capitulo 24

Bases de datos NOSQL y sistemas de almacenamiento de Big Data

24

e anota de para gestión a grandes densidades de

Desarrolladores para gestionar grandes cantidades de datos en organizaciones como Google, Amazon, Facebook y Twitter y en aplicaciones como redes sociales, enlaces web, perfiles de usuario, marketing y ventas, publicaciones y tweets, mapas de carreteras y datos espaciales y correo electrónico. El término **NOSQL** es generalmente se interpreta como No solo SQL, en lugar de NO a SQL, y está destinado para transmitir que muchas aplicaciones necesitan sistemas distintos a los relacionales tradicionales Sistemas SQL para aumentar sus necesidades de gestión de datos. La mayoría de los sistemas NOSQL son bases de datos distribuidas o sistemas de almacenamiento distribuidos, con un enfoque en semis- almacenamiento de datos estructurado, alto rendimiento, disponibilidad, replicación de datos y capacidad en lugar de un énfasis en la coherencia inmediata de los datos, lenguajes de consulta y almacenamiento de datos estructurados.

Comenzamos en la Sección 24.1 con una introducción a los sistemas NOSQL, sus características istics, y en qué se diferencian de los sistemas SQL. También describimos cuatro categorías generales gorías de los sistemas NOSQL: basados en documentos, almacenes de valores clave, basados en columnas, y basado en gráficos. La sección 24.2 analiza cómo los sistemas NOSQL abordan el problema de coherencia entre múltiples réplicas (copias) mediante el uso del paradigma conocido como **consistencia eventual** . Analizamos el teorema **CAP** , que se puede utilizar para comprender destaque el énfasis de los sistemas NOSQL en la disponibilidad. En las Secciones 24.3 hasta 24.6, presentamos una descripción general de cada categoría de sistemas NOSQL, comenzando con sistemas basados en documentos, seguidos de almacenes de valores-clave, luego basados en columnas y finalmente basado en gráficos. Algunos sistemas pueden no caer claramente en una sola categoría, pero en su lugar, utilice técnicas que abarquen dos o más categorías de sistemas NOSQL.

Finalmente, la Sección 24.7 es el resumen del capítulo.

Página 2

24.1 Introducción a los sistemas NOSQL

24.1.1 Aparición de sistemas NOSQL

Muchas empresas y organizaciones se enfrentan a aplicaciones que almacenan grandes cantidades de datos. Considere una aplicación de correo electrónico gratuita, como Google Mail o Yahoo. Correo u otro servicio similar: esta aplicación puede tener millones de usuarios y cada uno el usuario puede tener miles de mensajes de correo electrónico. Existe la necesidad de un sistema de almacenamiento. que pueda gestionar todos estos correos electrónicos; un sistema SQL relacional estructurado puede no ser apropiado porque (1) los sistemas SQL ofrecen demasiados servicios (lenguaje de consulta potente calibre, control de concurrencia, etc.), que esta aplicación puede no necesitar; y (2) un modelo de datos estructurados como el modelo relacional tradicional puede ser demasiado restrictivo. Aunque los sistemas relacionales más nuevos tienen modelos relacionales de objetos más complejos, opciones de ing (ver Capítulo 12), todavía requieren esquemas, que no son requeridos por muchos de los sistemas NOSQL.

Como otro ejemplo, considere una aplicación como Facebook, con millones de usuarios que envían publicaciones, muchas con imágenes y videos; entonces estas publicaciones deben ser que se muestran en las páginas de otros usuarios que utilizan las relaciones de redes sociales entre los usuarios. Los perfiles de usuario, las relaciones con los usuarios y las publicaciones deben almacenarse en una gran colección almacenamiento de datos, y las publicaciones apropiadas deben estar disponibles para los conjuntos de usuarios que se han registrado para ver estas publicaciones. Algunos de los datos para este tipo de aplicaciones catión no es adecuado para un sistema relacional tradicional y normalmente necesita múltiples tipos de bases de datos y sistemas de almacenamiento de datos.

Algunas de las organizaciones que se enfrentaron a esta gestión y almacenamiento de datos aplicaciones decidieron desarrollar sus propios sistemas:

- Google desarrolló un sistema NOSQL patentado conocido como **BigTable** , que es utilizado en muchas de las aplicaciones de Google que requieren grandes cantidades de almacenamiento de datos edad, como Gmail, Google Maps e indexación de sitios web. Apache Hbase es un Sistema NOSQL de código abierto basado en conceptos similares. Innovación de Google condujo a la categoría de sistemas NOSQL conocidos como **basados en columnas** o **amplia** almacenes de **columnas** ; a veces también se les conoce como tiendas de **familia de columnas** .
- Amazon desarrolló un sistema **NOSQL** llamado **DynamoDB** que está disponible a través de los servicios en la nube de Amazon. Esta innovación llevó a la categoría conocida como almacenes de datos de **valor-clave** o, a veces, almacenes de datos **de tuplas clave** o **de objetos clave** .

