Proyecto de Grado

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito final para obtener el Título de INGENIERO DE SISTEMAS

EVALUACIÓN DE POSTGRESQL COMO ALTERNATIVA A UNA BASE DE DATOS NOSQL BASADA EN DOCUMENTOS

Por

Br. Kenia del V. Vergara H.

Tutor: Solazver Solé Álvarez.

Noviembre, 2019

EVALUACIÓN DE POSTGRESQL COMO ALTERNATIVA A UNA BASE DE DATOS NOSQL BASADA EN DOCUMENTOS

Br. Kenia del V. Vergara H.

Proyecto de Grado – Sistemas Computacionales, 50 páginas
Escuela de Ingeniería de Sistemas, Universidad de Los Andes, 2019

Resumen: El contenido de este trabajo se fundamenta en la evaluación comparativa de PostgreSQL (JSONB) y MongoDB (BSON) usando datos JSON, dicha evaluación se enmarcó en la metodología puntos de referencia o benchmarking para obtener el mejor rendimiento de los manejadores. Cabe destacar, la evaluación se constituyó en las siguientes etapas. En la primera, se instaló la herramienta pg_nosql_benchmark, postgresql 10.8, mongodb 4.0.9, bc, git, mongo-tools, apache2, phppgadmin y compass. En la segunda, se eligió un caso de estudio compuesto por tres cargas de trabajo de 100, 1.000 y 100.000 registros o documentos. En la tercera, se creó una plantilla con un generador llamado JSON GENERATOR, el cual contiene una serie de etiquetas preestablecidas para generar documentos en línea. En la cuarta, se modificaron los archivos pg_nosql_benchmark, common_func_lib.sh, pg_func_lib.sh y mongo_func_lib.sh pertenecientes a la herramienta. Finalmente, se observó que el tiempo de respuesta es menor en PostgreSQL cuando maneja poca cantidad de datos y en MongoDB cuando maneja gran cantidad; por otra parte, en el almacenamiento MongoDB ocupa menos espacio que PostgreSQL ya sea utilizando pocos o muchos datos.

Palabras claves: Benchmarking, BSON, JSON, JSONB, MongoDB, NoSQL, PostgreSQL.

Índice

Lista de tablas

Lista de figuras

Capítulo 1

1. Introducción

Hoy en día la necesidad de almacenar grandes vólumenes de información es de vital importancia para la sociedad. Por ello, las bases de datos juegan un papel primordial, ya que proporcionan disponibilidad inmediata, ahorro de espacio físico, fácil mantenimiento, copias de seguridad, integridad de los datos, entre otros.

Cabe señalar, en el universo de las bases de datos existen dos vertientes que son, las bases de datos relacionales y las no relacionales o NoSQL, estas últimas surgen por las deficiencias encontradas en los modelos relacionales para manejar grandes vólumenes de información de una manera rápida y eficaz.

El objetivo principal de este proyecto de grado, es evaluar el comportamiento de las características NoSQL en PostgreSQL comparándolo con una base de datos NoSQL basada en documentos como MongoDB. Dicho trabajo, se basó en la instalación de la herramienta pg_nosql_benchmark desarrollada por EnterpriseDB Corporation, donde se realizó una serie de modificaciones a los archivos pg_nosql_benchmark, common_func_lib.sh, pg_func_lib.sh y mongo_func_lib.sh, pertenecientes a dicha herramienta para obtener los tiempos de respuestas de PostgreSQL y MongoDB, así como también, el tamaño que ocupa la tabla y la colección respectivamente.

Es importante destacar, que dicha evaluación se enmarcó en la metodología puntos de referencia o benchmarking, el cual es un proceso continuo de comparación para obtener el mejor rendimiento de los gestores. Según, Sim, Easterbrook & Holt (2003) señalan que: “El benchmarking debe cumplir con siete propiedades las cuales son, accesibilidad, asequibilidad, claridad, pertinencia, solubilidad, portabilidad y escalabilidad” (p.77).

El presente proyecto está conformado por cinco capítulos. En el primer capítulo, se describe el planteamiento del problema, los objetivos, metodología, alcance,

justificación y los antecedentes. En el segundo capítulo, se refiere al marco teórico. En el tercer capítulo, el análisis y diseño de casos de prueba. En el cuarto capítulo, la implementación. Finalmente, en el quinto capítulo, los resultados.

1.1 Planteamiento del Problema

Actualmente, el empleo de datos semiestructurados para el almacenamiento de la información es una tendencia. Por lo tanto, han surgido gestores especializados en manipular dichos datos, como son las bases de datos no relacionales o NoSQL, que ofrecen características de escalabilidad y velocidad en tiempos de respuesta, superiores a las bases de datos relacionales.

Cabe destacar, que el gestor de bases de datos PostgreSQL ha ido incorporando características NoSQL como la inclusión del tipo de dato JSON en la versión 9.2 y posteriormente el tipo de dato JSONB en la versión 9.4, así como también, los operadores modificadores para cada tipo de dato en la versión 9.3 y 9.5 respectivamente, todo esto encaminado a agilizar y flexibilizar la manipulación de los datos.

Por otra parte, MongoDB es un gestor de base de datos NoSQL de código abierto, el cual trabaja sobre una base de datos de documentos JSON (JavaScript Object Notation), es decir, que en lugar de guardar los datos en registros, los guarda en documentos y dichos documentos son almacenados en un formato binario llamado BSON. MongoDB almacena en sus bases de datos, un conjunto de colecciones constituidas por documentos que pueden tener esquemas diferentes.

Finalmente, el problema que se va a abordar en este trabajo, es comprobar si PostgreSQL satisface las mismas necesidades que MongoDB, los cuales representan modelos de datos distintos, así como también, determinar el mejor rendimiento entre ambos manejadores en relación a, tiempo de respuesta del copy vs import, insert, select vs find, update y el tamaño que ocupa la tabla y la colección.

1.2 Objetivos

1.2.1 Objetivo General

Evaluar el comportamiento de las características NoSQL en PostgreSQL comparándolo con una base de datos NoSQL basada en documentos como MongoDB.

1.2.2 Objetivos Específicos

1. Analizar las características NoSQL del gestor de bases de datos relacional PostgreSQL.
2. Revisar un gestor de bases de datos NoSQL basado en documentos.
3. Definir benchmarks con operaciones de consultas equivalentes en PostgreSQL y en MongoDB.
4. Ejecutar los benchmarks en un ambiente operativo con las últimas versiones disponibles de PostgreSQL y MongoDB.
5. Comparar resultados de PostgreSQL y MongoDB para consultas y manipulación de datos JSON.

1.3 Metodología

En las ciencias de la computación se han utilizado puntos de referencia o benchmarking para comparar el rendimiento de sistemas informáticos, algoritmos de recuperación de información, bases de datos y muchas otras tecnologías.

> Según, Sim et al. (2003), en su trabajo titulado "Usando Benchmarking para Avanzar en la Investigación: Un Desafío para la Ingeniería de Software", presentan siete propiedades que los puntos de referencia o bechmarking deben tener: accesible (el punto de referencia debe ser fácil de obtener y de usar), asequible (debe ser acorde con los beneficios), claridad (la especificación deber ser clara, autónoma y lo más breve posible), pertinencia (la tarea establecida en el punto de referencia deber ser representativa), soluble (debe producir una buena solución), portable (debe especificarse a un nivel de abstracción lo suficientemente alto para garantizar que sea portátil para diferentes herramientas y que no influya a favor de una tecnología) y escalable (las tareas del punto de referencia deben escalar para trabajar con herramientas o técnicas en diferentes niveles de madurez) (p.77).

Cabe destacar, que en este trabajo se aplicará dicha técnica de evaluación que tiene asociada las siguientes fases:

- Primera, se elige un caso de estudio, es decir, un conjunto de datos que se representarán en el modelo de documentos de PostgreSQL y MongoDB.
- Segunda, se instala PostgreSQL y MongoDB, se crean las bases de datos, tabla o colección según el manejador.
- Tercera, se almacenan los datos del caso de estudio en los manejadores (definiendo varias cargas de trabajo 100, 1.000 y 100.000 registros o documentos)
- Cuarta, se diseñan los benchmarks, con operaciones de consultas equivalentes en PostgreSQL y en MongoDB para tomar las métricas de rendimiento.
- Quinta, se ejecutan los benchmarks.
- Sexta, se comparan los resultados.
- Séptima, se realizan las conclusiones de la evaluación.

1.4 Alcance

La evaluación comparativa de PostgreSQL (JSONB) y MongoDB (BSON), se centrará en tres cargas de trabajos constituidas por 100, 1.000 y 100.000 registros o documentos para los manejadores antes descritos. Estás cargas se delimitaron en un rango de valores mínimos, medios y máximos para obtener tiempos de respuestas distintos y de esta manera determinar el mejor rendimiento de cada gestor.

A continuación, se especifica la cantidad de consultas select y sentencias update definidas para cada carga de trabajo en los manejadores, las cuales son: 20 select y 10 update para la carga de 100 registros, 80 select y 100 update para la carga de 1.000 registros y finalmente, 120 select y 300 update para la carga de 100.000 registros o documentos. Es importante acotar, que la cantidad de select y update fue incrementada de acuerdo al volumen de los datos.

1.5 Justificación

Actualmente, las bases de datos NoSQL han ganado espacio especialmente por la escalabilidad y velocidad en sus tiempos de respuestas, superiores a los sistemas relacionales. Es importante señalar, las bases de datos NoSQL se basan en el teorema CAP (consistencia, disponibilidad y tolerancia a particiones), el cual plantea que un sistema de datos distribuidos puede contar con dos de las tres propiedades antes descritas; en cambio, las bases de datos relacionales se basan en las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad). No obstante, las bases de datos NoSQL no se solapan con las relacionales ya que cada tipo garantiza las funcionalidades para las que fueron desarrolladas.

Este proyecto se realizó con la finalidad de evaluar el comportamiento de las características NoSQL en PostgreSQL comparándolo con una base de datos NoSQL basada en documentos como MongoDB. Para comprobar si PostgreSQL arroja tiempos de respuestas menores que MongoDB, así como también, si ocupa menos espacio de almacenamiento, logrando de esta manera un manejador más completo donde convergen las dos tecnologías SQL y NoSQL.

1.6 Antecedentes

Hanlon et al. (2015), en su trabajo titulado “Un Caso de Estudio para Aplicaciones NoSQL y Beneficios de Rendimiento: CouchDB vs. Postgres”, utilizaron un modelo de datos de prueba simple. Dicho trabajo, se fundamentó en varias etapas. En la primera, compararon inserciones de una y varias filas. En la segunda, compararon el número de consultas completadas en CouchDB y Postgres para aumentar el número de usuarios simultáneos. En la tercera, compararon el tiempo promedio de ejecución de consultas en un entorno de lectura / escritura. En cada etapa concluyen, que CouchDB superó a Postgres en inserciones de una sola fila, pero Postgres superó a CouchDB en las inserciones de múltiples filas. Del mismo modo, CouchDB completó más rápido las consultas que Postgres e incrementó el número de usuarios simultáneos. Inclusive, CouchDB obtuvo un gran rendimiento en el entorno de lectura/escritura. Finalmente señalan, que las bases de datos NoSQL no son un reemplazo de las bases de datos

relacionales, pues, en muchos casos tener una base de datos relacional sigue siendo necesaria, debido a que no todos los datos se pueden componer de documentos.

Por otra parte, Fotache & Cogean (2013), en su trabajo titulado "Bases de Datos NoSQL para Aplicaciones Móviles. Estudio de Caso: MongoDB versus PostgreSQL", utilizaron un esquema de base de datos relacional para PostgreSQL y una colección de documentos para MongoDB; dado que, el modelo de datos es diferente la conexión a cada manejador es distinta; del mismo modo, las diferencias entre los lenguajes de definición de datos y manipulación de datos de ambos manejadores. En este trabajo concluyen, que las bases de datos relacionales para aplicaciones móviles poseen dos puntos débiles como son, la rigidez del esquema y la imposibilidad de manejar todos los casos de uso diferentes que requieren dichas aplicaciones, en cambio, las no relacionales proporcionan escalabilidad y velocidad cuando las aplicaciones móviles administran gran cantidad de datos en un servidor central.

Kaur & Rani (2013), en su trabajo titulado "Modelado y Consulta de Datos en Bases de Datos NoSQL", utilizaron tres de bases de datos, la relacional para PostgreSQL, de documentos para MongoDB y de grafos para Neo4j. Luego, representaron los modelos de datos en los diversos diagramas de, entidad relación, clases y grafos respectivamente. Seguidamente, consideraron siete consultas de diversas complejidades para el caso de estudio, las cuales fueron representadas en los lenguajes, SQL, MQL y Cypher. En este trabajo concluyen, que las bases de datos no relacionales brindan mejoras sobre las relacionales, como rendimiento, flexibilidad y escalabilidad, del mismo modo, señalan que las bases de datos no relacionales no deben considerarse como un reemplazo de las relacionales, sino como un complemento.

