

Diplomatura en Bases de Datos

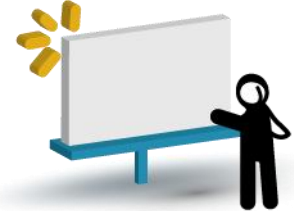
Módulo 3: Herramientas de cara al futuro

Unidad 4: Bases de datos orientadas a Big Data y distribuidas

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



Presentación:

En esta Unidad descubrimos cómo y cuáles son las bases de datos orientadas a Big Data como así también las bases de datos distribuidas. Revisamos, también, sus principales conceptos y arquitecturas. Finalmente, recorreremos una base de datos orientadas a Big Data.



Objetivos:

Que los participantes:

- Entiendan cuáles son los requerimientos que la era de Big Data y Big Analytics imponen a las bases de datos.
- Identifiquen las diferentes opciones que se tienen para satisfacer las necesidades de Big Data.
- Conozcan las principales herramientas disponibles en el mercado.

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



Bloques temáticos:

1. Conceptos de Big Data:

- ¿Qué es el análisis de Big Data?
- Almacenamiento y administración de Big Data.

2. Bases de datos distribuidas:

- Almacenamiento en una base de datos distribuida.
- Gestión distribuida de catálogos.
- Nada compartido y disco compartido.
- Ejemplos de bases de datos distribuidas no relacionales.

3. MongoDB



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

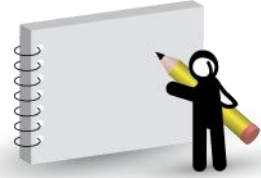
Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning



Tomen nota;

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.



1. Conceptos de Big Data

Sin duda, Big Data es un tema de moda utilizado por muchas personas en diferentes contextos y sin semántica precisa. Además existen muchas (pseudo) definiciones diferentes de este concepto. Por lo general, se habla de Big Data cuando el tamaño del conjunto de datos está más allá de la capacidad de las herramientas de base de datos actuales para recopilar, procesar, recuperar, administrar, y analizar el conjunto de datos.

Así, Big Data es un término amplio con definiciones múltiples y competitivas, pero que sugiere dos cambios significativos complementarios en el papel de los datos dentro de la informática y la sociedad:

- Más datos: la capacidad de almacenar y procesar todos los datos generados (multimedia, de redes sociales y transaccionales) es mayor; como así también es mayor la capacidad de mantener esta información potencialmente a perpetuidad.
- Mayor efecto: avances en aprendizaje automático, análisis predictivo, etc, permiten que se genere más valor a partir de los datos.

Big Data suele caracterizarse por varias V que también plantean problemas para su almacenamiento y procesamiento:

- Volumen: Big data implica un volumen enorme de datos. En un inicio los datos eran creados por los propios empleados pero ahora que los datos son generados automáticamente por máquinas, redes e interacciones personales en sistemas como redes sociales los volúmenes a analizar son masivos. La tecnología para guardar y procesar ha avanzado paralelamente por lo que el mayor problema ahora no es tanto el tamaño como otras dimensiones como la veracidad.
- Velocidad: La velocidad en Big data se refiere al ritmo en que los datos de entrada fluyen desde las diversas fuentes como procesos de negocio, máquinas y sensores, redes sociales, dispositivos móviles, etc. El flujo de datos es masivo y continuo. Estos datos recogidos en tiempo real permiten ayudar a investigadores y organizaciones a la hora de tomar decisiones aportando valiosa información que



suponen ventajas competitivas estratégicas. El ROI (retorno en la inversión) está asegurado para las empresas que sepan manejar esa velocidad. Muchas empresas comienzan con muestras de datos para ir comprendiendo el valor aportado y van ampliando a medida que se van viendo los resultados.

- Variedad: La variedad se refiere a las diferentes fuentes y tipos de datos tanto estructurados como no estructurados. Hace pocos años los únicos datos que se almacenaban eran de fuentes como hojas de cálculo y bases de datos. Ahora, los datos llegan en la forma de emails, fotos, videos, sistemas de monitorización, PDFs, ficheros de sonido, etc. Esta variedad en datos no estructurados crea problemas de almacenamiento, minería de datos y análisis de la información.
- Veracidad: La veracidad en el big Data se refiere al sesgo, el ruido y la alteración de datos. Los responsables del proyecto big data han de preguntarse honestamente si los datos que se almacenan y extraen son directamente relacionados y significativos al problema que se trata de analizar. Esta característica puede ser el mayor reto cuando se comparan con otras como el volumen o la velocidad. Cuando se valore el alcance en su estrategia de big data es necesario contar en el equipo con socios imparciales que ayuden a mantener los datos limpios y asegurarse que los procesos no acumulen “datos sucios” en sus sistemas.

Por último, hay quien añade una V más a las dimensiones, la volatilidad. Se refiere al tiempo durante el cual los datos recogidos son válidos y el tiempo que deben permanecer almacenados. En esta nueva era de aluvión de datos en tiempo real es necesario determinar hasta qué punto los datos son válidos y en qué momento dejan de ser relevantes para su estudio analítico.



¿Qué es el análisis de Big Data?

Big Analytics es el proceso de examinar grandes cantidades de diferentes tipos de datos, o Big Data, para descubrir patrones ocultos, desconocidos correlaciones y otra información útil.

La diferencia entre una búsqueda web y una búsqueda de un patrón es que el término buscado es conocido en el primer caso mientras que en el segundo no se sabe como será el patrón que emerja como resultado.

La caracterización de Big Data más bien simplificada y vaga no considera ocurrencia de complejidad explícita, por ejemplo, en conjuntos de datos de gráficos y entornos heterogéneos utilizado para Big Analytics.

La complejidad junto con el gran volumen en su mayoría requiere una escalabilidad de sistemas informáticos y algoritmos utilizados. De hecho, solo algunos algoritmos conocidos utilizados para analizar datos en un DW pueden no escalar y algunas técnicas de su paralelización y distribución son necesarios.

Big Analytics no solo requiere una nueva arquitectura de base de datos, sino también nuevos enfoques a los métodos para el análisis de datos. El último significa la reformulación de los viejos métodos de minería de datos, o su nueva implementación, o incluso un desarrollo de métodos completamente nuevos.

En término del planteo de bases de datos se pueden distinguir tres áreas relacionadas con la gestión de datos y el procesamiento de Big Data:

- almacenamiento y procesamiento de archivos de bajo nivel con funciones de base de datos simples,
- procesamiento de base de datos más sofisticado con lenguajes de consulta de alto nivel,



- Grandes transacciones que necesitan una extensión de los métodos utilizados generalmente en la tecnología de almacenes de datos (DW).

Las tecnologías Big Data incluyen aquellas que permiten obtener más significado del aprendizaje de máquina de datos, por ejemplo, y aquellos que permiten almacenar mayores volúmenes de datos con una granularidad mayor que nunca.

Google fue pionero en muchas de estas tecnologías de Big Data, y ampliaron su camino en la comunidad IT mediante Hadoop. Tanto la aparición de Hadoop como el desarrollo de otras tecnologías para el almacenamiento de datos no estructurados le dieron un impulso sin igual a Big Data.

En este sentido, cualquier sistema eficiente de procesamiento de datos requiere no solo algoritmos efectivos sino también herramientas para almacenar y procesar grandes conjuntos de datos. Entre estas herramientas se encuentran (entre las que se incluye Hadoop):

- sistemas tradicionales de bases de datos relacionales paralelas,
- sistemas de archivos distribuidos y tecnologías Hadoop,
- bases de datos NoSQL,
- nuevas arquitecturas de bases de datos (por ejemplo, Big Data Management Systems, bases de datos NewSQL, Bases de datos NoSQL con transacciones ACID).