- Facebook desarrolló inicialmente Apache Cassandra, que ahora está abierto tanto de almacenes de valores-clave como de sistemas basados en columnas.
- Otras empresas de software comenzaron a desarrollar sus propias soluciones y a crear están disponibles para los usuarios que necesitan estas capacidades, por ejemplo, **MongoDB** y **CouchDB** , que se clasifican como sistemas NOSQL **basados en** documentos o **almacenes de documentos** .
- Otra categoría de sistemas NOSQL son los sistemas NOSQL **basados en** gráficos, o **bases de datos gráficas** ; estos incluyen **Neo4J** y **GraphBase** , entre otros.

Página 3

- Algunos sistemas NOSQL, como **OrientDB** , combinan conceptos de muchos las categorías discutidas anteriormente.
- Además de los tipos más nuevos de sistemas NOSQL enumerados anteriormente, también es posible Es posible clasificar los sistemas de bases de datos según el modelo de objetos (consulte el Capítulo 12) o en el modelo XML nativo (ver Capítulo 13) como sistemas NOSQL, aunque pueden no tener las características de alto rendimiento y replicación de los otros tipos de sistemas NOSQL.

Estos son solo algunos ejemplos de sistemas NOSQL que se han desarrollado. Allí Hay muchos sistemas y enumerarlos todos está más allá del alcance de nuestra presentación.

24.1.2 Características de los sistemas NOSQL

Ahora discutimos las características de muchos sistemas NOSQL, y cómo estos sistemas Los sistemas difieren de los sistemas SQL tradicionales. Dividimos las características en dos categorías: aquellas relacionadas con bases de datos distribuidas y sistemas distribuidos, y los relacionados con modelos de datos y lenguajes de consulta.

Características NOSQL relacionadas con bases de datos distribuidas y distribuidas sistemas. Los sistemas NOSQL enfatizan la alta disponibilidad, por lo que replicar los datos es inherente a muchos de estos sistemas. La escalabilidad es otra característica importante, porque muchas de las aplicaciones que utilizan sistemas NOSQL tienden a tener datos que sigue creciendo en volumen. El alto rendimiento es otra característica necesaria, mientras que la consistencia serializable puede no ser tan importante para algunos de los NOSQL aplicaciones. A continuación, discutimos algunas de estas características.

1. **Escalabilidad:** como discutimos en la Sección 23.1.4, hay dos tipos de escalas capacidad en sistemas distribuidos: horizontal y vertical. En los sistemas NOSQL, Generalmente se usa la **escalabilidad horizontal** , donde el sistema distribuido es ampliado agregando más nodos para el almacenamiento y procesamiento de datos a medida que el volumen crece el volumen de datos. La escalabilidad vertical, por otro lado, se refiere a la expansión la capacidad de almacenamiento y computación de los nodos existentes. En los sistemas NOSQL,

La escalabilidad horizontal se emplea mientras el sistema está operativo, por lo que la tecnología
niques para distribuir los datos existentes entre nuevos nodos sin inter-
la operación del sistema de ruptura es necesaria. Discutiremos algunos de estos
técnicas en las Secciones 24.3 a 24.6 cuando hablamos de sistemas específicos.

2. **Disponibilidad, replicación y consistencia eventual:** muchas aplicaciones
que utilizan sistemas NOSQL requieren una disponibilidad continua del sistema. Para acompañar
Si hace esto, los datos se replican en dos o más nodos en un manual transparente.
ner, de modo que si un nodo falla, los datos todavía están disponibles en otros nodos.
La replicación mejora la disponibilidad de datos y también puede mejorar el rendimiento de lectura.
mance, porque las solicitudes de lectura a menudo pueden ser atendidas desde cualquiera de las
nodos de datos cated. Sin embargo, el rendimiento de escritura se vuelve más engorroso
porque se debe aplicar una actualización a cada copia de los elementos de datos replicados;
esto puede ralentizar el rendimiento de escritura si se requiere consistencia serializable
(vea la Sección 23.3). Muchas aplicaciones NOSQL no requieren serializable

Página 4

consistencia, formas más relajadas de consistencia conocidas como **eventuales**
se utilizan **consistencia** . Discutimos esto con más detalle en la Sección 24.2.