Por otro lado, Kumar (2014), en su trabajo titulado "pg_nosql_benchmark" es una herramienta para la evaluación comparativa de PostgreSQL (JSONB) y MongoDB (BSON) utilizando datos JSON. Copyright (c) 2013-2014, EnterpriseDB Corporation. Dicha herramienta genera un gran conjunto de documentos JSON, los cuales se cargan en PostgreSQL y MongoDB usando el comando INSERT, de la misma manera, se

ejecutan 4 consultas SELECT simples en los gestores y finalmente, se visualiza un reporte con los tiempos de respuesta del, COPY vs IMPORT, INSERT, SELECT, así como también, el tamaño que ocupa la tabla y la colección.

Capítulo 2

2. Marco Teórico

2.1 NoSQL

Es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales) en aspectos importantes, siendo el más destacado que no usan SQL como lenguaje principal de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad) y habitualmente escalan bien horizontalmente (NoSQL, s.f)

2.1.1 Tipos de base de datos NoSQL

Bases de datos de documentos

Este tipo almacena la información como un documento, generalmente utilizando para ello una estructura simple como JSON o XML y donde se utiliza una clave única para cada registro. Este tipo de implementación permite, además de realizar búsquedas por clave–valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de este tipo son: MongoDB o CouchDB (Bases de datos NoSQL, s.f)

Figura 1. Bases de datos de documentos.

Bases de datos de grafos

En este tipo de bases de datos, la información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada

relación dos. Este tipo de bases de datos ofrece una navegación más eficiente entre relaciones que en un modelo relacional. Algunos ejemplos de este tipo son: Neo4j, InfoGrid o Virtuoso (Bases de datos NoSQL, s.f)

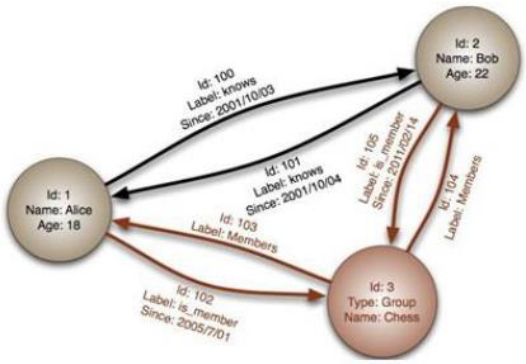


Figura 2. Bases de datos de grafos.

Bases de datos clave-valor

Son el modelo de base de datos NoSQL más popular, además de ser la más sencilla en cuanto a funcionalidad. En este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB). Se caracterizan por ser muy eficientes tanto para las lecturas como para las escrituras. Algunos ejemplos de este tipo son: Cassandra, BigTable o HBase (Bases de datos NoSQL, s.f)

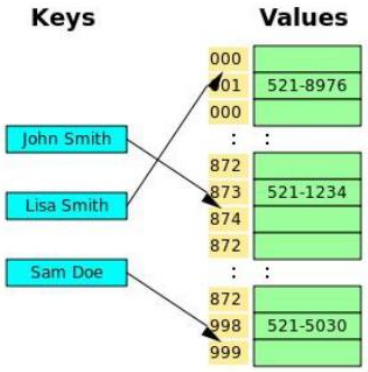


Figura 3. Bases de datos clave-valor.

Bases de datos columnares

Como su nombre lo indica, guardan los datos en columnas en lugar de filas. Por ejemplo, tendríamos una tabla como la que se muestra en la figura 1, mientras que en una base orientada a columnas tendríamos las tablas que muestra la figura 4. Con este cambio ganamos mucha velocidad en lecturas, ya que si queremos consultar un número reducido de columnas, es muy

rápido hacerlo. Al final tenemos una base muy parecida a la clave-valor. Por otro lado, este paradigma no es muy eficiente para realizar escrituras. Por ello este tipo de soluciones es usado en aplicaciones con un índice bajo de escrituras, pero muchas lecturas (Camacho, s.f)

ROWID	Matrícula
1	6548 HCF
2	6589 GDB
3	3215 FGD
4	4836 DVN

ROWID	Modelo
1	Fiat Bravo
2	VW Passat
3	Ford Fiesta
4	Audi A6

ROWID	Precio
1	9861
2	12500
3	4589
4	8956

Figura 4. Bases de datos columnares.

2.1.2 Ventajas de las bases de datos NoSQL

Soportan estructuras distribuidas. Suelen ser bases de datos mucho más abiertas y flexibles. Permiten adaptarse a necesidades de proyectos mucho más fácil que los modelos de entidad relación. Se pueden hacer cambios de los esquemas sin tener que parar las bases de datos. Escalabilidad horizontal, es decir, son capaces de crecer en número de máquinas. Se pueden ejecutar en máquinas con pocos recursos. Optimización de consultas en base de datos para grandes cantidades de datos (Suárez, 2015)

2.1.3 Desventajas de las bases de datos NoSQL

No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos. Soportan lo que se llama consistencia eventual. Problemas de compatibilidad entre instrucciones SQL. Las nuevas bases de datos utilizan sus propias características en el lenguaje de consulta y no son 100% compatibles con el SQL de las bases de datos relacionales. Falta de estandarización. Hay muchas bases de datos NoSQL y aún no hay un estándar como sí lo hay en las bases de datos relacionales. Soporte multiplataforma. Aún quedan muchas mejoras en algunos sistemas para que soporten sistemas operativos que no sean Linux.
Suelen tener herramientas de administración no muy usables o se accede por consola (Suárez, 2015)

2.2 JSON (JavaScript Object Notation)

"Es un formato de texto ligero para el intercambio de datos, está formado como pares de campo/valor. En los documentos JSON, los campos y los valores están encerrados por comillas dobles, separados por dos puntos, así como también, por comas y los conjuntos de campos están encapsulados en llaves" (Introduction to MongoDB, s.f)

A continuación, se visualiza un documento JSON:

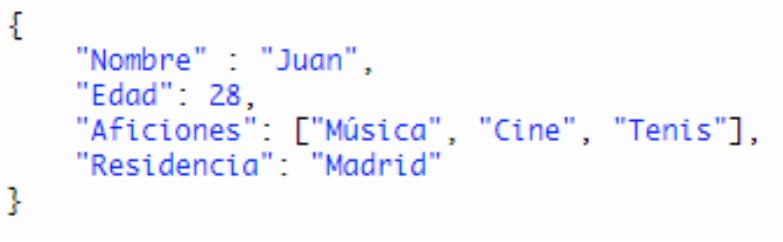


Figura 5. Documento JSON.

2.3 PostgreSQL

Es un gestor de base de datos relacional, su licencia y desarrollo es de código abierto, siendo mantenida por una comunidad de desarrolladores, colaboradores y organizaciones comerciales de forma libre y desinteresada. Esta comunidad es denominada PGDG (PostgreSQL Global Development Group). Utiliza SQL como lenguaje de consulta estructurada para administrar y recuperar la información. (PostgreSQL, s.f)

A continuación, se describen las características más destacadas de PostgreSQL:

Presenta un sistema de alta concurrencia: Presenta un sistema denominado MVCC, el cual permite que mientras un proceso escribe una tabla, otros puedan acceder a la misma tabla sin necesidad de verse bloqueados, y cada usuario obtiene una visión consistente.

Sistema Hot Standby: Este proceso permite a los usuarios poder conectarse con el servidor y ejecutar búsquedas en la base de datos, mientras la misma está en modo de recuperación o stand by. También, se puede pasar de este modo a modo normal sin detener el flujo de búsquedas o consultas de los usuarios, manteniendo las conexiones abiertas. Esto es posible únicamente cuando la base de datos se encuentra en modo de solo lectura.

Soporte nativo: PostgreSQL presenta soporte nativo para los siguientes tipos de datos: texto de largo ilimitado, números de precisión arbitraria, figuras geométricas con funciones asociadas, direcciones MAC, protocolos de direcciones IP (tanto IPv4 como IPv6), bloques de direcciones CDIR, arrays y tipos de datos propios de los usuarios.

Uso de formato JSON: El formato JSON se convierte en el punto débil de muchos sistemas de bases de datos relacionales. Sin embargo, PostgreSQL presenta buenas herramientas con las que es posible indexar elementos y realizar búsquedas en dicho formato. Aunque no se recomienda manejar toda la base de datos en JSON, y se utiliza para el guardado de información de algunos elementos e indexar sus propiedades.

Notificaciones a tiempo real: A pesar de que PostgreSQL no fue diseñada para ser una BD que trabaje al 100% en tiempo real, si es posible mantener sincronizado en varios dispositivos un sistema de notificación para cuando se hacen cambios específicos en la base de datos, gracias a las funciones LISTEN, UNLISTEN y NOTIFY.

Registro y guardado de transacciones: Una de las características más interesantes de PostgreSQL, es su capacidad de registrar cada transacción en un WAL (Write Ahead Log). Esto permite restaurar la base de datos a cualquier punto previamente guardado, una especie de Checkpoint. Esto permite que no sea necesario realizar respaldos completos de forma frecuente, en especial para los casos en los que se trabaja con una base de datos que es muy grande o que contiene mucha cantidad de datos.
Disparadores o triggers: se define como la ejecución de un procedimiento almacenado, basado en una acción determinada sobre una tabla específica en la base de datos. (PostgreSQL, 2019)

2.3.1 JSONB en PostgreSQL

Son datos que se almacenan en forma binaria descompuesta, es decir, no como una cadena ASCII/UTF-8, sino como un código binario. JSONB tiene una serie de ventajas las cuales son: de rápido procesamiento, admite diseños de esquemas más simples (reemplazando el número de tablas por solo una de tipo jsonb donde se almacenan todos los datos); sin embargo, cuenta con una serie de desventajas las cuales son: de entrada ligeramente más lenta, puede tomar más espacio en el disco duro debido a la superficie de la tabla, ciertas consultas pueden ser más lentas debido a que PostgreSQL guarda estadísticas descriptivas como el número de valores distintos y comunes, un histograma de la distribución de datos (Del Alba, 2017)

2.3.2 Modelo de documentos en PostgreSQL

Se refiere a la estructura del documento en el gestor de base de datos. En este caso, se constituye por: la base de datos, la tabla y los registros. Es importante señalar, que para consultar, actualizar y eliminar los registros, se deben utilizar los operadores que se observan en la siguiente tabla (JSON Fnctions and Operators, s.f)

Tabla 1
Operadores para el modelo de documentos en PostgreSQL.

Operadores JSON	**Descripción**
->	Obtener campo objeto json por clave.
->>	Obtener campo objeto json como texto.
#>	Obtener objeto json en la ruta especifica.
#>>	Obtener objeto json en la ruta especifica como texto.
Operadores JSONB	**Descripción**
@>	Obtener el objeto json que tiene la clave y el valor asociado.
?	Obtener los objetos json que tienen la clave asociada.
?|	Obtener los objetos json que tienen algunas de las claves asociadas.
?&	Obtener los objetos json que tienen todas las claves asociadas.
||	Permite concatenar dos jsonb.
	Permite actualizar el valor de un campo clave.
–	Permite eliminar la clave/valor.

2.4 MongoDB

> Es un sistema de bases de datos no relacionales, multiplataforma e inspirada en el tipo de bases de datos documental, admite esquemas flexibles, su nombre proviene del término en inglés "humongous". Está bajo licencia de software libre, específicamente GNU AGPL 3.0. Utiliza MQL como lenguaje de consulta, el cual brinda el rendimiento de las consultas nativas con la productividad de SQL. El entorno de MongoDB, está constituido por: documentos y colecciones. Un documento, es un conjunto de datos estructurados (sin un esquema estricto), que contiene pares clave-valor, el tamaño de un documento está limitado a 16MB, son almacenados en BSON, estos pueden ser comparados con los registro en una base de datos relacional. La colección, es un conjunto de documentos, similar a una tabla en las bases de datos relacionales. (Graterol, s.f)

A continuación, se describen las características más importantes de MongoDB:

> **Alto rendimiento:** se basa en dos puntos, la posibilidad de tener documentos con la información anidada, evitando, de esta forma, un número elevado de operaciones de entrada-salida. Y el soporte de índices y la posibilidad de crear índices sobre arrays y subdocumentos.
> **Alta disponibilidad:** la proporciona mediante la réplica automática conocida como replica set, la cual proporciona redundancia de datos y failover automático, es decir, la transferencia automática a un nuevo nodo cuando se encuentra un fallo en uno de los nodos.
> **Escalado horizontal:** el sistema de sharding permite distribuir información por diferentes máquinas (Características de MongoDB, s.f)

2.4.1 BSON en MongoDB

"Es un formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos MongoDB. Es una representación binaria de estructuras de datos y mapas. El nombre BSON está basado en el término JSON y significa Binary JSON" (BSON, s.f)

2.4.2 Modelo de documentos en MongoDB

Se refiere a la estructura del documento en el manejador de base de datos. En este caso, se constituye por: la base de datos, la colección y los documentos. Es importante destacar, que para consultar, actualiza y eliminar los documentos, se deben utilizar los operadores que se describren en la siguiente tabla (Query and Projection Operators, s.f)

Tabla 2
Operadores de consulta en MongoDB.

