., & Chango, W. (2023). Evaluación comparativa de bases de datos NoSQL: clave/valor en un entorno de creaciones de aplicaciones. *ESPOCH Congresses: The Ecuadorian Journal of S.T.E.A.M*, *3*(2), 129#142. https://doi.org/10.18502/espoch.v3i2.15813

Picher Vera, D., Martínez María#Dolores, S. M., & Bernal García, J. J. (2018). Big Data en la universidad y Power BI como el software más óptimo para alumnos universitarios: Análisis y herramientas. *3C Tecnología: Glosas de innovación aplicadas a la pyme*, *7*(1), 46#61. https://doi.org/10.17993/3ctecno.2018.v7n1e25.46#61

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: New opportunities for connected data* (2nd ed.). O'Reilly Media.

Sadalage, P. J., & Fowler, M. (2012). *NoSQL distilled: A brief guide to the emerging world of polyglot persistence*. Addison#Wesley Professional.

Sarasa Cabezuelo, A. (2016). Introducción a las bases de datos NoSQL usando MongoDB. *Revista Española de Innovación, Calidad e Ingeniería del Software*, *12*(2), 69#87.

Stonebraker, M. (2010). SQL databases v. NoSQL databases. *Communications of the ACM*, *53*(4), 10#11. https://doi.org/10.1145/1721654.1721659

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). A comparison of a graph database and a relational database: A data provenance perspective. En *Proceedings of the 48th Annual Southeast Regional Conference* (Artículo 42, pp. 1#6). ACM. https://doi.org/10.1145/1900008.1900067

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*, 57#81. https://doi.org/10.1016/j.aiopen.2021.01.001