3. **Modelos de replicación:** se utilizan dos modelos de replicación principales en el sistema NOSQL.
tems: réplica maestro-esclavo y maestro-maestro. **Replicación maestro-esclavo**
requiere una copia para ser la copia maestra; todas las operaciones de escritura deben aplicarse
a la copia maestra y luego se propaga a las copias esclavas, generalmente usando
consistencia eventual (las copias esclavas eventualmente serán las mismas que las mas-
ter copia). Para leer, el paradigma maestro-esclavo se puede configurar en varios
formas. Una configuración requiere que todas las lecturas también estén en la copia maestra, por lo que
esto sería similar al sitio principal o los métodos de copia principal de distribución
control de concurrencia uted (vea la Sección 23.3.1), con ventajas similares y
desventajas. Otra configuración permitiría lecturas en las copias esclavas
pero no garantizaría que los valores sean las últimas escrituras, ya que se escribe en
los nodos esclavos se pueden realizar después de aplicarlos a la copia maestra. los
La replicación maestro-maestro permite lecturas y escrituras en cualquiera de las réplicas, pero
puede no garantizar que las lecturas en los nodos que almacenan copias diferentes vean el
mismos valores. Diferentes usuarios pueden escribir el mismo elemento de datos simultáneamente en
diferentes nodos del sistema, por lo que los valores del elemento serán temporalmente
inconsistente. Un método de reconciliación para resolver operaciones de escritura conflictivas
del mismo elemento de datos en diferentes nodos debe implementarse como parte de la
esquema de replicación maestro-maestro.

4. **Fragmentación de archivos:** en muchas aplicaciones NOSQL, archivos (o colecciones de datos
objetos) pueden tener muchos millones de registros (o documentos u objetos), y
miles de usuarios pueden acceder simultáneamente a estos registros. Entonces no es
práctico para almacenar todo el archivo en un nodo. **Sharding** (también conocido como

partición horizontal ; consulte la Sección 23.2) de los registros de archivo empleado en sistemas NOSQL. Esto sirve para distribuir la carga de acceder el archivo se registra en varios nodos. La combinación de fragmentar el archivo registros y la replicación de los fragmentos funciona en conjunto para mejorar la carga equilibrio y disponibilidad de datos. Discutiremos algunos de los fragmentos técnicas en las Secciones 24.3 a 24.6 cuando hablamos de sistemas específicos.

5. **Acceso a datos de alto rendimiento:** en muchas aplicaciones NOSQL, es necesario sario para encontrar registros u objetos individuales (elementos de datos) de entre los leones de registros de datos u objetos en un archivo. Para lograr esto, la mayoría de los sistemas utilizan una de dos técnicas: hash o partición de rango en claves de objeto. los la mayoría de los accesos a un objeto serán proporcionando el valor clave en lugar de que mediante el uso de condiciones de consulta complejas. La clave de objeto es similar a la concepto de ID de objeto (ver Sección 12.1). En **hash** , una función hash h (K) es aplicada a la clave K, y se determina la ubicación del objeto con la clave K por el valor de h (K). En la **partición de rango** , la ubicación se determina mediante un rango de valores clave; por ejemplo, la ubicación donde guardaría los objetos cuya clave los valores K están en el rango $Ki_{min} \le K \le Ki_{max}$. En aplicaciones que requieran consultas de rango, donde varios objetos dentro de un rango de valores clave son recuperado, se prefiere el rango particionado. También se pueden utilizar otros índices para ubicar objetos basados en condiciones de atributo diferentes de la clave K.

Página 5

discutirá algunas de las técnicas de hash, particionamiento e indexación en Secciones 24.3 a 24.6 cuando hablamos de sistemas específicos.

Características de NOSQL relacionadas con modelos de datos y lenguajes de consulta. Los sistemas NOSQL enfatizan el rendimiento y la flexibilidad sobre el poder de modelado y consultas complejas. A continuación, discutimos algunas de estas características.