Nombre	Descripción
$eq	Coincide con valores que son iguales a un valor especificado.
$gt	Coincide con los valores que son mayores que un valor especificado.
$gte	Coincide con los valores que son mayores o iguales a un valor especificado.
$in	Coincide con cualquiera de los valores especificados en una matriz.
$lt	Coincide con valores que son menores que un valor especificado.
$lte	Coincide con valores que son menores o iguales a un valor especificado.
$ne	Coincide con todos los valores que no son iguales a un valor especificado.
$and	Devuelve todos los documentos que coinciden con las condiciones de ambas cláusulas.
$or	Devuelve todos los documentos que coinciden con las condiciones de las cláusulas.
$exists	Coincide con los documentos que tienen el campo especificado.
$regex	Selecciona documentos donde los valores coinciden con una expresión regular especificada.
$set	Establece el valor de un campo en un documento.
$unset	Elimina el campo especificado de un documento.

2.5 Documentos embebidos en PostgreSQL y MongoDB

Capturan las relaciones de los datos en estructuras de documentos, tales como en subdocumentos o arreglos de documentos. En general, se utilizan documentos embebidos cuando:

> Existan relaciones contenidas entre dos entidades; es decir, que dos entidades independientes sean habitualmente accedidas a través de solo una de ellas. Por ejemplo, *Persona* y *Domicilio*. Cuando únicamente se permitan búsquedas por *Persona* para obtener su dirección. En ese caso, es preferible tener el objeto *Dirección* embebido dentro de *Persona* (Modelado One-to-One, s.f)

Figura 6. Relación Persona – Domicilio.

A continuación, se visualiza la entidad domicilio como subdocumento de persona.

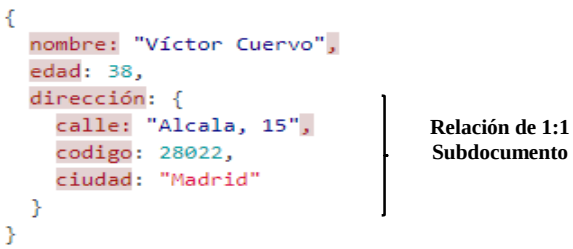


Figura 7. Relación uno a uno embebido.

"Cuando exista una relación uno a muchos entre entidades. En este caso, la parte de muchos suele ir embebida dentro de la de uno" (Modelado One-to-Many, s.f)

Figura 8. Relación Blog – Comentario.

A continuación, se visualiza la entidad comentario como un arreglo de documentos del blog.

```
{
  title: "Línea de Código",
  url: "http://lineadecodigo.com",
  text: "Aprende a Programar",
  comments: [
    {
      name: "Carlos Camacho",
      created_on: ISODate("2015-12-01T10:01:22Z"),
      comment: "Me gusta tu blog"
    },
    {
      name: "Fran Honrubia",
      created_on: ISODate("2015-12-01T14:15:10Z"),
      comment: "Gran trabajo"
    }
  ]
}
```

Relación de 1: N
Arreglo de documentos

Figura 9. Relación uno a muchos embebido.

2.6 Teorema CAP

CAP también conocido como el Teorema de Brewer, se aplica para sistemas distribuidos y se garantiza dos de las tres configuraciones que se mencionan a continuación:

Consistencia (Consistency): todos los nodos deben ver los mismos datos al mismo tiempo. Es decir, cualquier cambio realizado en los datos del sistema se deben aplicar en todos los nodos y debe ser el mismo datos en todos. Esto se llama consistencia atómica y se consigue aplicando la información en todos los nodos

Disponibilidad (Availability): garantiza que cada petición a un nodo reciba una confirmación de si ha sido o no satisfactoriamente.

Tolerancia al Particionado (Partition Tolerance): debe funcionar a pesar de que los nodos tengan un fallo de comunicación, garantizando la disponibilidad a pesar que un nodo se separe del grupo sin importar la causa (Moreno, 2018)

El teorema solo garantiza las siguientes configuraciones:

CA (Consistency & Availability): el sistema siempre estará disponibles respondiendo las peticiones y los datos procesados serán consistentes. En este caso no se puede permitir el particionado.

CP (Consistency & Partition): el sistema aplicara los cambios de forma consistente y aunque se pierda la comunicación entre nodos ocasionando el particionado. No se asegura la disponibilidad entre los nodos.

AP (Availability & Partition): el sistema siempre estará disponible a las peticiones aunque se pierda la comunicación entre los nodos ocasionando el particionado. En consecuencia por la pérdida de comunicación existirá inconsistencia porque no todos los nodos serán iguales (Moreno, 2018)

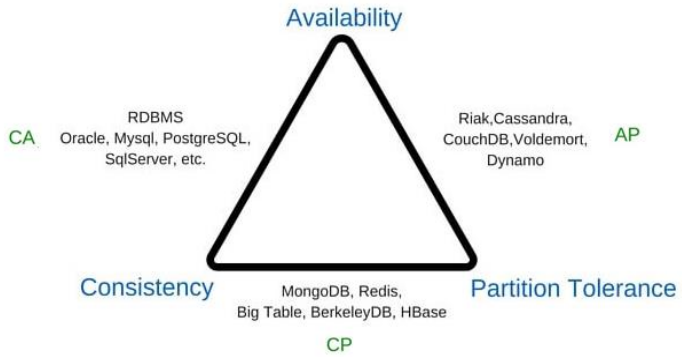


Figura 10. Configuraciones CAP en los sistemas de bases de datos.

Capítulo 3

3. Análisis y diseño de casos de prueba

En este trabajo de investigación, se pretende modificar y crear funciones a la herramienta pg_nosql_benchmark desarrollada para la evaluación comparativa de PostgreSQL (JSONB) y MongoDB (BSON) usando datos JSON (Kumar, 2014).

Cabe señalar, que dicha herramienta está conformada por cuatro archivos que llevan por nombre: pg_nosql_benchmark, common_func_lib.sh, pg_func_lib.sh y mongo_func_lib.sh; El primer archivo, almacena las variables de entorno tanto de PostgreSQL como de MongoDB. El segundo, contiene la semilla de datos json para generar los documentos. El tercero, reúne todas las funciones de PostgreSQL. El cuarto, guarda todas las funciones de MongoDB.

Se determinó para ambos manejadores un caso de estudio que estará constituido por las siguientes entidades, campos y relaciones.

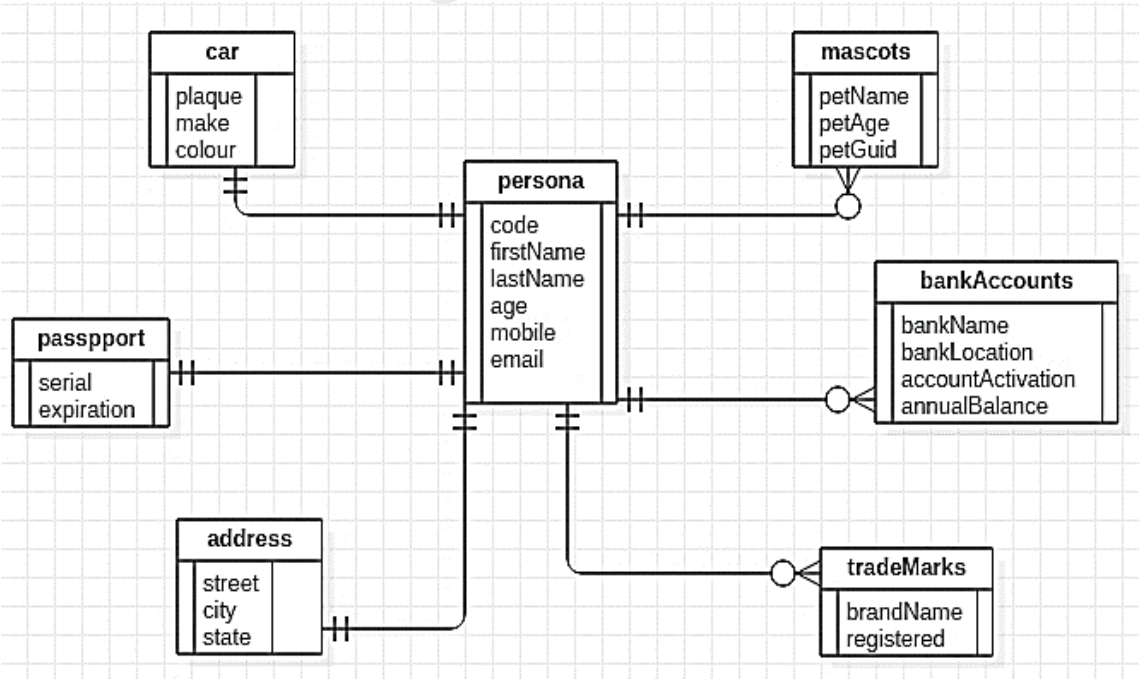


Figura 11. Diagrama del Caso de Estudio.

Ahora bien, las entidades, campos y relaciones antes descritas se representaron en una plantilla de JSON GENERATOR, es un generador en línea de archivos con formato JSON.

A continuación, se visualiza la plantilla:

Figura 12. Plantilla JSON.

Seguidamente, se generó el archivo JSON. A continuación, se observa:

{"code":0,"firstName":"Alta","lastName":"Chaney","age":60,"address":{"street":"Tiffany Place","city":"Edgewater","state":"District Of Columbia"},"mobile":"+1 (923) 468-3067","passport":{"serial":"0693eb1f-259e-4848-8f05-3491f3eef065","expiration":"16-03-2023"},"car":{"plaque":"5a4f237b-0462-4236-828c-8a8abbcee4ff","make":"JEEP","colour":"Black"},"tradeMarks":[{"brandName":"qui","registered":"23-05-2019"},{"brandName":"cillum","registered":"15-03-2008"}],"mascots":[{"petName":"Petra","petAge":14,"petGuid":"78e776eb-deab-47e9-b32b-7bdd9674d4f0"}],"bankAccounts":[{"bankName":"AEORA","bankLocation":"New York","accountActivation":"29-09-2005","annualBalance":"2,404.30$"},{"bankName":"PLASTO","bankLocation":"Kentucky","accountActivation":"19-03-2017","annualBalance":"3,020.78$"}],"email":"petrachaney@plasto.com"}
{"code":1,"firstName":"Bright","lastName":"Gilmore","age":29,"address":{"street":"Wolcott Street","city":"Zeba","state":"Arizona"},"mobile":"+1 (808) 533-2809","passport":{"serial":"686bcc57-5a09-474d-83bb-50e230dfbcb8","expiration":"23-10-2023"},"car":{"plaque":"74f85be4-b972-454b-883c-b80439cc46d0","make":"CHEVROLET","colour":"Green"},"tradeMarks":[{"brandName":"veniam","registered":"31-10-2000"}],"mascots":[{"petName":"Cotton","petAge":11,"petGuid":"d83506a1-6868-4c4d-aca2-c4382fec9b89"},{"petName":"Mayra","petAge":5,"petGuid":"bb1d5600-fcdb-482a-adb5-bbb14eafa1a2"}],"bankAccounts":[{"bankName":"COMCUR","bankLocation":"New Hampshire","accountActivation":"21-05-2002","annualBalance":"1,295.96$"}],"email":"mayragilmore@comcur.com"}
{"code":2,"firstName":"Jacobson","lastName":"Huber","age":67,"address":{"street":"Forbell Street","city":"Falconaire","state":"Alabama"},"mobile":"+1 (831) 460-2670","passport":{"serial":"844f1378-e7fa-4273-badc-49630da697e7","expiration":"28-06-2020"},"car":{"plaque":"0782b6fc-8a34-401f-ae1b-b2fdbc639d2a","make":"TOYOTA","colour":"Black"},"tradeMarks":[{"brandName":"cillum","registered":"07-04-2005"},{"brandName":"ullamco","registered":"18-12-2013"}],"mascots":[{"petName":"Parsons","petAge":11,"petGuid":"44464c1a-5036-49b2-ba55-f50c619b994e"},{"petName":"Dianne","petAge":7,"petGuid":"d111ea02-b825-456a-946f-093ac7a5db24"},{"petName":"Katy","petAge":4,"petGuid":"a7e5e7d7-ae8b-427a-8758-11993ca2c9e5"}],"bankAccounts":[{"bankName":"SAVVY","bankLocation":"Texas","accountActivation":"15-08-2010","annualBalance":"3,350.91$"}],"email":"katyhuber@savvy.com"}
{"code":3,"firstName":"Owen","lastName":"Fry","age":68,"address":{"street":"Woods Place","city":"Weogufka","state":"Georgia"},"mobile":"+1 (875) 587-3054","passport":{"serial":"e03cf64a-e46f-4f1d-94a2-a995ebf48331","expiration":"09-04-2023"},"car":{"plaque":"38574bf4-3606-42de-8b95-271423337e4a","make":"KIA","colour":"Red"},"tradeMarks":[{"brandName":"laborum","registered":"28-06-2001"},{"brandName":"irure","registered":"01-08-2007"}],"mascots":[{"petName":"Randall","petAge":12,"petGuid":"904c1e33-f744-4d08-80fc-921e435b3896"}],"bankAccounts":[{"bankName":"GEEKMOSIS","bankLocation":"Federated States Of Micronesia","accountActivation":"13-04-2006","annualBalance":"1,801.51$"},{"bankName":"MEDICROIX","bankLocation":"Wisconsin","accountActivation":"24-07-2008","annualBalance":"5,766.15$"},{"bankName":"GALLAXIA","bankLocation":"Missouri","accountActivation":"03-06-2015","annualBalance":"1,223.42$"}],"email":"randallfry@gallaxia.com
{"code":4,"firstName":"Blake","lastName":"Bonner","age":32,"address":{"street":"Fountain Avenue","city":"Chase","state":"Minnesota"},"mobile":"+1 (921) 405-3398","passport":{"serial":"46086864-4c9a-4eda-b866-908c33409a85","expiration":"04-03-2023"},"car":{"plaque":"09868dff-a014-4d60-a2ba-efeee98ee930","make":"HYUNDAI","colour":"Blue"},"tradeMarks":[{"brandName":"sint","registered":"08-07-2012"},{"brandName":"ea","registered":"09-12-2001"},{"brandName":"id","registered":"26-04-2013"}],"mascots":[{"petName":"Arlene","petAge":1,"petGuid":"eb1237e4-296c-4933-9ab5-b094f3b069e3"},{"petName":"Michael","petAge":14,"petGuid":"5b06970a-639f-4b9d-b464-34e8b001ae07"}],"bankAccounts":[{"bankName":"COLUMELLA","bankLocation":"Connecticut","accountActivation":"03-01-2018","annualBalance":"3,907.00$"},

Figura 13. Archivo JSON.