Almacenamiento y administración de Big Data

Para el almacenamiento y procesamiento de Big Data se prefieren dos características para el acceso a los masivos volúmenes de información:

- escalabilidad
- alta velocidad

Para el almacenamiento y administración se pueden utilizar las opciones presentadas en el punto anterior. En lo que sigue se presentamos estas opciones en términos de su idoneidad para Big Analytics.

DBMS relacionales

El DBMS relacional tradicional tanto centralizado como distribuido se basa en el uso del lenguaje SQL y propiedades transaccionales que garantizan propiedades ACID. ACID significa atomicidad, consistencia, aislamiento y durabilidad y es fundamental para las transacciones en una base de datos. Una parte significativa de la tecnología de base de datos relacional utilizable para Big Data se llama Bases de Datos Analíticos Masivamente Paralelos (MPAD).

A diferencia de los DWs, estos DBMS son capaces de proceder rápidamente a grandes cantidades de estructuras principalmente datos estructurados con modelado de datos mínimo requerido y puede escalar para acomodar múltiples terabytes y, a veces, petabytes de datos.

Las capacidades de consulta interactiva son posibles en MPAD. Las posibilidades de obtener resultados casi en tiempo real para consultas SQL complejas también están disponibles.



Sistemas de archivos distribuidos y tecnologías Hadoop

En este caso, los sistemas de archivos distribuidos considerados aquí se distinguen de los archivos de los sistemas de redes tradicionales (por ejemplo, en UNIX). Usan, por ejemplo, particiones y replicaciones de archivos. El más famoso de estos sistemas es Hadoop Distributed File System (HDFS). Hadoop es la parte principal de una arquitectura de software conocida llamada Hadoop Stack, y presenta un sistema de procesamiento por lotes basado en el marco MapReduce y HDFS.

En el aspecto analítico, MapReduce (MR) surgió como la plataforma para todas las necesidades de análisis de la empresa. Debido a que los clústeres Hadoop pueden escalar a petabytes e, incluso, exabytes de datos, las empresas ya no deben depender de conjuntos de datos de muestra, sino que pueden procesar y analizar todos los datos relevantes. Una primera solución SQL-on-Hadoop fue dada por Hive2 que proporcionaba una interfaz tipo SQL con MapReduce subyacente.

Hoy en día ya se cuenta con HiveQL que es un lenguaje similar a SQL originado a partir de Hive2. Por otro lado, MapReduce sigue siendo una técnica muy simple en comparación con aquellos utilizado en el área de bases de datos distribuidas. Los usuarios requieren aplicaciones más complejas y multi-etapas (por ejemplo, algoritmos de gráficos iterativos y aprendizaje automático) y consultas ad-hoc más interactivas.

Entonces se necesita compartir datos más rápidamente entre trabajos paralelos. Recientemente, hay otras herramientas de software que ofrecen muchos métodos de aprendizaje automático, por ejemplo, el motor Spark3 que es computación paralela distribuida en memoria. Spark está dirigido a algoritmos iterativos e interactivos, donde Hadoop no funciona bien.

Muchas bases de datos NoSQL se crean sobre el núcleo de Hadoop, es decir, su rendimiento depende de trabajos de MapReduce. Por ejemplo, Big Analytics requiere a menudo una iteración que apenas se logra en NoSQL basado en MapReduce. En consecuencia, algunas modificaciones como HaLoop framework brindan soporte para los procesos iterativos.



Bases de datos NoSQL

Las bases de datos NoSQL son un tipo relativamente nuevo de bases de datos que fueron iniciadas por las empresas Web a principios de la década de 2000. Aunque algunas listas populares de ellas incluyen mayoría todos modelos no relacionales de almacenamiento de datos como bases de datos XML, varias DW de valor-clave y bases de datos de gráficos representan también esta categoría.

Una de sus características típicas es que el diseño de la base de datos está impulsado por consultas, sin soporte de restricciones de integridad, sin lenguaje de consulta estándar, especialmente semántica de ACID que es relevante en el contexto del procesamiento de Big Data. NoSQL proporciona poca o ninguna compatibilidad con OLTP, que se requiere para la mayoría de las aplicaciones empresariales.

El teorema CAP (Conjetura de Brewer) ha demostrado que un sistema informático distribuido solo puede satisfacer como máximo dos de tres propiedades: consistencia (C, consistency), disponibilidad (A, availability) y tolerancia a Particiones (P, partition tolerance). Luego, considerando P en una red, las bases de datos NoSQL admiten A o C. En práctica, en su mayoría se prefiere A y la consistencia estricta se relajó principalmente a los llamados consistencia eventual.

La consistencia eventual solo garantiza eso, dado un suficiente período de tiempo durante el cual no hay escrituras, todas las escrituras previas se propagarán a través del sistema para que todas las copias replicadas de los datos sean consistentes.

La consistencia ha sido ampliamente adoptada, convirtiéndose en algo así como un valor predeterminado para una bases de datos NoSQL. Sin embargo, hay algunos ejemplos, donde la consistencia es ajustable (por ejemplo, en Cassandra) o configurable (por ejemplo, CP o AP en la base de datos Oracle NoSQL).

Sistemas de gestión de Big Data

Algunas de las bases de datos NoSQL son parte de arquitecturas de software más complejas, por ejemplo el caso de Hadoop stack, o incluso los llamados Big Data Management Systems (BDMS).



A diferencia de las bases de datos relacionales, donde el usuario ve solo SQL en la capa más externa de DBMS para manipular datos, estos sistemas permiten acceder a los datos a través de diversos medios en diferentes capas. A menudo, Hadoop stack se presenta como la primera generación de BDMS.

Por ejemplo, el sistema ASTERIX (Big Data management) utiliza diferentes tecnologías que a las de Hadoop Stack: un Hyracks (software de Partitionado-paralelo) especial para la plataforma de datos, un nivel algebraico Algebricks, HiveQL, pero también una compatibilidad con Hadoop MapReduce.

Normalmente, los BDMS usan muchas otras aplicaciones especiales de lenguajes de alto nivel. Oracle entiende a BDMS como una arquitectura que incorpora sin problemas Hadoop, NoSQL y DW relacional. Los BDMS a menudo están equipados con métodos avanzados para el análisis de datos.

Por ejemplo, Myria, un BDMS distribuido y de tipo shared-nothing Big Data management y un sistema de nube, es apropiado para el análisis estadístico real. Actualmente, los BDMS están evolucionando más bien para eliminar por completo la capa MapReduce.

Bases de datos NewSQL

En los últimos años, el desarrollo de la administración de datos muestra que hay aplicaciones que requieren una fuerte consistencia. NewSQL pertenece a esta categoría la cual mantiene las propiedades del lenguaje SQL y ACID. Sin embargo, prevalecen los problemas de rendimiento y escalabilidad planteado por los RDBM OLTP tradicionales.

Estos DBMS logran un alto rendimiento y escalabilidad ofrecen rediseños arquitectónicos que aprovechan mejor las plataformas de hardware modernas tales como clústeres de computadoras de muchos núcleos con almacenamiento en memoria grande o no volátiles. Esto les permiten realizar análisis en tiempo real y procesamiento de transacciones al mismo tiempo.