1. **No** requerir un esquema **:** la flexibilidad de no requerir un esquema es logrado en muchos sistemas NOSQL al permitir semi-estructurado, auto-descripción de datos (consulte la Sección 13.1). Los usuarios pueden especificar un esquema parcial en algunos sistemas para mejorar la eficiencia del almacenamiento, pero no es necesario tener un esquema en la mayoría de los sistemas NOSQL. Como puede que no haya un esquema para especificar restricciones, cualquier restricción en los datos tendría que ser pro gramatizado en los programas de aplicación que acceden a los elementos de datos. Existen varios lenguajes para describir datos semiestructurados, como JSON (JavaScript Notación de objetos) y XML (Lenguaje de marcado extensible; consulte el Capítulo 13). JSON se utiliza en varios sistemas NOSQL, pero otros métodos para describir También se pueden utilizar datos semiestructurados. Discutiremos JSON en la Sección 24.3 cuando presentamos sistemas NOSQL basados en documentos.
2. **Lenguajes de consulta menos potentes:** muchas aplicaciones que utilizan el sistema NOSQL Es posible que los tems no requieran un lenguaje de consulta potente como SQL, porque

Las consultas de búsqueda (lectura) en estos sistemas a menudo ubican objetos individuales en una sola archivo basado en sus claves de objeto. Los sistemas NOSQL suelen proporcionar un conjunto de funciones y operaciones como una API de programación (programación de aplicaciones interfaz), por lo que la lectura y escritura de los objetos de datos se logra llamando las operaciones apropiadas por parte del programador. En muchos casos, la operación Las operaciones se denominan **operaciones CRUD** , para crear, leer, actualizar y eliminar. En otros casos, se conocen como **SCRUD** debido a una búsqueda (o búsqueda) agregada operación. Algunos sistemas NOSQL también proporcionan un lenguaje de consulta de alto nivel, pero puede que no tenga toda la potencia de SQL; solo un subconjunto de consultas SQL se proporcionarían capacidades. En particular, muchos sistemas NOSQL no proporcionar operaciones de combinación como parte del propio lenguaje de consulta; las uniones necesitan ser implementado en los programas de aplicación.

3. Control de **versiones:** algunos sistemas NOSQL proporcionan almacenamiento de varias versiones de los elementos de datos, con las marcas de tiempo de cuando se creó la versión de datos. Discutiremos este aspecto en la Sección 24.5 cuando presentemos Sistemas NOSQL.

En la siguiente sección, damos una descripción general de las diversas categorías de NOSQL sistemas.

24.1.3 Categorías de sistemas NOSQL

Los sistemas NOSQL se han caracterizado en cuatro categorías principales, con algunos categorías adicionales que abarcan otros tipos de sistemas. Los más comunes La categorización enumera las siguientes cuatro categorías principales:

Página 6

1. **Sistemas NOSQL basados en documentos:** estos sistemas almacenan datos en forma de documentos que utilizan formatos conocidos, como JSON (Objeto JavaScript Notación). Los documentos son accesibles a través de su ID de documento, pero también se pueden se accede rápidamente utilizando otros índices.
2. **Almacenes de clave-valor NOSQL:** estos sistemas tienen un modelo de datos simple basado en el acceso rápido por la clave al valor asociado con la clave; el valor puede ser un registro o un objeto o un documento o incluso tener datos más complejos estructura.
3. **Sistemas NOSQL de columna ancha o basados en columnas:** estos sistemas dividen un tabla por columna en familias de columnas (una forma de partición vertical; consulte Sección 23.2), donde cada familia de columnas se almacena en sus propios archivos. Ellos también permitir el versionado de valores de datos.
4. **Sistemas NOSQL basados en gráficos: los** datos se representan como gráficos y los nodos se pueden encontrar atravesando los bordes utilizando expresiones de ruta.

Se pueden agregar categorías adicionales de la siguiente manera para incluir algunos sistemas que no categorizar fácilmente en las cuatro categorías anteriores, así como algunos otros tipos de sistemas Temas que han estado disponibles incluso antes de que el término NOSQL se usara ampliamente.

5. **Sistemas NOSQL híbridos:** estos sistemas tienen características de dos o más de las cuatro categorías anteriores.
6. **Bases de datos de objetos:** estos sistemas se discutieron en el Capítulo 12.
7. **Bases de datos XML:** discutimos XML en el Capítulo 13.

Incluso los motores de búsqueda basados en palabras clave almacenan grandes cantidades de datos con búsquedas rápidas. acceso, por lo que los datos almacenados se pueden considerar como grandes almacenes de datos grandes NOSQL.

El resto de este capítulo está organizado de la siguiente manera. En cada una de las Secciones 24.3 a 24.6, discutiremos una de las cuatro categorías principales de sistemas NOSQL y elaboraremos Califique más en qué características se centra cada categoría. Antes de eso, en Sec- 24.2, discutimos con más detalle el concepto de consistencia eventual, y discutir el teorema de CAP asociado.