Luego, se desarrolló el ScriptA para eliminar la última coma de cada documento y agregar un salto de línea, obteniendo de esta manera el formato definido por PostgreSQL y MongoDB. A continuación, se visualiza el script y los datos.


```
# python
#! /bin/env python

file = open("archivo.json", "r")
file2 = open("archivoModificado.json", "w")

for f in file:
    f = f.replace('.com\"},', '.com\"}\n')

    file2.write(f)

file.close()
file2.close()
```

Figura 14. ScriptA.

```
{"code":0,"firstName":"Alta","lastName":"Chaney","age":60,"address":{"street":"Tiffany Place","city":"Edgewater","state":"District Of
Columbia"},"mobile":"+1 (923) 468-3067","passport":{"serial":"0693eb1f-259e-4848-8f05-3491f3eef065","expiration":"16-03-2023"},"car":
{"plaque":"5a4f237b-0462-4236-828c-8a8abbcee4ff","make":"JEEP","colour":"Black"},"tradeMarks":[{"brandName":"qui","registered":"23-05-2019"},
{"brandName":"cillum","registered":"15-03-2008"}],"mascots":[{"petName":"Petra","petAge":14,"petGuid":"78e776eb-deab-47e9-
b32b-7bdd9674d4f0"}],"bankAccounts":[{"bankName":"AEORA","bankLocation":"New
York","accountActivation":"29-09-2005","annualBalance":"2,404.30$"},
{"bankName":"PLASTO","bankLocation":"Kentucky","accountActivation":"19-03-2017","annualBalance":"3,020.78$"}],"email":"petrachaney@plasto.com"}
{"code":1,"firstName":"Bright","lastName":"Gilmore","age":29,"address":{"street":"Wolcott Street","city":"Zeba","state":"Arizona"},"mobile":"+1
(808) 533-2809","passport":{"serial":"686bcc57-5a09-474d-83bb-50e230dfbcb8","expiration":"23-10-2023"},"car":{"plaque":"74f85be4-
b972-454b-883c-b80439cc46d0","make":"CHEVROLET","colour":"Green"},"tradeMarks":[{"brandName":"veniam","registered":"31-10-2000"}],"mascots":
[{"petName":"Cotton","petAge":11,"petGuid":"d83506a1-6868-4c4d-aca2-c4382fec9b89"},{"petName":"Mayra","petAge":5,"petGuid":"bb1d5600-fcdb-482a-
adb5-bbb14eafa1a2"}],"bankAccounts":[{"bankName":"COMCUR","bankLocation":"New
Hampshire","accountActivation":"21-05-2002","annualBalance":"1,295.96$"}],"email":"mayragilmore@comcur.com"}
{"code":2,"firstName":"Jacobson","lastName":"Huber","age":67,"address":{"street":"Forbell
Street","city":"Falconaire","state":"Alabama"},"mobile":"+1 (831) 460-2670","passport":{"serial":"844f1378-e7fa-4273-
badc-49630da697e7","expiration":"28-06-2020"},"car":{"plaque":"0782b6fc-8a34-401f-ae1b-
b2fdbc639d2a","make":"TOYOTA","colour":"Black"},"tradeMarks":[{"brandName":"cillum","registered":"07-04-2005"},
{"brandName":"ullamco","registered":"18-12-2013"}],"mascots":[{"petName":"Parsons","petAge":11,"petGuid":"44464c1a-5036-49b2-ba55-
f50c619b994e"},{"petName":"Dianne","petAge":7,"petGuid":"d111ea02-b825-456a-946f-093ac7a5db24"},{"petName":"Katy","petAge":
4,"petGuid":"a7e5e7d7-ae8b-427a-8758-11993ca2c9e5"}],"bankAccounts":
[{"bankName":"SAVVY","bankLocation":"Texas","accountActivation":"15-08-2010","annualBalance":"3,350.91$"}],"email":"katyhuber@savvy.com"}
{"code":3,"firstName":"Owen","lastName":"Fry","age":68,"address":{"street":"Woods Place","city":"Weogufka","state":"Georgia"},"mobile":"+1
(875) 587-3054","passport":{"serial":"e03cf64a-e46f-4f1d-94a2-a995ebf48331","expiration":"09-04-2023"},"car":
{"plaque":"38574bf4-3606-42de-8b95-271423337e4a","make":"KIA","colour":"Red"},"tradeMarks":[{"brandName":"laborum","registered":"28-06-2001"},
{"brandName":"irure","registered":"01-08-2007"}],"mascots":[{"petName":"Randall","petAge":12,"petGuid":"904c1e33-
f744-4d08-80fc-921e435b3896"}],"bankAccounts":[{"bankName":"GEEKMOSIS","bankLocation":"Federated States Of
Micronesia","accountActivation":"13-04-2006","annualBalance":"1,801.51$"},
{"bankName":"MEDICROIX","bankLocation":"Wisconsin","accountActivation":"24-07-2008","annualBalance":"5,766.15$"},
{"bankName":"GALLAXIA","bankLocation":"Missouri","accountActivation":"03-06-2015","annualBalance":"1,223.42$"}],"email":"randallfry@gallaxia.com
{"code":4,"firstName":"Blake","lastName":"Bonner","age":32,"address":{"street":"Fountain
Avenue","city":"Chase","state":"Minnesota"},"mobile":"+1 (921) 405-3398","passport":{"serial":"46086864-4c9a-4eda-
b866-908c33409a85","expiration":"04-03-2023"},"car":{"plaque":"09868dff-a014-4d60-a2ba-
efeee98ee930","make":"HYUNDAI","colour":"Blue"},"tradeMarks":[{"brandName":"sint","registered":"08-07-2012"},
{"brandName":"ea","registered":"09-12-2001"},{"brandName":"id","registered":"26-04-2013"}],"mascots":[{"petName":"Arlene","petAge":
1,"petGuid":"eb1237e4-296c-4933-9ab5-b094f3b069e3"},{"petName":"Michael","petAge":14,"petGuid":"5b06970a-639f-4b9d-
b464-34e8b001ae07"}],"bankAccounts":
[{"bankName":"COLUMELLA","bankLocation":"Connecticut","accountActivation":"03-01-2018","annualBalance":"3,907.00$"},
```


Figura 15. Datos con el formato definido por PostgreSQL y MongoDB.

Posteriormente, se desarrolló el ScriptB para agregar una barra (\) antes de las comillas, obteniendo de esta manera el formato definido por la herramienta pg_nosql_benchmark. A continuación, se visualiza el script y los datos.


```
# python
#! /bin/env python

file = open("archivoModificado.json", "r")
file2 = open("formatoFinal.json", "w")

for f in file:
    f = f.replace('\"', '\\\"')
    file2.write(f)

file.close()
file2.close()
```

Figura 16. ScriptB.

Figura 17. Datos con el formato definido por la herramienta.

Capítulo 4

4. Implementación

En está etapa de la investigación se llevó a cabo una serie de pasos, los cuales se describen a continuación, se instaló la herramienta pg_nosql_benchmark, así como también, los paquetes de postgresql 10.8, mongodb 4.0.9, bc, git, mongo-tools, apache2, phppgadmin y compass. Luego, se creó un usuario root en MongoDB, como se observa en la figura 18. Posteriormente, se creó un superusuario en PostgreSQL de la siguiente manera: *CREATE USER kenia WITH LOGIN ENCRYPTED PASSWORD 'kenia' SUPERUSER;*

La creación del usuario root y superusuario, se realizó con la finalidad de tener todos los privilegios para administrar las bases de datos.

```
> show databases;
admin   0.000GB
bcm     0.046GB
bd      0.000GB
bdm     0.005GB
config  0.000GB
json3   0.000GB
local   0.000GB
prueba  0.009GB
> use admin
switched to db admin
> db.createUser({user:"kenia",pwd:"kenia",roles:[{role:"root",db:"admin"}]})
Successfully added user: {
        "user" : "kenia",
        "roles" : [
                {
                        "role" : "root",
                        "db" : "admin"
                }
        ]
}
```


Figura 18. Creación de usuario root en MongoDB.

A continuación, se visualizan las funciones modificadas y creadas en los archivos de la herramienta.

Archivo pg_nosql_benchmark: se modificaron las variables de entorno de PostgreSQL y MongoDB, las cuales permitirán la conexión a las bases de datos. A continuación, se observan:

```
###################################
# set postgre variables.
###################################
PGHOME="/usr"
PGHOST="127.0.0.1"
PGPORT="5432"
PGUSER="kenia"
PGPASSWORD="kenia"
PGDATABASE="benchmark"

PGBIN="/usr/bin"

###################################
# set mongo variables.
###################################
MONGO="/usr/bin/mongo"
MONGOIMPORT="/usr/bin/mongoimport"
MONGOHOST="127.0.0.1"
MONGOPORT="27017"
MONGOUSER="kenia"
MONGOPASSWORD="kenia"
MONGODBNAME="benchmark"
```


Figura 19. Variables de entorno de PostgreSQL y MongoDB.

Del mismo modo, se modificó el valor de la variable json_rows que sirve para generar dinámicamente las tres cargas de trabajo de 100, 1.000 y 100.000 registros o documentos que se almacenaran en PostgreSQL y MongoDB.

```
##############################
# declare require arrays
##############################
declare -a json_rows=(100)

##############################
# declare require arrays
##############################
declare -a json_rows=(1000)

##############################
# declare require arrays
##############################
declare -a json_rows=(100000)
```

Figura 20. Declaración de variable generadora de documentos.

También, se diseño el reporte de los resultados comparativos para PostgreSQL y MongoDB.

```
print_result "------------------------------------------------"
print_result "number of rows"     "${json_rows[@]}"
print_result "------------------------------------------------"
print_result "*** Response Time (ns)                          "
print_result "------------------------------------------------"
print_result "     PostgreSQL COPY  "     "${pg_copy_time[@]}"
print_result "     MongoDB IMPORT   "     "${mongo_copy_time[@]}"
print_result "------------------------------------------------"
print_result "     PostgreSQL INSERT"     "${pg_inserts_time[@]}"
print_result "     MongoDB INSERT   "     "${mongo_inserts_time[@]}"
print_result "------------------------------------------------"
print_result "     PostgreSQL SELECT"     "${pg_select_time[@]}"
print_result "     MongoDB SELECT   "     "${mongo_select_time[@]}"
print_result "------------------------------------------------"
print_result "     PostgreSQL UPDATE"     "${pg_update_time[@]}"
print_result "     MongoDB UPDATE   "     "${mongo_update_time[@]}"
print_result "------------------------------------------------"
print_result "*** Collection or Table Size (bytes)            "
print_result "------------------------------------------------"
print_result "        PostgreSQL"         "${pg_size_time[@]}"
print_result "        MongoDB   "         "${mongo_size_time[@]}"
print_result "------------------------------------------------"
```

Figura 21. Reporte de los resultados comparativos para PostgreSQL y MongoDB.