NoSQL con transacciones ACID

También muchos diseñadores de NoSQL están explorando un retorno a las transacciones con propiedades ACID como el medio preferido para administrar la concurrencia para una amplia gama de aplicaciones.

Algunas observaciones muestran que CAP se refiere solo a una parte del espacio de diseño para distribución y escalabilidad. Cuando las particiones de red son raras, no es necesario decidir firmemente entre C y A, porque es habitual adoptar una consistencia eventual.

Cloud Spanner de Google pertenece a esta categoría, pero no es un sistema relacional puro implementado sobre un DW de tipo clave-valor. FoundationDB Store es otro ejemplo de DW de tipo valor-clave con características de base de datos distribuida con transacciones, escalabilidad y tolerancia verdadera a fallas ACID.

Tales almacenes de datos ofrecen también la capa SQL. Ambos ejemplos también muestran que la tecnología de almacenamiento de datos está desacoplada de su modelo de datos.



2. Bases de datos distribuidas

Un sistema de base de datos distribuida se llama al sistema que cuenta con varios sitios de bases de datos vinculado por un sistema de comunicaciones de tal manera que los datos en cualquier sitio es disponible para usuarios en otros sitios.

De esta forma, cada sitio o nodo tiene un sistema completo de procesamiento de información, con su propia función de administración de datos, personal, usuarios, hardware y software, incluida una base de datos local, sistema de gestión de bases de datos y software de comunicaciones.

Así mismo, un sitio debe tener memoria y un procesador de comunicaciones. Los sitios pueden estar en locaciones remotas unos de otros, pero conectadas por un sistema de telecomunicaciones.

También es posible tener un sistema distribuido en un área local vinculado por una red dentro de un solo edificio o área pequeña. Idealmente, los usuarios no necesitan estar al tanto de la verdadera ubicación de los datos a los que acceden, y el sistema parece ser una base de datos local para ellos.

Este tipo de sistemas distribuidos, dependiendo de las necesidades de la organización, proporcionan acceso remoto a los usuarios, incluida una mayor autonomía local, confiabilidad mejorada, mejor disponibilidad de datos, mayor rendimiento, menor tiempo de respuesta y menor costo de comunicaciones. Estas características representan ventajas sobre un solo sistema centralizado.

Entre los sistemas distribuidos se encuentran sistemas relacionales tradicionales basadas en SQL, bases de datos de objetos, bases de datos basadas en XM, sistemas de datos no estructurados, NoSQL y bases de datos NewSQL.

Si se comparan con un sistema en paralelo se encuentran las siguientes diferencias:

Bases de datos paralelas

- Las máquinas están físicamente cerca unas de otras, por ejemplo, la misma sala de servidores.

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



- Las máquinas se conectan con LAN e interruptores dedicados de alta velocidad.
- Se supone que el costo de comunicación es pequeño.
- Pueden tener la memoria compartida, el disco compartido o una arquitectura de nada compartido (share-nothing).

Bases de datos distribuidas:

- Las máquinas pueden estar muy alejadas entre sí, por ejemplo, en diferentes continentes
- Se puede conectar usando una red pública, por ejemplo, Internet
- Costos de comunicación y problemas no pueden ser ignorados
- Generalmente usa la arquitectura de nada compartido (share-nothing)

Una de las características que pueden tener las bases de datos distribuidas es que pueden ser de dos tipos según el modelo de administración que tengan: homogéneas o heterogéneas.

Homogéneo:

Cada sitio ejecuta el mismo tipo de DBMS

Heterogéneo:

Diferentes sitios ejecutan diferentes DBMS y pueden tener diferentes RDBMS que pueden llegar a ser, incluso, DBMS no relacionales.

En particular las bases de datos distribuidas heterogéneas presentan un acceso a los servidores de dichas bases de datos a través de una buena aceptación y protocolos estándar de Gateway.



Esto permite enmascarar las diferencias de los DBMS (capacidad, formato de datos, etc.), por ejemplo en el caso de tener ODBC y JDBC. Aunque, puede ser costoso y no ser capaz de ocultar todas las diferencias.

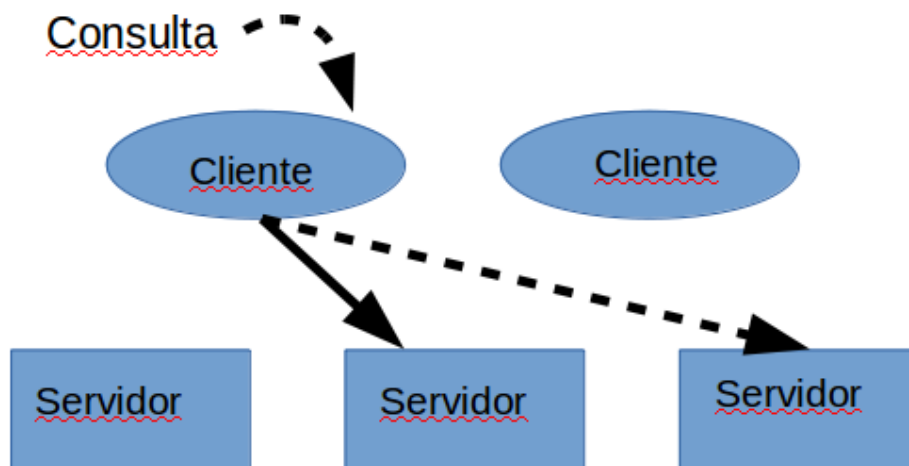
Esto ocurre, por ejemplo, cuando un servidor no es capaz de soportar la gestión de transacciones distribuidas.

Arquitecturas de bases de datos distribuidas

Las posibles arquitecturas para las DBMS tienen tres enfoques alternativos:

Cliente-Servidor

En este tipo de arquitectura se tiene uno o más clientes (por ejemplo, una computadora personal) y uno o más procesos de servidor (por ejemplo, un mainframe) como se muestra en la siguiente figura.





Las características que presenta son:

- Un proceso de cliente puede enviar una consulta a cualquier proceso de servidor
- Los clientes son responsables de las interfaces de usuario
- El servidor gestiona datos y ejecuta consultas

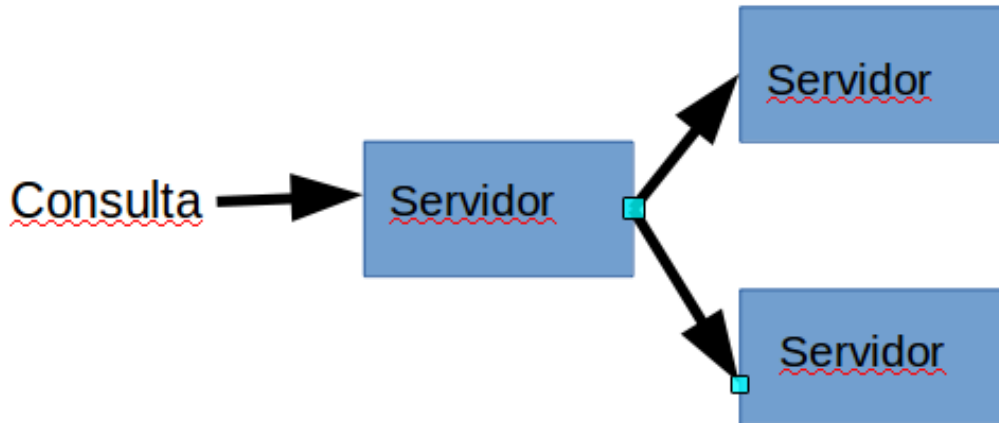
Entre las ventajas que se pueden mencionar de este tipo de arquitectura están:

- separación limpia y servidor centralizado
- las costosas interacciones del usuario no subutilizan las costosas máquinas servidor
- los usuarios pueden ejecutar GUI en clientes con los que están familiarizados

Sin embargo, también presenta ciertos desafíos importante en cuanto a la necesidad de manejar cuidadosamente los costos de comunicación.