24.2 El teorema CAP

Cuando discutimos el control de concurrencia en bases de datos distribuidas en la Sección 23.3, asumimos que el sistema de base de datos distribuida (DDBS) es necesario para hacer cumplir la Propiedades ACID (atomicidad, consistencia, aislamiento, durabilidad) de las transacciones que se están ejecutando al mismo tiempo (consulte la Sección 20.3). En un sistema con replicación de datos, El control de divisas se vuelve más complejo porque puede haber múltiples copias de cada elemento de datos. Entonces, si se aplica una actualización a una copia de un elemento, debe aplicarse a todas las demás copias de forma coherente. Existe la posibilidad de que una copia de un El artículo X se actualiza mediante una transacción T_1, mientras que otra copia se actualiza mediante una transacción. T_2, por lo que existen dos copias inconsistentes del mismo elemento en dos nodos diferentes en el sistema distribuido. Si otras dos transacciones T_3 y T_4 quieren leer X, cada una puede lea una copia diferente del artículo X.

Página 7

Vimos en la Sección 23.3 que existen métodos de control de concurrencia distribuidos que no permita esta inconsistencia entre copias del mismo elemento de datos, haciendo cumplir serializabilidad y, por tanto, la propiedad de aislamiento en presencia de replicación. Cómo- Nunca, estas tcnicas a menudo vienen con una gran sobrecarga, que frustrar el propsito plantear la creación de múltiples copias para mejorar el rendimiento y la disponibilidad en sistemas de bases de datos distribuidas como NOSQL. En el campo de los sistemas distribuidos, Hay varios niveles de coherencia entre los elementos de datos replicados, desde el con- consistencia a una consistencia fuerte. Hacer cumplir la serialización se considera el más fuerte forma de consistencia, pero tiene una alta sobrecarga por lo que puede reducir el rendimiento de lectura y escribir operaciones y, por tanto, afectar negativamente al rendimiento del sistema.

El teorema CAP, que se introdujo originalmente como el principio CAP, se puede utilizado para explicar algunos de los requisitos que compiten en un sistema distribuido con replicación. Las tres letras de CAP se refieren a tres propiedades deseables de sistemas con datos replicados: **consistencia** (entre copias replicadas), **disponibilidad** (de el sistema para operaciones de lectura y escritura) y **tolerancia de partición** (frente a la nodos en el sistema que están siendo particionados por una falla de red). Disponibilidad significa que cada solicitud de lectura o escritura de un elemento de datos se procesará correctamente o recibir un mensaje de que la operación no se puede completar. Medios de tolerancia de partición que el sistema puede seguir funcionando si la red que conecta los nodos tiene un falla que da como resultado dos o más particiones, donde los nodos en cada partición pueden sólo se comunican entre sí. La coherencia significa que los nodos tendrán las mismas copias de un elemento de datos replicado visible para varias transacciones.

Es importante señalar aquí que el uso de la palabra consistencia en CAP y su uso en ACID no se refieren al mismo concepto idéntico. En CAP, el término consistencia se refiere a la consistencia de los valores en diferentes copias del mismo elemento de datos en un sistema distribuido replicado. En ACID, se refiere al hecho de que una transacción no violar las restricciones de integridad especificadas en el esquema de la base de datos. Sin embargo, si consideramos que la consistencia de las copias replicadas es una restricción especificada, entonces los dos usos del término coherencia estarían relacionados.

El **teorema CAP** establece que no es posible garantizar los tres valores deseables propiedades (consistencia, disponibilidad y tolerancia de partición) al mismo tiempo en un sistema distribuido con replicación de datos. Si este es el caso, entonces el sistema distribuido El diseñador del tem tendría que elegir dos propiedades de las tres para garantizar. Eso Generalmente se asume que en muchas aplicaciones tradicionales (SQL), garantizar la consistencia a través de las propiedades ACID es importante. Por otro lado, en un Almacén de datos distribuidos de NOSQL, un nivel de consistencia más débil es a menudo aceptable, y garantizar las otras dos propiedades (disponibilidad, tolerancia de partición) es importante tant. Por lo tanto, los niveles de consistencia más débiles se utilizan a menudo en el sistema NOSQL en lugar de garantizando la serialización. En particular, una forma de coherencia conocida como **eventual La coherencia** se adopta a menudo en los sistemas NOSQL. En las Secciones 24.3 a 24.6, discutirá algunos de los modelos de consistencia utilizados en sistemas NOSQL específicos.