Archivo common_func_lib.sh

En la **function json_seed_data**, se agregaron los datos con el formato definido por la herramienta, así como también, se añadió al campo code una función random para generar números aleatorios.

```
################################################################################
# function: json_seed_data
################################################################################
function json_seed_data ()
{

     local INDX="$1"
     local SEED_DATA

     if [[ ${INDX} -eq 0 ]]
     then
          INDX=1
     fi
       SEED_DATA="{\"code\":\"AC3$((${RANDOM}/$INDX + $INDX ))\",\"firstName\":\"Ana\",\"lastName\":\"Chaney\",\"age\":60,\"address\":
{\"street\":\"Tiffany Place\",\"city\":\"Edgewater\",\"state\":\"District Of Columbia\"},\"mobile\":\"+1 (923) 468-3067\",\"passport\":
{\"serial\":\"0693eb1f-259e-4848-8f05-3491f3eef065\",\"expiration\":\"16-03-2023\"},\"car\":{\"plaque\":
\"5a4f237b-0462-4236-828c-8a8abbcee4ff\",\"make\":\"JEEP\",\"colour\":\"Black\"},\"tradeMarks\":[{\"brandName\":\"qui\",\"registered\":
\"23-05-2019\"},{\"brandName\":\"cillum\",\"registered\":\"15-03-2008\"}],\"mascots\":[{\"petName\":\"Petra\",\"petAge\":14,\"petGuid\":
\"78e776eb-deab-47e9-b32b-7bdd9674d4f0\"}],\"bankAccounts\":[{\"bankName\":\"AEORA\",\"bankLocation\":\"New York\",\"accountActivation\":
\"29-09-2005\",\"annualBalance\":\"2,404.30$\"},{\"bankName\":\"PLASTO\",\"bankLocation\":\"Kentucky\",\"accountActivation\":\"19-03-2017\",
\"annualBalance\":\"3,020.78$\"}],\"email\":\"petrachaney@plasto.com\"}
{\"code\":\"AC3$((${RANDOM}/$INDX + $INDX ))\",\"firstName\":\"Bright\",\"lastName\":\"Gilmore\",\"age\":29,\"address\":{\"street\":\"Wolcott
Street\",\"city\":\"Zeba\",\"state\":\"Arizona\"},\"mobile\":\"+1 (808) 533-2809\",\"passport\":{\"serial\":
\"686bcc57-5a09-474d-83bb-50e230dfbcb8\",\"expiration\":\"23-10-2023\"},\"car\":{\"plaque\":\"74f85be4-b972-454b-883c-b80439cc46d0\",\"make\":
\"CHEVROLET\",\"colour\":\"Green\"},\"tradeMarks\":[{\"brandName\":\"veniam\",\"registered\":\"31-10-2000\"}],\"mascots\":[{\"petName\":
\"Cotton\",\"petAge\":11,\"petGuid\":\"d83506a1-6868-4c4d-aca2-c4382fec9b89\"},{\"petName\":\"Mayra\",\"petAge\":5,\"petGuid\":\"bb1d5600-
fcdb-482a-adb5-bbb14eafa1a2\"}],\"bankAccounts\":[{\"bankName\":\"COMCUR\",\"bankLocation\":\"New Hampshire\",\"accountActivation\":
\"21-05-2002\",\"annualBalance\":\"1,295.96$\"}],\"email\":\"mayragilmore@comcur.com\"}
{\"code\":\"AC3$((${RANDOM}/$INDX + $INDX ))\",\"firstName\":\"Jacobson\",\"lastName\":\"Huber\",\"age\":67,\"address\":{\"street\":\"Forbell
Street\",\"city\":\"Falconaire\",\"state\":\"Alabama\"},\"mobile\":\"+1 (831) 460-2670\",\"passport\":{\"serial\":\"844f1378-e7fa-4273-
badc-49630da697e7\",\"expiration\":\"28-06-2020\"},\"car\":{\"plaque\":\"0782b6fc-8a34-401f-ae1b-b2fdbc639d2a\",\"make\":\"TOYOTA\",\"colour\":
\"Black\"},\"tradeMarks\":[{\"brandName\":\"cillum\",\"registered\":\"07-04-2005\"},{\"brandName\":\"ullamco\",\"registered\":\"18-12-2013\"}],
\"mascots\":[{\"petName\":\"Parsons\",\"petAge\":11,\"petGuid\":\"44464c1a-5036-49b2-ba55-f50c619b994e\"},{\"petName\":\"Dianne\",\"petAge\":
7,\"petGuid\":\"d111ea02-b825-456a-946f-093ac7a5db24\"},{\"petName\":\"Katy\",\"petAge\":4,\"petGuid\":\"a7e5e7d7-
ae8b-427a-8758-11993ca2c9e5\"}],\"bankAccounts\":[{\"bankName\":\"SAVVY\",\"bankLocation\":\"Texas\",\"accountActivation\":\"15-08-2010\",
```

Figura 22. Datos de la herramienta.

Archivo pg_func_lib.sh

En la **function pg_json_insert_maker**, se insertan en la tabla json_tables los registros que se encuentran en sample_pg_inserts.json.

```
#########################################################################
# function: pg_json_insert_maker
#########################################################################
function pg_json_insert_maker ()

{
   typeset -r COLLECTION_NAME="$1"
   typeset -r NO_OF_ROWS="$2"
   typeset -r JSON_FILENAME="$3"

   process_log "preparing postgresql INSERTs."
   rm -rf ${JSON_FILENAME}
   NO_OF_LOOPS=$((${NO_OF_ROWS}/10))
   for ((i=0;i<${NO_OF_LOOPS};i++))
   do
       json_seed_data $i | \
        sed "s/^/INSERT INTO ${COLLECTION_NAME} VALUES(\$JSON\$/"| \
        sed "s/$/\$JSON\$);/" >>${JSON_FILENAME}
   done
}
```

Figura 23. Inserción de datos JSON en PostgreSQL.

En la **function pg_select_benchmark**, se crearon las consultas SELECT simples y compuestas para cada carga de trabajo respectivamente. A continuación, se visualiza un extracto de estas.

Para 100 registros le corresponde 20 consultas SELECT.

```
#############################################################################
# function: benchmark postgresql select
#############################################################################
function pg_select_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_SELECT1="SELECT data->'firstName', data->'lastName' FROM ${F_COLLECTION};"

   typeset -r F_SELECT2="SELECT *FROM ${F_COLLECTION} WHERE data#>>'{passport,serial}'='0693eb1f-259e-4848-8f05-3491f3eef065';"

   typeset -r F_SELECT3="SELECT * FROM ${F_COLLECTION} WHERE data->'car'->>'plaque' = '74f85be4-b972-454b-883c-b80439cc46d0';"

   typeset -r F_SELECT4="SELECT * FROM ${F_COLLECTION} WHERE data->'passport'->>'serial' = '686bcc57-5a09-474d-83bb-50e230dfbcb8';"

   typeset -r F_SELECT5="SELECT * FROM ${F_COLLECTION} WHERE (data?& array['firstName','email','mascots']);"

   typeset -r F_SELECT6="SELECT * FROM ${F_COLLECTION} WHERE (data?| array['firstName','passport']);"

   typeset -r F_SELECT7="SELECT * FROM ${F_COLLECTION} WHERE (data?'email');"

   typeset -r F_SELECT8="SELECT * FROM ${F_COLLECTION};"

   typeset -r F_SELECT9="SELECT data->>'firstName', data->>'lastName', data->>'email' FROM ${F_COLLECTION};"

   typeset -r F_SELECT10="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'lastName') = 'Chaney';"

   typeset -r F_SELECT11="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '32';"

   typeset -r F_SELECT12="SELECT * FROM ${F_COLLECTION} WHERE  (data->'address'->> 'city') = 'Fredericktown';"

   typeset -r F_SELECT13="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'firstName') = 'Lowery';"
```

Figura 24. Sentencias SELECT para 100 registros en PostgreSQL.

Para 1.000 registros le corresponden 80 consultas SELECT.

```
################################################################################
# function: benchmark postgresql select
################################################################################
function pg_select_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_SELECT1="SELECT * FROM ${F_COLLECTION};"

   typeset -r F_SELECT2="SELECT data->'firstName', data->'lastName' FROM ${F_COLLECTION};"

   typeset -r F_SELECT3="SELECT data->'firstName', data->'email', data->'mascots' FROM ${F_COLLECTION};"

   typeset -r F_SELECT4="SELECT data->'firstName', data->'lastName', data->'email' FROM ${F_COLLECTION};"

   typeset -r F_SELECT5="SELECT data->'firstName', data->'mobile' FROM ${F_COLLECTION};"

   typeset -r F_SELECT6="SELECT * FROM ${F_COLLECTION} WHERE (data?& array['firstName','mobile','mascots']);"

   typeset -r F_SELECT7="SELECT * FROM ${F_COLLECTION} WHERE (data?| array['firstName','lastName:1','age']);"

   typeset -r F_SELECT8="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '60';"

   typeset -r F_SELECT9="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '29';"

   typeset -r F_SELECT10="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '67';"

   typeset -r F_SELECT11="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '68';"

   typeset -r F_SELECT12="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '32';"
```

Figura 25. Sentencias SELECT para 1.000 registros en PostgreSQL.

Para 100.000 registros le corresponden 120 consultas SELECT.

```
################################################################################
# function: benchmark postgresql select
################################################################################
function pg_select_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_SELECT1="SELECT * FROM ${F_COLLECTION};"

   typeset -r F_SELECT2="SELECT data->'firstName', data->'lastName' FROM ${F_COLLECTION};"

   typeset -r F_SELECT3="SELECT data->'firstName', data->'email', data->'mascots' FROM ${F_COLLECTION};"

   typeset -r F_SELECT4="SELECT data->'firstName', data->'lastName', data->'email' FROM ${F_COLLECTION};"

   typeset -r F_SELECT5="SELECT data->'firstName', data->'mobile' FROM ${F_COLLECTION};"

   typeset -r F_SELECT6="SELECT * FROM ${F_COLLECTION} WHERE (data?& array['firstName','mobile','mascots']);"

   typeset -r F_SELECT7="SELECT * FROM ${F_COLLECTION} WHERE (data?| array['firstName','lastName:1','age']);"

   typeset -r F_SELECT8="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '60';"

   typeset -r F_SELECT9="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '29';"

   typeset -r F_SELECT10="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '67';"

   typeset -r F_SELECT11="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '68';"

   typeset -r F_SELECT12="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '32';"

   typeset -r F_SELECT13="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '62';"
```

Figura 26. Sentencias SELECT para 100.000 registros en PostgreSQL.

En la **function pg_update_benchmark**, se crearon sentencias UPDATE para cada carga de trabajo respectivamente. A continuación, se visualiza un extracto de estas.

Para 100 registros le corresponde 10 sentencias UPDATE.

```
#############################################################################
# function: benchmark postgresql update
#############################################################################
function pg_update_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_UPDATE1="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Jacobson';"

   typeset -r F_UPDATE2="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Ana';"

   typeset -r F_UPDATE3="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Bright';"

   typeset -r F_UPDATE4="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Owen';"

   typeset -r F_UPDATE5="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' = 'Blake';"

   typeset -r F_UPDATE6="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Barlow';"

   typeset -r F_UPDATE7="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Faye';"

   typeset -r F_UPDATE8="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Buckley';"

   typeset -r F_UPDATE9="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Fannie';"
```

Figura 27. Sentencias UPDATE para 100 registros en PostgreSQL.

Para 1.000 registros le corresponden 100 sentencias UPDATE.

```
#############################################################################
# function: benchmark postgresql update
#############################################################################
function pg_update_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_UPDATE1="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Jacobson';"

   typeset -r F_UPDATE2="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Ana';"

   typeset -r F_UPDATE3="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Bright';"

   typeset -r F_UPDATE4="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Owen';"

   typeset -r F_UPDATE5="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' = 'Blake';"

   typeset -r F_UPDATE6="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Barlow';"

   typeset -r F_UPDATE7="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Faye';"

   typeset -r F_UPDATE8="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Buckley';"
```

Figura 28. Sentencias UPDATE para 1.000 registros en PostgreSQL.

Para 100.000 registros le corresponden 300 sentencias UPDATE.

```
###############################################################################
# function: benchmark postgresql update
###############################################################################
function pg_update_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_UPDATE1="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Jacobson';"

   typeset -r F_UPDATE2="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Ana';"

   typeset -r F_UPDATE3="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Bright';"

   typeset -r F_UPDATE4="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Owen';"

   typeset -r F_UPDATE5="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' = 'Blake';"

   typeset -r F_UPDATE6="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Barlow';"

   typeset -r F_UPDATE7="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Faye';"

   typeset -r F_UPDATE8="UPDATE ${F_COLLECTION} SET data = data::JSONB || '{\"nationality\":\"American\"}'::JSONB WHERE data->>'firstName' =
'Buckley';"

   typeset -r F_UPDATE9="UPDATE ${F_COLLECTION} SET data = data::jsonb || '{\"nationality\":\"American\"}' WHERE data->>'firstName' =
'Fannie';"
```

Figura 29. Sentencias UPDATE para 100.000 registros en PostgreSQL.

Archivo mongo_func_lib.sh

En la **function mongo_json_insert_maker**, se insertan en la colección json_tables los documentos que se encuentran en sample_mongo_inserts.json.


```
###############################################################################
# function: mongo_insert_maker
###############################################################################
function mongo_json_insert_maker ()
{
   typeset -r COLLECTION_NAME="$1"
   typeset -r NO_OF_ROWS="$2"
   typeset -r JSON_FILENAME="$3"

   rm -rf ${JSON_FILENAME}
   process_log "preparing mongo insert commands."
   NO_OF_LOOPS=$((${NO_OF_ROWS}/10))
   for ((i=0;i<${NO_OF_LOOPS};i++))
   do
       json_seed_data $i | sed "s/^/db.${COLLECTION_NAME}.insert( /" | \
                           sed "s/$/ )/" >>${JSON_FILENAME}
   done
}
```

Figura 30. Inserción de datos JSON en MongoDB.