Servidor colaborador

El servidor colaborador es conocido en inglés como "collaborating server". En este tipo de arquitecturas las consultas pueden abarcar múltiples sitios como se muestra en la siguiente figura:



En este caso es importante destacar que las consultas no están permitidas en los servidores del cliente ya que los clientes tendrían que interrumpir las consultas y combinar los resultados.

Además cuando un servidor recibe una consulta que requiere acceso a datos en otros servidores se generan subconsultas apropiadas y luego se junta el resultado. Esto elimina la distinción entre cliente y servidor.

Middleware

Este tipo de arquitecturas de los sistemas permite que una sola consulta abarque varios servidores. Sin embargo, no requiere que todos los servidores de bases de datos sean capaces de manejando estrategias de ejecución multi-sitio.

Esto implica que solo necesita un servidor de bases de datos capaz de gestionar consultas y transacciones que abarcan varios servidores (llamado middleware).

La capa de middleware es capaz de ejecutar uniones y otras operaciones en datos obtenidos de otros servidores, pero típicamente no mantiene ningún dato, mientras que los servidores restantes solo pueden manejar las consultas locales y transacciones.

En general, este tipo de sistema es útil al tratar de integrar varios "sistemas heredados" cuyas capacidades básicas no se pueden extender.



Almacenamiento en una base de datos distribuida

Cuando se tiene una base de datos distribuida las relaciones se almacenan en varios sitios. El problema que se presenta en estos casos es que el acceso a los datos en un sitio remoto incurre en el costos por el pasaje de mensajes.

En este caso, en donde se debe reducir la sobrecarga de mensajes es necesario una sola relación sea particionada o fragmentada en varios sitios.

Esto se realiza típicamente en los sitios donde se accede con más frecuencia. Sin embargo los datos se pueden replicar también cuando la relación presente una alta demanda.

A continuación se explican estos dos conceptos cruciales en el entendimiento de las bases de datos distribuidas.

Fragmentación

Este proceso se denomina a romper una relación en relaciones o fragmentos más pequeños para almacenarlos en diferentes sitios según sea necesario. La partición de relaciones se puede realizar de dos formas:

- **Horizontal:**
Por lo general, se hace en el caso de conjuntos disjuntos, como podría ser el caso de querer identificar mediante una consulta de selección (empleados en una ciudad - localidad de referencia). Aunque para recuperar la relación completa se deberá realizar una unión.
- **Vertical:**
En este caso se identifica por consultas de proyección donde los identificadores de tuplas (un registro o fila) son típicamente únicos agregados a cada tupla. Luego estos identificadores son replicados en cada fragmento.



Replicación

La replicación se da cuando se almacena varias copias de una relación o fragmento de relación en uno o más sitios. Por ejemplo, R se fragmenta en R1, R2, R3; y se coloca una copia de R2, R3; pero dos copias en R1 en dos sitios diferentes. Esto tiene como ventajas que da una mayor disponibilidad, por ejemplo cuando un enlace de sitio o comunicación se cae, como así también brinda la posibilidad de evaluación de consultas más rápida, por ejemplo, usando una copia local.

Este tipo de proceso puede ser sincrónico y asíncrono (posterior). Lo cual varía en cómo son de diferentes copias actuales cuando se modifica una relación.

Gestión distribuida de catálogos

Cuando se habla de gestión distribuida de catálogos se hace énfasis en cómo debe realizarse el seguimiento de cómo los datos se fragmentan y se replican en los sitios.

Esto implica además del esquema habitual, la autorización y la información estadística de estos procesos. Así, el sistema debe ser capaz de identificar de manera única cada réplica de cada fragmento.

Esto se logra a través de una identificación mediante un nombre de la relación. Pero como un nombre global único puede comprometer la autonomía de los servidores, en general para preservar esta autonomía local se utilizar como nombre de la relación global la siguiente forma:

Así para identificar una réplica se debe agregar un campo de id de réplica (ahora llamada réplica global).

De esta manera se debe tener un catálogo del sitio que describe todos los objetos (fragmentos, réplicas) en un sitio.

Además este sistema de gestión realiza un seguimiento de las réplicas de las relaciones creadas en este sitio y así para encontrar una relación, se debe buscar el catálogo de su sitio de nacimiento que nunca cambia, incluso si la relación se mueve.



Nada compartido y disco compartido

El patrón de replicación para distribuir las cargas de trabajo de la base de datos funciona bien para distribuir la actividad de lectura a través de servidores múltiples, pero no distribuye cargas de escritura transaccionales, que todavía deben ser dirigidas exclusivamente al servidor principal.

La replicación también tiene un valor limitado para la distribución de cargas de trabajo de almacenamiento de datos. Una carga de trabajo OLTP típicamente consiste en un gran número de solicitudes de corta duración.

Sin embargo, en un entorno de almacenamiento de datos, la carga de trabajo generalmente consiste en un número menor de consultas de uso intensivo de datos.

En servidores de bases de datos de alta gama, estas consultas masivas se ejecutan mediante múltiples procesos o subprocesos, cada uno de los cuales puede aprovechar un espacio separado y aprovecha múltiples canales de IO.

De esta manera, un servidor de base de datos se puede clasificar como:

- **Shared-everything:** en este caso, cada proceso de base de datos comparte la misma memoria, CPU y recursos de disco. Compartir la memoria implica que cada proceso está en el mismo servidor y, por lo tanto, esta arquitectura es una arquitectura de base de datos de un solo nodo.
- **Disco compartido:** en este caso, los procesos de la base de datos pueden existir en nodos separados en el clúster y tener acceso a la CPU y la memoria del servidor en el que residen. Sin embargo, cada proceso tiene el mismo acceso a los dispositivos de disco, que se comparten en todos los nodos del clúster.
- **No compartido (shared-nothing):** en este caso, cada nodo del clúster tiene acceso no solo a su propia memoria y CPU, sino también a dispositivos de disco dedicados y su propio subconjunto de la base de datos.



Ejemplos de bases de datos distribuidas no relacionales

El mantenimiento de la integridad transaccional de ACID en múltiples nodos en una base de datos relacional distribuida es un desafío significativo. Sin embargo, en los sistemas de bases de datos no relacionales, a menudo no se proporciona el cumplimiento de ACID. Para bases de datos distribuidas no relacionales, las siguientes consideraciones se vuelven más significativas:

- Equilibrar la disponibilidad y la coherencia:

El teorema CAP argumenta que una base de datos distribuida que pretende escalar más allá de una sola red local debe elegir entre disponibilidad y consistencia en el caso de dividir una red. Una base de datos compatible con ACID está obligada a favorecer la coherencia con respecto a cualquier otro factor. Sin embargo, una base de datos no relacional, sin la restricción de ACID estricto, este cumplimiento puede alcanzar un equilibrio diferente.