Las siguientes cuatro secciones de este capítulo discuten las características de las cuatro categorías principales egorías de los sistemas NOSQL. Analizamos los sistemas NOSQL basados en documentos en Sección 24.3, y usamos MongoDB como sistema representativo. En la Sección 24.4, discutimos

Página 8

Sistemas NOSQL conocidos como almacenes de valores clave. En la Sección 24.5, damos una descripción general de sistemas NOSQL basados en columnas, con una discusión de Hbase como un sistema representativo tem. Finalmente, presentamos los sistemas NOSQL basados en gráficos en la Sección 24.6.

24.3 Sistemas NOSQL basados en documentos y MongoDB

Los sistemas NOSQL basados en documentos u orientados a documentos normalmente almacenan datos como **colecciones** de **documentos** similares . Estos tipos de sistemas también se conocen a veces como **almacenes de documentos** . Los documentos individuales se parecen un poco a objetos complejos (consulte la Sección 12.3) o documentos XML (consulte el Capítulo 13), pero una diferencia importante entre sistemas basados en documentos versus objetos y sistemas relacionales de objetos y XML es que no hay ningún requisito para especificar un esquema; más bien, los documentos son especificados como **datos autodescriptivos** (consulte la Sección 13.1). Aunque los documentos en un la recopilación debe ser similar, pueden tener diferentes elementos de datos (atributos), y Los nuevos documentos pueden tener nuevos elementos de datos que no existen en ninguno de los documentos de la colección. El sistema básicamente extrae los nombres de los elementos de datos. de los documentos autodescriptivos de la colección, y el usuario puede solicitar que el sistema crea índices en algunos de los elementos de datos. Los documentos se pueden especificar fied en varios formatos, como XML (consulte el Capítulo 13). Un lenguaje popular para especificar ify documentos en los sistemas NOSQL es **JSON** (JavaScript Object Notation).

Hay muchos sistemas NOSQL basados en documentos, incluidos MongoDB y CouchDB, entre muchos otros. Daremos una descripción general de MongoDB en esta sección ción. Es importante tener en cuenta que diferentes sistemas pueden utilizar diferentes modelos, idiomas calibres y métodos de implementación, pero dando una encuesta completa de todos Los sistemas NOSQL basados en documentos están más allá del alcance de nuestra presentación.

24.3.1 Modelo de datos de MongoDB

Los documentos de MongoDB se almacenan en formato BSON (JSON binario), que es una variante de JSON con algunos tipos de datos adicionales y es más eficiente para el almacenamiento que JSON. Los **documentos** individuales se almacenan en una **colección** . Usaremos un examen simple basado en la base de datos de nuestra EMPRESA que usamos a lo largo de este libro. los La operación createCollection se utiliza para crear cada colección. Por ejemplo, los siguientes El comando lowing se puede usar para crear una colección llamada **proyecto** para contener PROYECTO objetos de la base de datos de la COMPAÑÍA (ver Figuras 5.5 y 5.6):

```
db.createCollection ("proyecto", {capped: true, size: 1310720, max: 500})
```

El primer parámetro "proyecto" es el **nombre** de la colección, seguido de un documento opcional que especifica **las opciones de colección** . En nuestro ejemplo, la colección está **tapado** ; esto significa que tiene límites superiores en su espacio de almacenamiento (**tamaño**) y número de documentos (**máx** .). Los parámetros de limitación ayudan al sistema a elegir el almacenamiento opciones para cada colección. Hay otras opciones de recopilación, pero no las maldícelos aquí.

Página 9

Para nuestro ejemplo, crearemos otra colección de documentos llamada **trabajador** para mantener información sobre los EMPLEADOS que trabajan en cada proyecto; para ejemplo:

db.createCollection ("trabajador", {capped: true, size: 5242880, max: 2000}))

Cada documento de una colección tiene un campo **ObjectId** único , llamado **_id** , que es indexado automáticamente en la colección a menos que el usuario solicite explícitamente ningún índice para el campo _id. El valor de ObjectId puede ser especificado por el usuario, o puede ser generado por el sistema si el usuario no especifica un campo _id para un documento en particular. Los ObjectIds generados por el sistema tienen un formato específico, que combina la marca de tiempo cuando se crea el objeto (4 bytes, en un formato interno de MongoDB), la identificación del nodo (3 bytes), la identificación del proceso (2 bytes) y un contador (3 bytes) en un valor de identificación de 16 bytes. Los ObjectsIds generados por el usuario pueden tener cualquier valor especificado por el usuario siempre que identifica de forma única el documento, por lo que estos identificadores son similares a las claves primarias en sistemas relacionales.