En la **function mongodb_select_benchmark**, se crearon las consultas FIND simples y compuestas para cada carga de trabajo respectivamente. A continuación, se visualiza un extracto de estas.

Para 100 registros le corresponde 20 consultas FIND.

```
#############################################################################
# function: benchmark mongo-select
#############################################################################
function mongodb_select_benchmark ()
{

   typeset -r F_MONGOHOST="$1"
   typeset -r F_MONGOPORT="$2"
   typeset -r F_MONGODBNAME="$3"
   typeset -r F_MONGOUSER="$4"
   typeset -r F_MONGOPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_MONGOSELECT1="db.${F_COLLECTION}.find({},{firstName:1, lastName:1})"

   typeset -r F_MONGOSELECT2="db.${F_COLLECTION}.find({passport.serial: '0693eb1f-259e-4848-8f05-3491f3eef065'})"

   typeset -r F_MONGOSELECT3="db.${F_COLLECTION}.find({car.plaque: '74f85be4-b972-454b-883c-b80439cc46d0'})"

   typeset -r F_MONGOSELECT4="db.${F_COLLECTION}.find({passport.serial: '686bcc57-5a09-474d-83bb-50e230dfbcb8'})"

   typeset -r F_MONGOSELECT5="db.${F_COLLECTION}.find({},{firstName:1, email:1, mascots:1})"

   typeset -r F_MONGOSELECT6="db.${F_COLLECTION}.find({\$or:[{firstName:/^/},{passport:/^/}]})"

   typeset -r F_MONGOSELECT7="db.${F_COLLECTION}.find({email:1, firstName:1, age:1})"

   typeset -r F_MONGOSELECT8="db.${F_COLLECTION}.find()"

   typeset -r F_MONGOSELECT9="db.${F_COLLECTION}.find({},{firstName:1, lastName:1, email:1})"

   typeset -r F_MONGOSELECT10="db.${F_COLLECTION}.find({lastName: 'Chaney'})"

   typeset -r F_MONGOSELECT11="db.${F_COLLECTION}.find({age: 32})"

   typeset -r F_MONGOSELECT12="db.${F_COLLECTION}.find({address.city: 'Fredericktown'})"
```

Figura 31. Sentencias FIND para 100 documentos en MongoDB.

Para 1.000 registros le corresponden 80 consultas FIND.


```
#############################################################################
# function: benchmark mongo-select
#############################################################################
function mongodb_select_benchmark ()
{

   typeset -r F_MONGOHOST="$1"
   typeset -r F_MONGOPORT="$2"
   typeset -r F_MONGODBNAME="$3"
   typeset -r F_MONGOUSER="$4"
   typeset -r F_MONGOPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_MONGOSELECT1="db.${F_COLLECTION}.find()"

   typeset -r F_MONGOSELECT2="db.${F_COLLECTION}.find({},{firstName:1, lastName:1})"

   typeset -r F_MONGOSELECT3="db.${F_COLLECTION}.find({},{firstName:1, email:1, mascots:1})"

   typeset -r F_MONGOSELECT4="db.${F_COLLECTION}.find({},{firstName:1, lastName:1, email:1})"

   typeset -r F_MONGOSELECT5="db.${F_COLLECTION}.find({},{firstName:1, mobile:1})"

   typeset -r F_MONGOSELECT6="db.${F_COLLECTION}.find({firstName:1, mobile:1, mascots:1})"

   typeset -r F_MONGOSELECT7="db.${F_COLLECTION}.find({firstName:1, lastName:1, age:1})"

   typeset -r F_MONGOSELECT8="db.${F_COLLECTION}.find({age: 60})"

   typeset -r F_MONGOSELECT9="db.${F_COLLECTION}.find({age: 29})"

   typeset -r F_MONGOSELECT10="db.${F_COLLECTION}.find({age: 67})"

   typeset -r F_MONGOSELECT11="db.${F_COLLECTION}.find({age: 68})"

   typeset -r F_MONGOSELECT12="db.${F_COLLECTION}.find({age: 32})"
```

Figura 32. Sentencias FIND para 1.000 documentos en MongoDB.

Para 100.000 registros le corresponden 120 consultas FIND.

```
#################################################################################
# function: benchmark postgresql select
#################################################################################
function pg_select_benchmark ()
{
   typeset -r F_PGHOST="$1"
   typeset -r F_PGPORT="$2"
   typeset -r F_DBNAME="$3"
   typeset -r F_PGUSER="$4"
   typeset -r F_PGPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_SELECT1="SELECT * FROM ${F_COLLECTION};"

   typeset -r F_SELECT2="SELECT data->'firstName', data->'lastName' FROM ${F_COLLECTION};"

   typeset -r F_SELECT3="SELECT data->'firstName', data->'email', data->'mascots' FROM ${F_COLLECTION};"

   typeset -r F_SELECT4="SELECT data->'firstName', data->'lastName', data->'email' FROM ${F_COLLECTION};"

   typeset -r F_SELECT5="SELECT data->'firstName', data->'mobile' FROM ${F_COLLECTION};"

   typeset -r F_SELECT6="SELECT * FROM ${F_COLLECTION} WHERE (data?& array['firstName','mobile','mascots']);"

   typeset -r F_SELECT7="SELECT * FROM ${F_COLLECTION} WHERE (data?| array['firstName','lastName:1','age']);"

   typeset -r F_SELECT8="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '60';"

   typeset -r F_SELECT9="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '29';"

   typeset -r F_SELECT10="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '67';"

   typeset -r F_SELECT11="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '68';"

   typeset -r F_SELECT12="SELECT * FROM ${F_COLLECTION} WHERE  (data->>'age') = '32';"

   typeset -r F_SELECT13="SELECT * FROM ${F_COLLECTION} WHERE  (data->'age') = '62';"
```

Figura 33. Sentencias FIND para 100.000 documentos en MongoDB.

En la **function mongodb_update_benchmark**, se crearon sentencias UPDATE para cada carga de trabajo respectivamente. A continuación, se visualiza un extracto de estas.

Para 100 registros le corresponde 10 sentencias UPDATE.

```
#################################################################################
# function: benchmark mongo-update
#################################################################################
function mongodb_update_benchmark ()
{
   typeset -r F_MONGOHOST="$1"
   typeset -r F_MONGOPORT="$2"
   typeset -r F_MONGODBNAME="$3"
   typeset -r F_MONGOUSER="$4"
   typeset -r F_MONGOPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_MONGOUPDATE1="db.${F_COLLECTION}.update({firstName:\"Jacobson\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE2="db.${F_COLLECTION}.update({firstName:\"Ana\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE3="db.${F_COLLECTION}.update({firstName:\"Bright\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE4="db.${F_COLLECTION}.update({firstName:\"Owen\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE5="db.${F_COLLECTION}.update({firstName:\"Blake\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE6="db.${F_COLLECTION}.update({firstName:\"Barlow\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE7="db.${F_COLLECTION}.update({firstName:\"Faye\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE8="db.${F_COLLECTION}.update({firstName:\"Buckley\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE9="db.${F_COLLECTION}.update({firstName:\"Fannie\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE10="db.${F_COLLECTION}.update({firstName:\"Lowery\"},{\"$\"set:{\"nationality\":\"American\"}})"
```

Figura 34. Sentencias UPDATE para 100 documentos en MongoDB.

Para 1.000 registros le corresponden 100 sentencias UPDATE.

```
#############################################################################
# function: benchmark mongo-update
#############################################################################
function mongodb_update_benchmark ()
{
   typeset -r F_MONGOHOST="$1"
   typeset -r F_MONGOPORT="$2"
   typeset -r F_MONGODBNAME="$3"
   typeset -r F_MONGOUSER="$4"
   typeset -r F_MONGOPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_MONGOUPDATE1="db.${F_COLLECTION}.update({firstName:\"Jacobson\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE2="db.${F_COLLECTION}.update({firstName:\"Ana\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE3="db.${F_COLLECTION}.update({firstName:\"Bright\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE4="db.${F_COLLECTION}.update({firstName:\"Owen\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE5="db.${F_COLLECTION}.update({firstName:\"Blake\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE6="db.${F_COLLECTION}.update({firstName:\"Barlow\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE7="db.${F_COLLECTION}.update({firstName:\"Faye\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE8="db.${F_COLLECTION}.update({firstName:\"Buckley\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE9="db.${F_COLLECTION}.update({firstName:\"Fannie\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE10="db.${F_COLLECTION}.update({firstName:\"Lowery\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE11="db.${F_COLLECTION}.update({firstName:\"Jacobson\"},{\"$\"set:{\"civilStatus\":\"married\"}})"

   typeset -r F_MONGOUPDATE12="db.${F_COLLECTION}.update({firstName:\"Ana\"},{\"$\"set:{\"civilStatus\":\"single\"}})"

   typeset -r F_MONGOUPDATE13="db.${F_COLLECTION}.update({firstName:\"Bright\"},{\"$\"set:{\"civilStatus\":\"married\"}})"
```

Figura 35. Sentencias UPDATE para 1.000 documentos en MongoDB.

Para 100.000 registros le corresponden 300 sentencias UPDATE.


```
#############################################################################
# function: benchmark mongo-update
#############################################################################
function mongodb_update_benchmark ()
{
   typeset -r F_MONGOHOST="$1"
   typeset -r F_MONGOPORT="$2"
   typeset -r F_MONGODBNAME="$3"
   typeset -r F_MONGOUSER="$4"
   typeset -r F_MONGOPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_MONGOUPDATE1="db.${F_COLLECTION}.update({firstName:\"Jacobson\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE2="db.${F_COLLECTION}.update({firstName:\"Ana\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE3="db.${F_COLLECTION}.update({firstName:\"Bright\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE4="db.${F_COLLECTION}.update({firstName:\"Owen\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE5="db.${F_COLLECTION}.update({firstName:\"Blake\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE6="db.${F_COLLECTION}.update({firstName:\"Barlow\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE7="db.${F_COLLECTION}.update({firstName:\"Faye\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE8="db.${F_COLLECTION}.update({firstName:\"Buckley\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE9="db.${F_COLLECTION}.update({firstName:\"Fannie\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE10="db.${F_COLLECTION}.update({firstName:\"Lowery\"},{\"$\"set:{\"nationality\":\"American\"}})"

   typeset -r F_MONGOUPDATE11="db.${F_COLLECTION}.update({firstName:\"Jacobson\"},{\"$\"set:{\"civilStatus\":\"married\"}})"

   typeset -r F_MONGOUPDATE12="db.${F_COLLECTION}.update({firstName:\"Ana\"},{\"$\"set:{\"civilStatus\":\"single\"}})"

   typeset -r F_MONGOUPDATE13="db.${F_COLLECTION}.update({firstName:\"Bright\"},{\"$\"set:{\"civilStatus\":\"married\"}})"
```

Figura 36. Sentencias UPDATE para 100.000 documentos en MongoDB.

En la **function mongo_collection_size**, se calculó el tamaño de la colección.

```
#################################################################################
# function: mongdb collection size
#################################################################################
function mongo_collection_size ()
{
   typeset -r F_MONGOHOST="$1"
   typeset -r F_MONGOPORT="$2"
   typeset -r F_MONGODBNAME="$3"
   typeset -r F_MONGOUSER="$4"
   typeset -r F_MONGOPASSWORD="$5"
   typeset -r F_COLLECTION="$6"

   typeset -r F_COMMAND="printjson(db.getSiblingDB('${F_MONGODBNAME}').${F_COLLECTION}.stats().size)"

   process_log "calculating the size of mongo collection."
   output="$(run_mongo_command "${F_MONGOHOST}" "${F_MONGOPORT}"   \
                               "${F_MONGODBNAME}" "${F_MONGOUSER}" \
                               "${F_MONGOPASSWORD}" "${F_COMMAND}")"

   collectionsize="$(echo -n ${output})"

   echo "${collectionsize}"

}
```

Figura 37. Tamaño de la colección en MongoDB.

En la **function mongodb_create_index**, se crearon los índices para cada carga de trabajo respectivamente.