- Economía de hardware:

Incluso pequeñas diferencias en el costo de los servidores individuales, estos se multiplican rápidamente cuando un sistema escala a miles o cientos de miles de nodos. Por lo tanto, una arquitectura de base de datos económica aprovechará mejor los productos básicos de hardware para aprovechar las mejores relaciones precio / rendimiento disponibles. Además, puede ser necesario poder hacer frente a las disparidades entre configuraciones de servidor, para que el nuevo hardware se pueda agregar al clúster de la base de datos sin requerir que todos los nodos existentes se actualicen a la última especificación de hardware.

- Capacidad de recuperación:

En un clúster de base de datos masivo, los nodos fallarán de vez en cuando. En el evento de estas fallas, no puede haber pérdida de datos, interrupción de la disponibilidad o tal vez incluso falla en el nivel de transacción.



Ha habido tres amplias categorías de arquitectura de base de datos distribuidas adoptadas por la próxima generación bases de datos. Los tres modelos son:

- Variaciones en la arquitectura de fragmentación tradicional, en la que los datos se segmentan en nodos basados en el valor de una "clave de fragmento".
- Variaciones en el modelo Hadoop HDFS / HBase, en el que un "maestro omnisciente" determina dónde deben ubicar los datos en el clúster, en función de la carga y otros factores
- El modelo hash consistente de Amazon Dynamo, en el que los datos se distribuyen a través de nodos del clúster basados en hashing matemático predecible de un valor clave.

La replicación puede ser inherente a cada una de estas arquitecturas para garantizar que no se pierdan datos en el evento de una falla del servidor, aunque las estrategias de replicación varían.

Estos tres enfoque se pueden entender mediante tres bases de datos diferentes:

MongoDB:

Como ejemplo de una arquitectura de sharding. MongoDB admite sharding para proporcionar capacidades de escalado horizontal y replicación para alta disponibilidad. A pesar de que cada uno puede implementarse independientemente del otro, generalmente ambos están presentes en un escenario de producción. en el sharding, cada fragmento es implementado por una base de datos MongoDB distinta, que en la mayoría de los aspectos desconoce su papel en el más amplio servidor fragmentado. Una base de datos independiente de MongoDB, que funciona como servidor de configuración, contiene los metadatos que pueden ser utilizado para determinar cómo se distribuyen los datos a través de los fragmentos. Un enrutador es responsable del enrutamiento y solicita al servidor shard apropiado. Vale la pena recordar, como se vio en la unidad 1, que en MongoDB, una colección se usa para almacenar múltiples documentos JSON que generalmente tienen algunos atributos comunes. Para fragmentar una colección, se elige una clave de fragmento, que es uno o más atributos indexados que se usarán para determinar la distribución de documentos a través de fragmentos. La estructura del árbol



B de el índice MongoDB contiene la información necesaria para distribuir las claves de manera uniforme en los fragmentos.

La distribución de datos a través de fragmentos puede ser basada en rango o basada en hash. En partición basada en rango, a cada fragmento se le asigna un rango específico de valores de clave de fragmento. MongoDB consulta la distribución de valores clave en el índice para garantizar que a cada fragmento se le asigne aproximadamente el mismo número de claves. En hash-based sharding, las claves se distribuyen en función de una función hash aplicada a la clave shard. Hay ventajas y compromisos involucrados en cada esquema. El particionamiento basado en rangos permite una ejecución más eficiente de consultas que procesan rangos de valores, ya que estas consultas a menudo se pueden resolver accediendo a un solo fragmento. La fragmentación basada en hash requiere que las consultas de rango se resuelvan accediendo a todos los fragmentos. Por otro lado, la fragmentación basada en hash es más probable para distribuir documentos "en caliente" (pedidos sin completar o publicaciones recientes, por ejemplo) de manera uniforme en todo el clúster, por lo tanto, equilibrando la carga de manera más efectiva. Sin embargo, cuando la partición de rango está habilitada y la clave de fragmento aumenta continuamente, la carga tiende a agregarse solo a uno de los fragmentos, lo que desequilibra el clúster. Con particionamiento basado en hash los documentos nuevos se distribuyen de manera uniforme en todos los miembros del clúster. Además, aunque MongoDB intenta distribuir las claves de fragmentos uniformemente en el clúster, es posible que haya puntos de acceso dentro de un determinado rangos de claves de fragmentos que nuevamente desequilibren la carga. La fragmentación basada en hash es más probable que distribuya uniformemente la carga en este escenario. El sharding sensible a etiquetas permite al administrador de MongoDB ajustar la distribución de los documentos fragmentos. Al asociar un fragmento con la etiqueta y asociar un rango de claves dentro de una colección con la misma etiqueta, el administrador puede determinar explícitamente el fragmento en el que residirán estos documentos. Esto puede ser utilizado para archivar datos en fragmentos en un almacenamiento más económico y lento o para dirigir datos particulares a un centro de datos específico o geográfico.

HBase:

Como ejemplo de un maestro omnisciente. HBase ee puede pensar como la "base de datos Hadoop" y como "BigTable de código abierto". Es decir, se puede describir HBase como un mecanismo para proporcionar servicios de bases de datos de acceso aleatorio

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



sobre el sistema de archivos Hadoop HDFS, o se la puede pensar como una implementación de código abierto de la base de datos BigTable de Google que pasa a usar HDFS para el almacenamiento de datos. Ambas descripciones son precisas: aunque HBase teóricamente puede implementarse sobre cualquier sistema de archivos distribuidos, o incluso un sistema de archivos no distribuido, casi siempre se implementa sobre Hadoop HDFS, y muchos de los supuestos arquitectónicos de HBase reflejan esto. Por otro lado, HBase implementa la funcionalidad de base de datos de acceso aleatorio en tiempo real, que es esencialmente distinto de las capacidades básicas de Hadoop.

La arquitectura de HBase, comúnmente encontrada, es la de implementado sobre HDFS. Esto crea una especie de híbrido, una mezcla de patrones de clústeres compartidos y nada compartidos. Por un lado, cada nodo de HBase puede acceder a cualquier elemento de datos en la base de datos porque todos los datos son accesibles a través de HDFS. Por lo tanto, es típico ubicar conjuntamente los servidores HBase con HDFS DataNodes, lo que significa que en la práctica cada nodo tiende a ser responsable de un subconjunto exclusivo de datos almacenados en el disco local. En cualquier caso, HDFS proporciona las garantías de fiabilidad para los datos en disco. Esto implica que no es necesario preocuparse por la duplicación de escritura o la gestión de fallas de disco, ya que se manejan automáticamente por el sistema HDFS subyacente.

Cassandra:

Como un ejemplo de hash consistente al estilo Dynamo. Un gran número de sistemas de código abierto han implementado el modelo Dynamo. Una de esas implementaciones es Cassandra. En HBase y MongoDB, uno se encuentra con el concepto de nodos maestro, que tienen una función especializada de supervisión, coordinación de actividades de otros nodos y registro del estado actual de la base de datos. En Cassandra y otras bases de datos de Dynamo, no hay nodos maestros especializados. Cada nodo es igual y cada nodo es capaz de realizar cualquiera de las actividades requeridas para la operación del clúster. Los nodos en Cassandra, sin embargo, tienen responsabilidades especializadas a corto plazo. Por ejemplo, cuando un el cliente realiza una operación, se asignará un nodo como coordinador para esa operación. Cuando un nuevo miembro se agrega al clúster, un nodo se denominará como el nodo semilla del cual el nuevo nodo busca información. Sin embargo, estas responsabilidades a corto plazo pueden ser realizadas por cualquier nodo en el clúster.