Una colección no tiene esquema. La estructura de los campos de datos en los documentos. se elige en función de cómo se accederá y utilizará los documentos, y el usuario puede elegir un diseño normalizado (similar a las tuplas relacionales normalizadas) o una desnor- diseño malizado (similar a documentos XML u objetos complejos). Interdocumento Las referencias se pueden especificar almacenando en un documento el ObjectId o ObjectIds de otros documentos relacionados. La figura 24.1 (a) muestra un documento MongoDB simplificado mostrando algunos de los datos de la Figura 5.6 del ejemplo de la base de datos de EMPRESA que se utiliza en todo el libro. En nuestro ejemplo, los valores de _id están definidos por el usuario, y los documentos cuyo _id comience con P (para proyecto) se almacenarán en el "proyecto" colección, mientras que aquellos cuyo _id comienza con W (para trabajador) se almacenarán en la Colección "trabajador".

En la Figura 24.1 (a), la información de los trabajadores está incluida en el documento del proyecto; entonces no hay necesidad de la colección "trabajador". Esto se conoce como el patrón desnormalizado tern, que es similar a la creación de un objeto complejo (consulte el Capítulo 12) o un XML documento (ver Capítulo 13). Una lista de valores entre corchetes […] dentro de un documento representa un campo cuyo valor es una **matriz** .

Otra opción es utilizar el diseño de la Figura 24.1 (b), donde las referencias de los trabajadores son incrustado en el documento del proyecto, pero los propios documentos del trabajador son almacenados en una colección separada de "trabajadores". Una tercera opción en la Figura 24.1 (c) sería utilizar un diseño normalizado, similar a las relaciones de la primera forma normal (consulte la sección sección 14.3.4). La elección de qué opción de diseño utilizar depende de cómo se se accederá.

Es importante notar que el diseño simple de la Figura 24.1 (c) no es la norma general. Diseño malizado para una relación de muchos a muchos, como la que existe entre empleados. y proyectos; más bien, necesitaríamos tres colecciones para "proyecto", "empleado" y “Works_on”, como discutimos en detalle en la Sección 9.1. Muchas de las compensaciones del diseño que se discutieron en los Capítulos 9 y 14 (para las relaciones de primera forma normal y para ER- opciones de mapeo relacional) y los Capítulos 12 y 13 (para objetos complejos y XML) son aplicables para elegir el diseño apropiado para estructuras de documentos.

Figura 24.1
Ejemplo de simple documentos en MongoDB.
(a) desnormalizado diseño de documentos con incrustado subdocumentos.
(b) Matriz incrustada de referencias de documentos.
(c) Normalizado documentos.

(a) documento de proyecto con una serie de trabajadores integrados:

```
{
        _carné de identidad:    "P1",
        Pname:                  "ProductoX",
        Ubicación:              "Bellaire",
        Trabajadores: [
                        {Ename: "John Smith",
                          Horas: 32.5
                        },
                        {Ename: "Joyce English",
                          Horas: 20.0
                        }
                ]
);
```

(b) documento de proyecto con una matriz integrada de ID de trabajador:

```
{
        _carné de identidad:    "P1",
        Pname:                  "ProductoX",
        Ubicación:              "Bellaire",
        WorkerIds:              ["W1", "W2"]
}
        { _carné de identidad:  "W1",
        Ename:                  "John Smith",
        Horas:                  32,5
}
        { _carné de identidad:  "W2",
        Ename:                  "Joyce English",
        Horas:                  20,0
}
```

(c) documentos normalizados del proyecto y del trabajador (no un diseño completamente normalizado para relaciones M: N):

```
{
        _carné de identidad:    "P1",
        Pname:                  "ProductoX",
        Ubicación:              "Bellaire"
}
{ _carné de identidad:          "W1",
        Ename:                  "John Smith",
        Projecto ID:            "P1",
        Horas:                  32,5
}
```

```
{ _carné de identidad:    "W2",
      Ename:              "Joyce English",
      Projecto ID:        "P1",
      Horas:              20,0
}
```

(d) insertar los documentos en (c) en sus colecciones "proyecto" y "trabajador":

```
db.project.insert ({_id: "P1" , Pname: "ProductX", Ubicación: "Bellaire"})
db.worker.insert ([{_id: "W1", Ename: "John Smith", ProjectId: "P1", Horas: 32.5},
                   {_id: "W2" , Ename: "Joyce English", ProjectId: "P1",
                    Horas: 20.0}])
```