Para 100 registros le corresponden 14 índices.

```
typeset -r F_MONGODBIDX1="db.${F_COLLECTION}.ensureIndex( { \"firstName\": 1});"
typeset -r F_MONGODBIDX2="db.${F_COLLECTION}.ensureIndex( { \"lastName\": 1});"
typeset -r F_MONGODBIDX3="db.${F_COLLECTION}.ensureIndex( { \"passport.serial\": 1});"
typeset -r F_MONGODBIDX4="db.${F_COLLECTION}.ensureIndex( { \"car.plaque\": 1});"
typeset -r F_MONGODBIDX5="db.${F_COLLECTION}.ensureIndex( { \"email\": 1});"
typeset -r F_MONGODBIDX6="db.${F_COLLECTION}.ensureIndex( { \"mascots\": 1});"
typeset -r F_MONGODBIDX7="db.${F_COLLECTION}.ensureIndex( { \"passport\": 1});"
typeset -r F_MONGODBIDX8="db.${F_COLLECTION}.ensureIndex( { \"age\": 1});"
typeset -r F_MONGODBIDX9="db.${F_COLLECTION}.ensureIndex( { \"address.city\": 1});"
typeset -r F_MONGODBIDX10="db.${F_COLLECTION}.ensureIndex( { \"car.make\": 1});"
typeset -r F_MONGODBIDX11="db.${F_COLLECTION}.ensureIndex( { \"car.colour\": 1});"
typeset -r F_MONGODBIDX12="db.${F_COLLECTION}.ensureIndex( { \"mascots.petName\": 1});"
typeset -r F_MONGODBIDX13="db.${F_COLLECTION}.ensureIndex( { \"tradeMarks.brandName\": 1});"
typeset -r F_MONGODBIDX14="db.${F_COLLECTION}.ensureIndex( { \"bankAccounts.bankName\": 1});"
```

Figura 38. Índices en MongoDB para 100 registros.

Para 1.000 registros le corresponden 16 índices.

```
typeset -r F_MONGODBIDX1="db.${F_COLLECTION}.ensureIndex( { \"firstName\": 1});"
typeset -r F_MONGODBIDX2="db.${F_COLLECTION}.ensureIndex( { \"lastName\": 1});"
typeset -r F_MONGODBIDX3="db.${F_COLLECTION}.ensureIndex( { \"passport.serial\": 1});"
typeset -r F_MONGODBIDX4="db.${F_COLLECTION}.ensureIndex( { \"car.plaque\": 1});"
typeset -r F_MONGODBIDX5="db.${F_COLLECTION}.ensureIndex( { \"email\": 1});"
typeset -r F_MONGODBIDX6="db.${F_COLLECTION}.ensureIndex( { \"mascots\": 1});"
typeset -r F_MONGODBIDX7="db.${F_COLLECTION}.ensureIndex( { \"passport\": 1});"
typeset -r F_MONGODBIDX8="db.${F_COLLECTION}.ensureIndex( { \"age\": 1});"
typeset -r F_MONGODBIDX9="db.${F_COLLECTION}.ensureIndex( { \"address.city\": 1});"
typeset -r F_MONGODBIDX10="db.${F_COLLECTION}.ensureIndex( { \"car.colour\": 1});"
typeset -r F_MONGODBIDX11="db.${F_COLLECTION}.ensureIndex( { \"car.make\": 1});"
typeset -r F_MONGODBIDX12="db.${F_COLLECTION}.ensureIndex( { \"tradeMarks.brandName\": 1});"
typeset -r F_MONGODBIDX13="db.${F_COLLECTION}.ensureIndex( { \"mascots.petName\": 1});"
typeset -r F_MONGODBIDX14="db.${F_COLLECTION}.ensureIndex( { \"mobile\": 1});"
typeset -r F_MONGODBIDX15="db.${F_COLLECTION}.ensureIndex( { \"bankAccounts.bankName\": 1});"
typeset -r F_MONGODBIDX16="db.${F_COLLECTION}.ensureIndex( { \"passport.expiration\": 1});"
```

Figura 39. Índices en MongoDB para 1.000 registros.

Para 100.000 registros le corresponden 16 índices.

```
typeset -r F_MONGODBIDX1="db.${F_COLLECTION}.ensureIndex( { \"firstName\": 1});"
typeset -r F_MONGODBIDX2="db.${F_COLLECTION}.ensureIndex( { \"lastName\": 1});"
typeset -r F_MONGODBIDX3="db.${F_COLLECTION}.ensureIndex( { \"passport.serial\": 1});"
typeset -r F_MONGODBIDX4="db.${F_COLLECTION}.ensureIndex( { \"car.plaque\": 1});"
typeset -r F_MONGODBIDX5="db.${F_COLLECTION}.ensureIndex( { \"email\": 1});"
typeset -r F_MONGODBIDX6="db.${F_COLLECTION}.ensureIndex( { \"mascots\": 1});"
typeset -r F_MONGODBIDX7="db.${F_COLLECTION}.ensureIndex( { \"passport\": 1});"
typeset -r F_MONGODBIDX8="db.${F_COLLECTION}.ensureIndex( { \"age\": 1});"
typeset -r F_MONGODBIDX9="db.${F_COLLECTION}.ensureIndex( { \"address.city\": 1});"
typeset -r F_MONGODBIDX10="db.${F_COLLECTION}.ensureIndex( { \"car.make\": 1});"
typeset -r F_MONGODBIDX11="db.${F_COLLECTION}.ensureIndex( { \"car.colour\": 1});"
typeset -r F_MONGODBIDX12="db.${F_COLLECTION}.ensureIndex( { \"mascots.petName\": 1});"
typeset -r F_MONGODBIDX13="db.${F_COLLECTION}.ensureIndex( { \"tradeMarks.brandName\": 1});"
typeset -r F_MONGODBIDX14="db.${F_COLLECTION}.ensureIndex( { \"bankAccounts.bankName\": 1});"
typeset -r F_MONGODBIDX15="db.${F_COLLECTION}.ensureIndex( { \"passport.expiration\": 1});"
typeset -r F_MONGODBIDX16="db.${F_COLLECTION}.ensureIndex( { \"mobile\": 1});"
```

Figura 40. Índices en MongoDB para 100.000 registros.

Es importante señalar, que se deben levantar los servidores de la siguiente manera:

PostgreSQL

```
kenia@kenia-Dell-System-Inspiron-N4110:~$ sudo su
[sudo] contraseña para kenia:
root@kenia-Dell-System-Inspiron-N4110:/home/kenia# su postgres
postgres@kenia-Dell-System-Inspiron-N4110:/home/kenia$ psql
psql (10.10 (Ubuntu 10.10-0ubuntu0.18.04.1))
Type "help" for help.

postgres=#
```


Figura 41. Levantar servidor de PostgreSQL.

MongoDB

Se agrega en una consola lo siguiente:

```
kenia@kenia-Dell-System-Inspiron-N4110:~$ sudo mongod --dbpath=/var/lib/mongodb
```

Y en otra:

```
kenia@kenia-Dell-System-Inspiron-N4110:~$ mongo
```

Figura 42. Levantar servidor de MongoDB.

Finalmente, la herramienta se ejecuta de la siguiente forma:

```
kenia@kenia-Dell-System-Inspiron-N4110:~/pg_nosql_benchmark_cien$ ./pg_nosql_benchmark
```

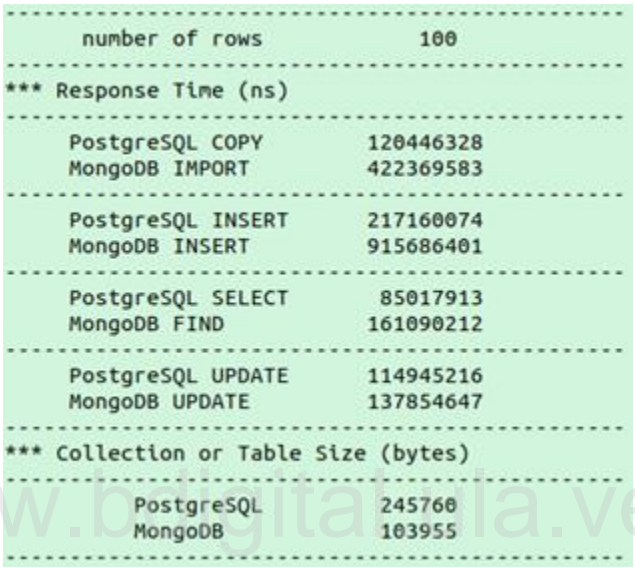
Figura 43. Ejecución de la herramienta.

Capítulo 5

5. Resultados

A continuación, se visualizan los reportes con los resultados obtenidos correspondientes a las tres cargas de trabajo respectivamente.

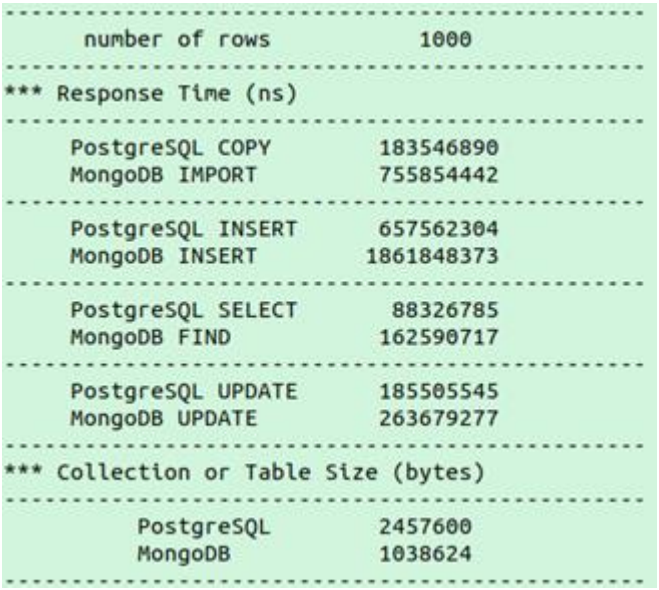
Para 100 registros.



number of rows	100
*** Response Time (ns)	
PostgreSQL COPY	120446328
MongoDB IMPORT	422369583
PostgreSQL INSERT	217160074
MongoDB INSERT	915686401
PostgreSQL SELECT	85017913
MongoDB FIND	161090212
PostgreSQL UPDATE	114945216
MongoDB UPDATE	137854647
*** Collection or Table Size (bytes)	
PostgreSQL	245760
MongoDB	103955

Figura 44. Reporte de los resultados con 100 registros.

Para 1.000 registros.



number of rows	1000
*** Response Time (ns)	
PostgreSQL COPY	183546890
MongoDB IMPORT	755854442
PostgreSQL INSERT	657562304
MongoDB INSERT	1861848373
PostgreSQL SELECT	88326785
MongoDB FIND	162590717
PostgreSQL UPDATE	185505545
MongoDB UPDATE	263679277
*** Collection or Table Size (bytes)	
PostgreSQL	2457600
MongoDB	1038624

Figura 45. Reporte de los resultados con 1.000 registros.

Para 100.000 registros.

```
-----------------------------------------------
     number of rows         100000
-----------------------------------------------
*** Response Time (ns)
-----------------------------------------------
     PostgreSQL COPY       6322754204
     MongoDB IMPORT         524843856
-----------------------------------------------
     PostgreSQL INSERT    41965708973
     MongoDB INSERT      172029763771
-----------------------------------------------
     PostgreSQL SELECT      619980738
     MongoDB FIND           136230483
-----------------------------------------------
     PostgreSQL UPDATE    52121683459
     MongoDB UPDATE         201459122
-----------------------------------------------
*** Collection or Table Size (bytes)
-----------------------------------------------
          PostgreSQL      245760000
          MongoDB         103940312
-----------------------------------------------
```

Figura 46. Reporte de los resultados con 100.000 registros.

Tabla general de los resultados

Tabla 3

Resultados de la evaluación comparativa de PostgreSQL y MongoDB.

	100	1.000	100.000	Unidades
PostgreSQL COPY	120446328	183546890	6322754204	nanosegundos
MongoDB IMPORT	422369583	755854442	524843856	
PostgreSQL INSERT	217160074	657562304	41965708973	
MongoDB INSERT	915686401	1861848373	172029763771	
PostgreSQL SELECT	85017913	88326785	619980738	
MongoDB FIND	161090212	162590717	136230483	
PostgreSQL UPDATE	114945216	185505545	52121683459	
MongoDB UPDATE	137854647	263679277	201459122	
PostgreSQL TABLE	245760	2457600	245760000	bytes
MongoDB COLLECTION	103955	1038624	103940312	

A continuación, se visualizan los resultados gráficamente

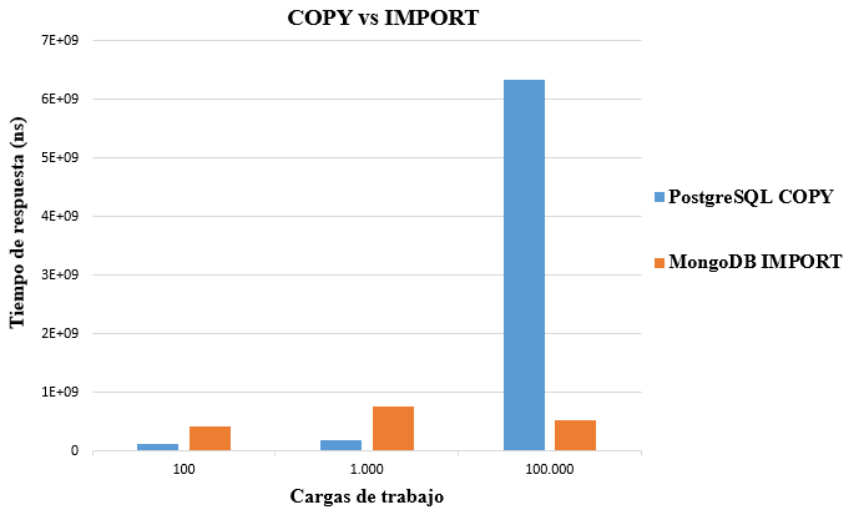


Figura 47. Representación gráfica del COPY vs IMPORT.