Una de las ventajas de un nodo maestro es que puede mantener una versión canónica de la configuración del clúster y su estado. En ausencia de dicho nodo maestro, Cassandra

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



requiere que todos los miembros del el clúster se mantengan actualizados con el estado actual de la configuración y el estado del clúster. Esto se logra mediante el uso del protocolo de chismes (gossip). Cada segundo, cada miembro del clúster transmitirá información sobre su estado y el estado de cualquier otro nodo del que tenga conocimiento hasta hasta otros tres nodos en el clúster. De esta forma, el estado del clúster es constantemente actualizado en todos los miembros del clúster. La configuración del clúster se conserva en el espacio del sistema, que está disponible para todos los miembros. Un espacio de claves es más o menos análogo a un esquema en una base de datos relacional: el espacio de claves del sistema contiene tablas que registran metadatos sobre la configuración del clúster. Esta arquitectura elimina cualquier punto único de falla dentro del clúster. Aunque las bases de datos distribuidas con nodos maestros tienen estrategias para permitir la conmutación por error rápida, el bloqueo de un nodo maestro por lo general crea una reducción temporal en la disponibilidad, por ejemplo, cayendo momentáneamente al modo de solo lectura. Uno de los principales problemas con el protocolo de chismes dentro de un clúster de Cassandra es la disponibilidad de nodos. La detección de fallas de Cassandra es más probabilística: si lo desea, los nodos en el clúster se vuelven cada vez más "Preocupado" por otros nodos. Si parece probable que un nodo esté inactivo, las operaciones se dirigirán a nodos "conocidos buenos".



3. MongoDB

MongoDB es una base de datos multiplataforma y orientada a documentos que proporciona, alto rendimiento, alta disponibilidad, y fácil escalabilidad.

Los conceptos básicos que maneja MongoDB son el de colección y documento, que se repasarán en lo que sigue.

La base de datos es un contenedor físico para colecciones, y cada base de datos obtiene su propio conjunto de archivos en el sistema de archivos.

Colección

Colección es un grupo de documentos MongoDB. Es el equivalente de una tabla RDBMS. Una colección existe dentro de un

única base de datos. Las colecciones no hacen cumplir un esquema, es decir, los documentos dentro de una colección pueden tener diferentes campos.

Por lo general, todos los documentos en una colección tienen un propósito similar o relacionado.

Documento

Un documento es un conjunto de pares clave-valor. Los documentos tienen un esquema dinámico. Esquema dinámico significa que los documentos en la misma colección no necesitan tener el mismo conjunto de campos o estructura, y además los campos comunes en los documentos de una colección pueden contener diferentes tipos de datos.

Puede notarse que este concepto difiere significativamente de la estructura utilizada en bases de datos relacionales.

¿Cuáles son las ventajas de usar mongoDB por sobre una base de datos relacional?



Cualquier base de datos relacional tiene un diseño de esquema típico que muestra el número de tablas y la relación entre estas tablas. Mientras que en MongoDB no hay un concepto de relación. Esto le otorga a MongoDB una flexibilidad muy grande, que suele ser una gran ventaja a la hora de trabajar con Big Data y datos no estructurados. Entre las ventajas de MongoDB sobre RDBMS se pueden citar:

- Esquema más flexible: MongoDB es una base de datos de documentos en la que una colección contiene diferentes documentos. El número de campos, el contenido y el tamaño del documento pueden diferir de un documento a otro. Esto le otorga a MongoDB una flexibilidad sumamente deseable, especialmente con datos no estructurados y en aplicaciones muy dinámicas.
- Capacidad de búsqueda profunda. MongoDB admite consultas dinámicas en documentos utilizando un lenguaje de consulta basado en documentos que es casi tan poderoso como SQL.
- Facilidad de escalamiento horizontal: MongoDB es fácil de escalar. También permite Replicamiento y Sharding, discutido más adelante.
- Conversión / mapeo de objetos de aplicación a objetos de base de datos innecesarios.
- Usa memoria interna para almacenar el conjunto de trabajo (en ventana), lo que permite un acceso más rápido a los datos.

Instalación

Para instalar MongoDB en Windows, primero debe descargarse la última versión de MongoDB desde <http://www.mongodb.org/downloads>

Hay que asegurarse de obtener la versión correcta de MongoDB dependiendo de la versión de Windows (32 o 64 bits)



Luego, debe extraerse el archivo descargado a c: \ drive o cualquier otra ubicación, y asegurarse de que el nombre de la carpeta extraída sea:

- mongodb-win32-i386- [versión]
- mongodb-win32-x86_64- [versión]

Aquí [versión] es la versión de MongoDB

A descargar!

Luego debe abrirse el símbolo del sistema y ejecutar el siguiente comando en la carpeta bin de la carpeta Mongo:

```
mongod.exe --dbpath "d:\set up\mongodb\data"
```

Esto mostrará que la espera del mensaje de conexiones en la salida de la consola indica que el proceso de mongod.exe está corriendo con éxito.

Ahora para ejecutar mongodb es necesario abrir otro símbolo del sistema y escribir el siguiente comando

```
D:\set up\mongodb\bin>mongo.exe
```

```
MongoDB shell version: 2.4.6
```

```
connecting to: test
```

```
>db.test.save( { a: 1 } )
```

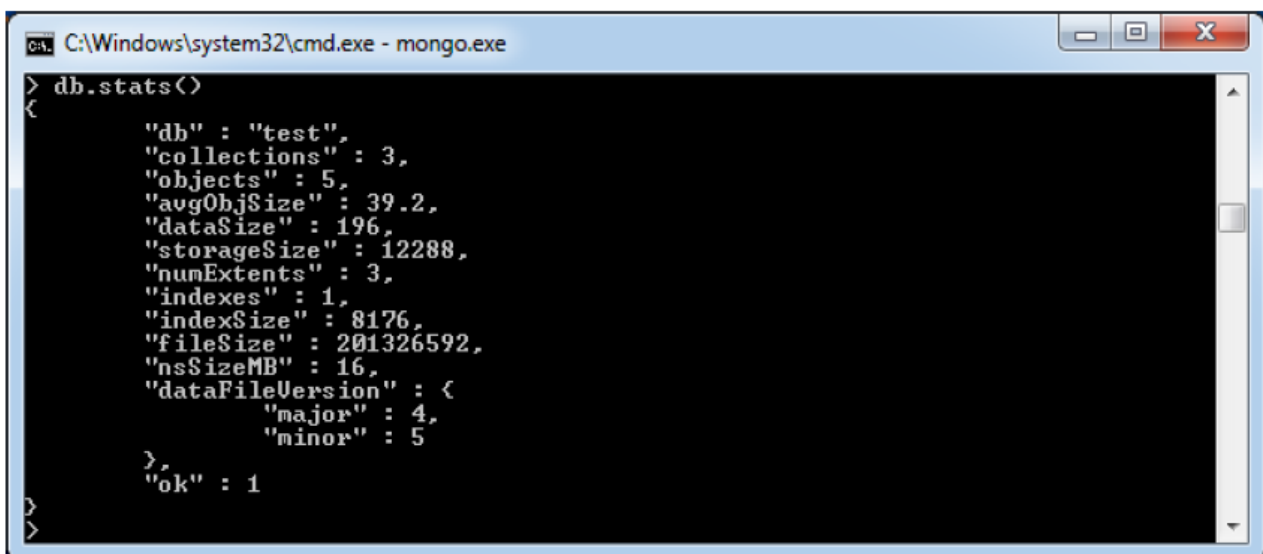



```
>db.test.find()
```

```
{ "_id" : ObjectId(5879b0f65a56a454), "a" : 1 }
```

Para obtener estadísticas sobre el servidor mongodb, debe utilizarse el comando `db.stats()` en el cliente mongodb. Esto mostrará la base de datos nombre, cantidad de colecciones y documentos en la base de datos.