En la carga de 100 registros, se puede observar que el comando COPY tiene un mejor rendimiento, es decir, el tiempo de respuesta es menor que el comando IMPORT. Del mismo modo, ocurre con la carga de 1.000 registros. Sin embargo, en la carga de 100.000 registros el comando IMPORT tiene mejor rendimiento.

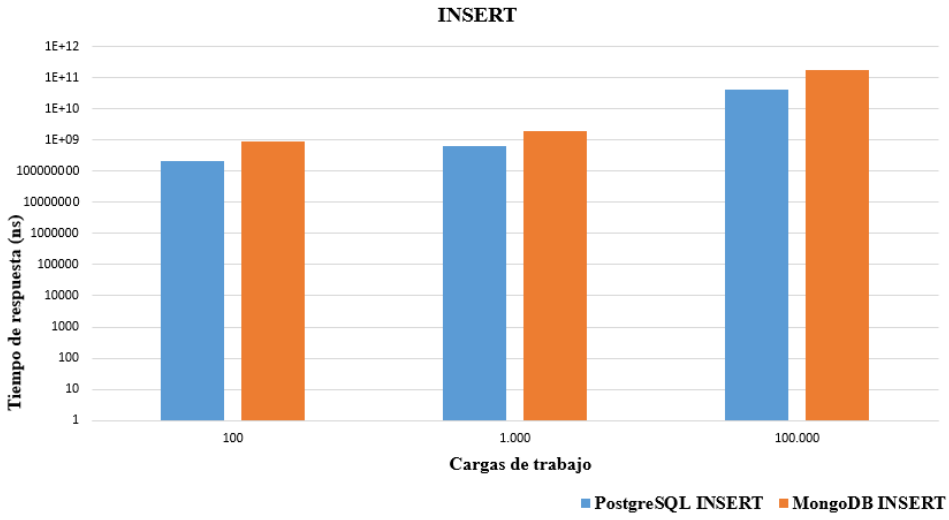


Figura 48. Representación gráfica del INSERT.

En la carga de 100 registros, se puede observar que el comando INSERT de PostgreSQL arroja un tiempo de respuesta menor que el comando INSERT de MongoDB. Así mismo, ocurre con la carga de 1.000 y 100.000 registros.

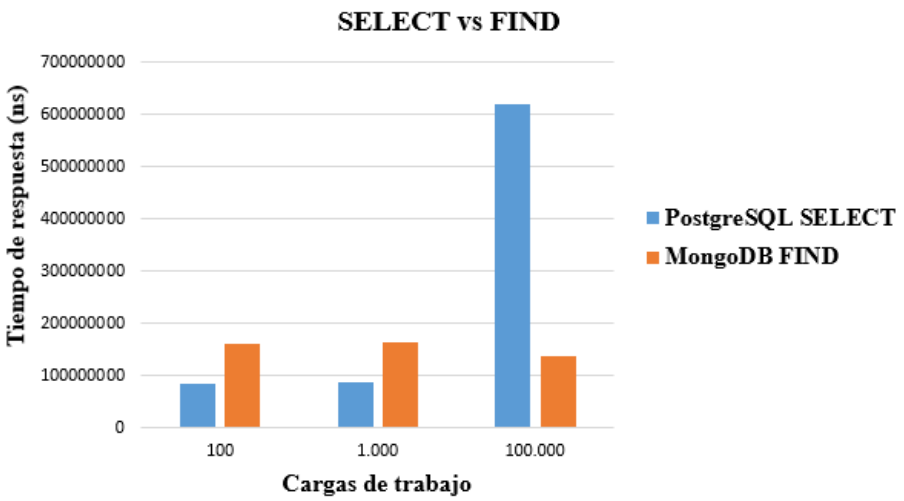


Figura 49. Representación gráfica del SELECT vs FIND.

En la carga de 100 registros, se puede observar que las consultas SELECT arrojan un tiempo de respuesta menor que las consultas FIND. Del mismo modo, ocurre con la carga de 1.000 registros. No obstante, en la carga de 100.000 registros las consultas FIND tienen mejor rendimiento.

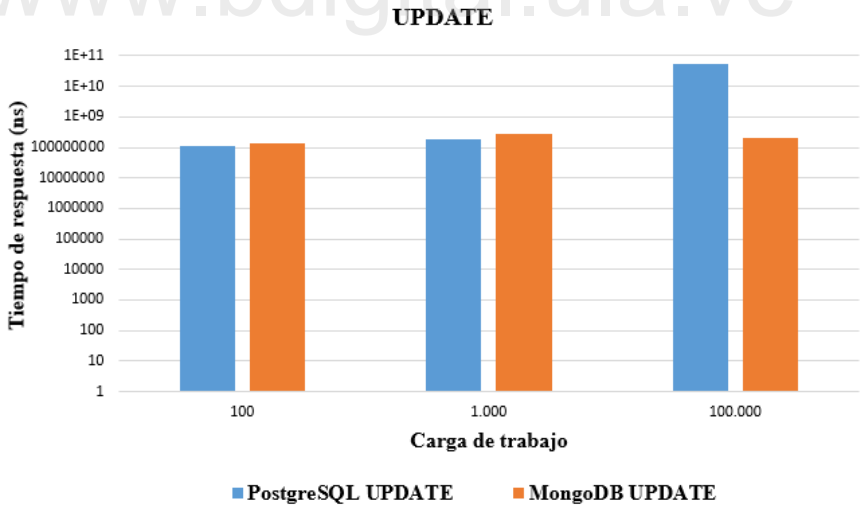


Figura 50. Representación gráfica del UPDATE.

En la carga de 100 registros, se puede observar que las sentencias UPDATE de PostgreSQL arrojan un tiempo de respuesta menor que el UPDATE de MongoDB. Así mismo, ocurre con la carga de 1.000 registros. Sin embargo, en la carga de 100.000 registros las sentencias UPDATE de MongoDB tienen mejor rendimiento.

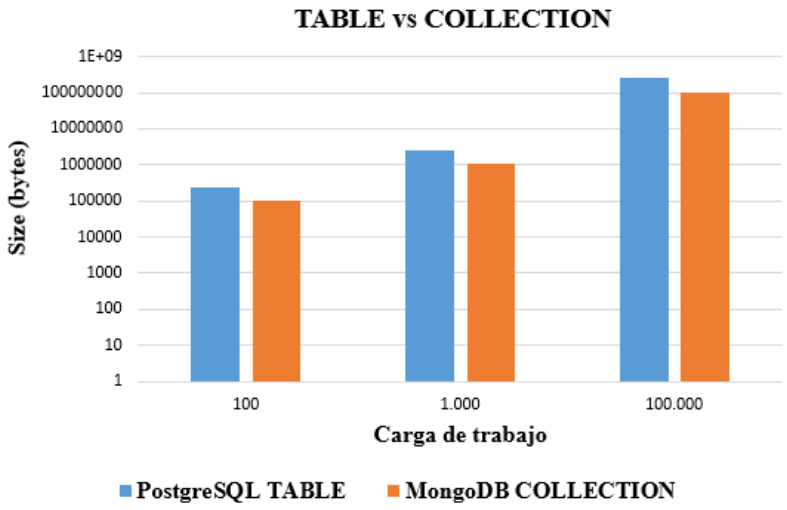


Figura 51. Representación gráfica del tamaño de la tabla y la colección.

En la carga de 100, 1.000 y 100.000 registros, se puede observar que hay mejor rendimiento cuando se almacena en una colección que en una tabla.

De modo general, se puede concluir que el manejador de base de datos PostgreSQL arroja tiempos de respuesta óptimos cuando manipula poca cantidad de datos, pero ocupa más espacio de almacenamiento que MongoDB. Por el contrario, MongoDB arroja tiempos de respuesta aceptables cuando maneja gran cantidad de datos, así como también, ocupa menos espacio de almacenamiento que PostgreSQL.

6. Conclusiones

En esta época donde se generan grandes cantidades de datos semiestructurados, las bases de datos relacionales empiezan a mostrar deficiencias, en almacenamiento u operaciones; siendo esta una de las principales razones de impulsar el uso de las NoSQL.

Es por ello, que se llega a la siguiente interrogante ¿cúando se debe utilizar una base de datos NoSQL y cuándo una relacional? A continuación, se señalan las características más importantes para resolver dicha interrogante. Cuando los datos deben ser consistentes sin dar posibilidad al error, SQL. Cuando se tiene poco presupuesto para máquinas de alto rendimiento, NoSQL. Cuando las estructuras de datos que se manejan son variables, NoSQL. Cuando existe el análisis de grandes cantidades de datos en modo lectura, NoSQL.

Es importante señalar, que las características NoSQL incorporadas en las últimas versiones de PostgreSQL satisfacen las mismas necesidades que MongoDB. Todo esto encaminado a agilizar y flexibilizar la manipulación de los datos. En este trabajo de grado, se obtuvo el mejor rendimiento de los manejadores en cuanto a tiempo de respuesta y tamaño de la tabla y la colección.

En PostgreSQL, se obtuvo mejores tiempo de respuesta cuando maneja poca cantidad de datos y en MongoDB, cuando maneja gran cantidad. Por otra parte, el tamaño de la colección de MongoDB ocupa menos espacio que la tabla de PostgreSQL, ya sea manipulando pocos o muchos datos.

En conclusión, las bases de datos no relacionales o NoSQL no reemplazan a las relacionales, sino que las complementan cuando éstas se quedan pequeñas o poco prácticas para el manejo y almacenamiento de grandes cantidades de información.

Bibliografía

Bases de datos NoSQL. (s.f.). Recuperado 10 de septiembre de 2018 de https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

BSON. (s.f.). En *Wikipedia*. Recuperado el 10 de septiembre de 2018 de https://es.wikipedia.org/wiki/BSON

Camacho, E. (s.f.). NoSQL la evolución de las bases de datos. Recuperado 10 de septiembre de 2018 de https://sg.com.mx/revista/28/nosql-evolucion-bases-datos

JSON Fnctions and Operators. (s.f.). Recuperado el 10 de septiembre de 2018 de https://www.postgresql.org/docs/current/functions-json.html

Características de MongoDB. (s.f.). Recuperado el 10 de septiembre de 2018 de http://www.manualweb.net/mongodb/que-es-mongodb/

Del Alba, L. (20 de marzo de 2017). Operaciones más rápidas con el tipo de datos JSONB en PostgreSQL. Recuperado 10 de septiembre de 2018 de https://www.compose.com/articles/faster-operations-with-the-jsonb-data-type-in-postgresql/

Fotache M & Cogean D. (2013). NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL. *Informatica Economica, 17*(2), 41-58. doi: 10.12948/issn14531305/17.2.2013.04

Graterol, Y. (s.f.). Mongo DB en Español. Recuperado el 10 de septiembre de 2018 de https://tutorialesenpdf.com/mongodb/

Hanlon et al. (2015). A Case Study for NoSQL Applications and Performance Benefits: CouchDB vs. Postgres. *Figshare,* 1-6.

Introduction to MongoDB. (s.f.). Recuperado 10 de septiembre de 2018 de https: https://docs.mongodb.com/manual/introduction/

Kaur K & Rani R. (2013). Modeling and Querying Data in NoSQL Databases. International Conference on Big Data, pp. 1-7. IEEE, 2013.

Kumar, V. (17 de julio de 2014). Pg_nosql_benchmark. Recuperado 18 de noviembre de 2018 de https://github.com/EnterpriseDB/pg_nosql_benchmark

Modelado One-to-Many. (s.f.). Recuperado el 10 de septiembre de 2018 de http://www.manualweb.net/mongodb/modelado-one-to-many-mongodb/

Modelado One-to-One. (s.f.). Recuperado el 10 de septiembre de 2018 de http://www.manualweb.net/mongodb/modelado-one-to-one-mongodb/

Moreno, G. (31 de enero de 2018). Medium. [Blog]. Recuperado de https://medium.com/@Gildder/teorema-cap-e99d66fde6a0

NoSQL. (s.f.). En *Wikipedia*. Recuperado el 10 de septiembre de 2018 de https://es.wikipedia.org/wiki/NoSQL

PostgreSQL. (s.f.). En *Wikipedia*. Recuperado el 10 de septiembre de 2018 de https://es.wikipedia.org/wiki/PostgreSQL

PostgreSQL. (7 de febrero de 2019). Recuperado el 29 de marzo de2019 de https://hostingpedia.net/postgresql.html

Query and Projection Operators. (s.f.). Recuperado 10 de septiembre de 2018 de https://docs.mongodb.com/manual/reference/operator/query/

Sim S., Easterbrook S & Holt R. (2003). Using Benchmarking to Advance Research: A Challenge to Software Engineering. In Proceedings of the 25th International Conference on Software Engineering, pp. 74–83. IEEE, 2003.

Suárez, J. (18 de noviembre de 2015). Pandorafms. [Blog]. Recuperado de https://blog.pandorafms.org/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una