La salida del comando se muestra a continuación:



```
C:\Windows\system32\cmd.exe - mongo.exe
> db.stats()
{
  "db" : "test",
  "collections" : 3,
  "objects" : 5,
  "avgObjSize" : 39.2,
  "dataSize" : 196,
  "storageSize" : 12288,
  "numExtents" : 3,
  "indexes" : 1,
  "indexSize" : 8176,
  "fileSize" : 201326592,
  "nsSizeMB" : 16,
  "dataFileVersion" : {
    "major" : 4,
    "minor" : 5
  },
  "ok" : 1
}
```



Consideraciones para el modelado de datos en mongoDB

Los datos en MongoDB tienen un esquema flexible. Los documentos en la misma colección no necesitan tener el mismo conjunto de campos o la estructura, y los campos comunes en los documentos de una colección pueden contener diferentes tipos de datos.

Algunas consideraciones al diseñar el esquema en MongoDB

- Es importante diseñar el esquema de acuerdo a los requerimientos del usuario.
- Es recomendable combinar objetos en un documento si se usarán juntos. De lo contrario, deben separarse (pero asegurando que no serán necesarias operaciones de unión de los datos).
- Es recomendable duplicar los datos porque el espacio en el disco es barato en comparación con el tiempo de cálculo.
- Es recomendable realizar uniones (joins) mientras se escriben los datos, y no en lectura.
- Si existen agregaciones complejas, es preferible realizarlas directamente al diseñar el esquema.

Ejemplo de modelado de datos

El siguiente ejemplo ha sido adaptado de MONGODB TUTORIAL Simply Easy Learning by tutorialspoint.com

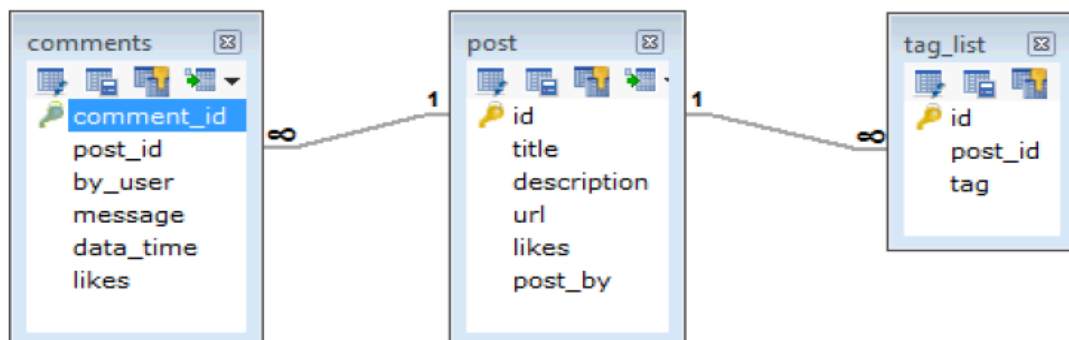
Supongamos que un cliente necesita un diseño de base de datos para su sitio web de blog y quiere ver las diferencias entre RDBMS y un esquema MongoDB.



El sitio web tiene los siguientes requisitos.

- Cada publicación tiene el título único, descripción y url.
- Cada publicación puede tener una o más etiquetas.
- Cada publicación tiene el nombre de su editor y el número total de me gusta.
- Cada publicación tiene comentarios proporcionados por los usuarios junto con su nombre, mensaje, data-time y me gusta.
- En cada publicación, puede haber cero o más comentarios.

En RDBMS, el diseño del esquema para los requisitos anteriores tendrá un mínimo de tres tablas.



En cambio, en mongoDB se tendría:

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



```
{
  _id: POST_ID,
  title: TITLE_OF_POST,
  description: POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
    {
      user: COMMENT_BY,
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    },
    {
      user: COMMENT_BY,
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    }
  ]
}
```

El ejemplo muestra claramente la potencia y practicidad del paradigma NoSQL, donde es posible condensar un conjunto de tablas (en este caso 3) en un sólo objeto (la colección), que puede acomodar relaciones en forma sencilla y directa a través del uso de cadenas JSON.



Tipos de Datos

MongoDB admite muchos tipos de datos cuya lista se proporciona a continuación:

- Cadena: este es el tipo de datos más comúnmente utilizado para almacenar los datos. La cadena en mongodb debe ser válida para UTF-8.
- Entero: este tipo se usa para almacenar un valor numérico. El entero puede ser de 32 bits o 64 bits, dependiendo del servidor.
- Boolean: este tipo se usa para almacenar un valor booleano (verdadero / falso).
- Doble: este tipo se usa para almacenar valores de coma flotante.
- Teclas mín. / Máx .: este tipo se usa para comparar un valor con los elementos BSON más bajos y más altos.
- Arrays: este tipo se usa para almacenar matrices o listar o múltiples valores en una clave.
- Marca de tiempo (timestamp). Esto puede ser útil para registrar cuando un documento ha sido modificado o agregado.
- Objeto: este tipo de datos se usa para documentos insertados.
- Nulo: este tipo se usa para almacenar un valor nulo.
- Símbolo: este tipo de datos se usa de manera idéntica a una cadena, sin embargo, generalmente se reserva para los idiomas que usan un tipo de símbolo específico.
- Fecha: este tipo de datos se usa para almacenar la fecha u hora actual en formato de hora UNIX. Se puede especificar un formato de fecha específico creando un objeto de Fecha y pasando día, mes, año en él.
- ID de objeto: este tipo de datos se usa para almacenar la identificación del documento.

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



- Datos binarios: este tipo de datos se usa para almacenar datos binarios.
- Código: este tipo de datos se usa para almacenar código JavaScript en el documento.
- Expresión regular: este tipo de datos se usa para almacenar expresiones regulares

Ejemplos de Uso de MongoDB

Comandos básicos

Para crear una base de datos, basta con utilizar el comando

```
--  
use DATABASE_NAME  
--
```

donde DATABASE_NAME es el nombre de la base a crear.

Para verificar qué base está actualmente seleccionada, debe utilizarse el comando 'db'. Una vez seleccionada la base correcta (cuyo puntero quedará guardado en la variable 'db'), es posible insertar registros.

Por ejemplo

```
--  
db.pelicula.insert({"Titulo":"Rambo"})
```



--

inserta un documento 'pelicula', con un título dado.

Otro ejemplo del comando 'insert' es el siguiente:

```
--  
db.mycol.insert(  
  _id: ObjectId(7df78ad8902c),  
  title: 'Curso de MongoDB',  
  description: 'Ejemplos prácticos',  
  by: 'UTN',  
  url: 'http://www.utn.edu.ar',  
  tags: ['mongodb', 'base de datos', 'NoSQL'],  
  likes: 150  
)  
--
```

que inserta en la colección 'mycol', el documento con un dado _id, título, descripción, autores, url, tags y likes.



Queries

Si bien la sintaxis entre mongoDB y bases relacionales como SQL es muy distinta, la lógica de los queries es similar. Por ejemplo, para encontrar registros con ciertos valores (el equivalente al WHERE de SQL), podemos utilizar el comando find().

Por ejemplo:

```
--  
db.mycol.find({"by": "UTN", "title": "Curso de MongoDB"}).pretty()  
—
```

devuelve el documento que insertamos anteriormente:

```
{  
  _id: ObjectId(7df78ad8902c),  
  title: 'Curso de MongoDB',  
  description: 'Ejemplos prácticos',  
  by: 'UTN',  
  url: 'http://www.utn.edu.ar',  
  tags: ['mongodb', 'base de datos', 'NoSQL'],  
  likes: 150  
}
```

Aquí, el comando find() se utiliza para encontrar aquellos documentos que tienen al autor (campo 'by') igual a 'UTN', y al título (campo 'title') igual a 'Curso de MongoDB'.

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



El comando `pretty()` que aparece en el ejemplo anterior se utiliza para que la salida (respuesta al query) salga en un formato cómodo de leer.

Los queries con `find()` pueden contener lógica, que se denota con el operador `$`, y combinar distintas relaciones. Por ejemplo, `$and`, `$or`, `$gte` (mayor o igual), y todas las relaciones básicas.

Siguiendo con el caso anterior, se tiene por ejemplo:

```
--  
db.mycol.find({$or:[{"by":"UTN"}, {"title": "Curso de MongoDB"}]}).pretty()  
--
```

Este comando va a producir la misma salida, porque se trata de una operación `$or`, es decir, los documentos que tienen como autor a 'UTN', o bien aquellos que tienen como título a 'Curso de MongoDB'.

Proyección

La proyección se refiere a la operación de restringir el query a algunos campos. Por default, si no se utiliza proyección en un query, entonces el query devuelve todos los campos dentro de la colección.

Por ejemplo:

```
--  
db.mycol.find({}, {"title":1, _id:0})  
--
```



Este query sobre la colección 'mycol' corresponde a encontrar todos los registros (por eso el query es vacío: '{}') pero limitando la salida únicamente al título (campo 'title'), de ahí que aparezca "title":1 como proyección. Es decir, este query no mostrará como resultado a los campos 'by', 'url', etc, y sólo devolverá el título. Por default, el campo intrínseco '_id' (que es un índice automático generado por MongoDB), siempre aparece en un query, a menos que explícitamente indiquemos lo contrario con la proyección _id:0 (que indica que este campo NO debe aparecer en la salida).

Agregaciones y Queries especiales

Las operaciones de agregación procesan registros de datos y devuelven resultados calculados, como por ejemplo el promedio o la suma de un subconjunto de los datos originales.

Las operaciones de agregación agrupan valores de varios documentos juntos, y puede realizar una variedad de operaciones en los datos agrupados para devolver un solo resultado. Las sentencias COUNT(*) o GROUP BY de SQL son el equivalente del comando aggregation mongoDB.

Por ejemplo,

```
--  
db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])  
--
```

tomaría los documentos de la colección 'mycol' y los agruparía por el campo 'by_user', y sumaría los valores del campo 'num_tutorial'.



La sintaxis general es:

```
--  
db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)  
--
```

donde `COLLECTION_NAME` es el nombre de la colección a agregar, y `AGGREGATE_OPERATION` es la operación de agregación a realizar. Algunas de las operaciones de agregación posibles son `$sum` (suma), `$avg` (promedio), `$min` (mínimo), y `$max` (máximo).

Hay que notar también que en el ejemplo de agregación se ha realizado una operación de `$group`, que selecciona un subconjunto de los datos originales, de acuerdo a un grupo. Es el equivalente a la sentencia `GROUP BY` en SQL.

Como comandos auxiliares, y muy útiles al trabajar con alto volumen de datos, se encuentran los comandos `limit()` y `skip()`, que permiten levantar datos por lotes.

```
--  
db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)  
--
```

En un query `find()`, el comando `limit()` permite limitar el número de documentos devueltos en la consulta, mientras que el comando `skip()` permite saltar un cierto número de registros.



Sharding

Sharding es el proceso de almacenar registros de datos en varias máquinas y es el enfoque de MongoDB para cubrir las demandas de crecimiento de datos. A medida que aumenta el tamaño de los datos, una sola máquina puede no ser suficiente para almacenar los datos ni proporcionar un rendimiento de lectura y escritura aceptable. Sharding resuelve el problema de la escala horizontal. Con Sharding, se agregan más máquinas para admitir el crecimiento de datos y las demandas de las operaciones de lectura y escritura.

Cuáles son las ventajas de la arquitectura Sharding?

Por citar sólo algunas, podemos encontrar:

- En la replicación, todas las escrituras van al nodo maestro
- Las consultas sensibles a la latencia aún se dirigen al maestro
- El único conjunto de réplicas tiene una limitación de 12 nodos
- La memoria no puede ser lo suficientemente grande cuando el conjunto de datos activo es grande
- El disco local no es lo suficientemente grande
- El escaleo vertical es muy costoso

La arquitectura Sharding tiene tres componentes esenciales:

- Shards (fragmentos): los shards se utilizan para almacenar datos. Proporcionan alta disponibilidad y coherencia de datos. En entornos de producción cada shard es un conjunto de réplicas por separado.



- **Servidores de configuración:** los servidores de configuración almacenan los metadatos del clúster. Estos datos contienen una asignación de los datos del clúster para asignar a los shards. El enrutador de consulta utiliza estos metadatos para dirigir las operaciones a fragmentos específicos. En producción los clústeres fragmentados de entorno tienen exactamente 3 servidores de configuración.
- **Query Routers (enrutadores):** Query Routers son básicamente instancias mongos, la interfaz con aplicaciones cliente y operaciones al fragmento apropiado. El enrutador de consulta procesa y dirige operaciones a fragmentos y luego devuelve resultados a los clientes. Un clúster fragmentado puede contener más de un enrutador de consulta para dividir la solicitud del cliente. Un cliente envía solicitudes a un enrutador de consulta. En general, un clúster fragmentado tiene muchos enrutadores de consulta.

La arquitectura Sharding esté muy vinculado al concepto de replicación, muy común en computación de alta demanda (por ejemplo como la que es habitual al trabajar con Big Data).

La replicación es el proceso de sincronización de datos en múltiples servidores. La replicación proporciona redundancia y aumenta la disponibilidad de datos con múltiples copias de datos en diferentes servidores de bases de datos, la replicación protege una base de datos de la pérdida de un solo servidor. La replicación también le permite recuperarse de fallas de hardware e interrupciones del servicio.



Con copias adicionales de los datos, se puede dedicar uno a recuperación ante desastres, informes o copias de seguridad.

¿Por qué Replicación?

Para mantener los datos seguros

Alta disponibilidad de datos (24 * 7)

Recuperación de desastres

Sin tiempo de inactividad para mantenimiento (como copias de seguridad, reconstrucción de índices, compactación)

Escalado de lectura (copias adicionales para leer)

El conjunto de réplicas es transparente para la aplicación



Bibliografía utilizada y sugerida

Catherine M. Ricardo, Susan D. Urban, Databases Illuminated, USA New York, Jones & Bartlett Learning, 2017.

Guy Harrison, Next Generation Databases, USA New York, APRESS 2016.

Steve Hoberman, Data Modeling for MongoDB, USA New Jersey, Technics Publications, 2014.



Lo que vimos:

En esta Unidad vimos los conceptos de bases de datos distribuidas y sus usos en Big Data. Comparamos diferentes características de las bases distribuidas y estudiamos en detalle MongoDB y sus diferentes características.



Lo que viene:

Ahora toca el desafío del examen final. Ánimo, es un pequeño paso más!