Figura 24.1 (continuado) Ejemplo de simple documentos en MongoDB. (d) Insertar los documentos en Figura 24.1 (c) en sus colecciones.

y colecciones de documentos, por lo que no repetiremos las discusiones aquí. En el diseño En la Figura 24.1 (c), un EMPLEADO que trabaja en varios proyectos estaría representado enviado por varios documentos de trabajador con diferentes valores _id; cada documento representaría al empleado como trabajador para un proyecto en particular. Esto es similar a las decisiones de diseño para el diseño de esquemas XML (consulte la Sección 13.6). Sin embargo, es de nuevo Es importante tener en cuenta que el sistema típico basado en documentos no tiene un esquema, por lo que las reglas de diseño tendrían que seguirse siempre que los documentos individuales sean insertado en una colección.

24.3.2 Operaciones CRUD de MongoDB

MongoDb tiene varias **operaciones CRUD** , donde CRUD significa (crear, leer, actualizar, eliminar). Los documentos se pueden crear e insertar en sus colecciones utilizando la operación de **inserción** , cuyo formato es:

db. <nombre_colección> .insertar (<documento (s)>)

Los parámetros de la operación de inserción pueden incluir un solo documento o un conjunto de documentos, como se muestra en la Figura 24.1 (d). La operación de eliminación se llama **eliminar** , y el formato es:

db. <nombre_colección> .remove (<condición>)

Los documentos que se eliminarán de la colección se especifican mediante una condición booleana. dición sobre algunos de los campos en los documentos de la colección. También hay una **actualización** operación, que tiene una condición para seleccionar ciertos documentos, y una cláusula $ set para especificar la actualización. También es posible utilizar la operación de actualización para reemplazar un documento existente con otro pero mantener el mismo ObjectId.

Para las consultas de lectura, el comando principal se llama **buscar** y el formato es:

db. <nombre_colección> .find (<condición>)

Las condiciones booleanas generales se pueden especificar como <condición> y los documentos en la colección que devuelve **verdadero** se selecciona para el resultado de la consulta. Para una discusión completa de las operaciones CRUD de MongoDb, consulte la documentación en línea de MongoDB en el referencias de capítulos.

Pagina 12

24.3.3 Características de los sistemas distribuidos MongoDB

La mayoría de las actualizaciones de MongoDB son atómicas si se refieren a un solo documento, pero MongoDB también proporciona un patrón para especificar transacciones en varios documentos. Ya que MongoDB es un sistema distribuido, el método de **confirmación de dos fases** se utiliza para garantizar atomicidad y consistencia de transacciones de documentos múltiples. Discutimos el atomiciudad y propiedades de consistencia de las transacciones en la Sección 20.3, y las dos fases protocolo de confirmación en la Sección 22.6.

Replicación en MongoDB. El concepto de **conjunto** de **réplicas** se utiliza en MongoDB para crear múltiples copias del mismo conjunto de datos en diferentes nodos en el sistema distribuido, y utiliza una variación del enfoque **maestro-esclavo** para la replicación. Por ejemplo, suponga que queremos replicar una colección de documentos en particular C.Un conjunto de réplicas tendrá una **copia primaria** de la colección C almacenada en un nodo N1, y al menos una **copia secundaria** (réplica) de C almacenada en otro nodo N2. Copias adicionales pueden ser almacenados en los nodos N3, N4, etc., según sea necesario, pero el costo de almacenamiento y actualización (escritura) aumenta con el número de réplicas. El número total de participantes en un conjunto de réplicas debe ser al menos tres, por lo que si solo se necesita una copia secundaria, un participante en el El conjunto de réplicas conocido como **árbitro** debe ejecutarse en el tercer nodo N3. El árbitro no tiene una réplica de la colección pero participa en las **elecciones** para elegir una nueva primaria si el nodo que almacena la copia principal actual falla. Si el número total de miembros en un replica conjunto es n (un primario más i secundarios, para un total de n = i + 1), entonces n debe ser un impar número; si no es así, se agrega un árbitro para garantizar que el proceso de elección funcione correctamente si el primario falla. Discutimos las elecciones en sistemas distribuidos en la Sección 23.3.1.

En la replicación de MongoDB, todas las operaciones de escritura deben aplicarse a la copia principal y luego se propagó a los secundarios. Para operaciones de lectura, el usuario puede elegir la **preferencia de lectura** particular para su aplicación. La preferencia de lectura predeterminada procesa todas las lecturas en la copia primaria, por lo que todas las operaciones de lectura y escritura se realizan formado en el nodo primario. En este caso, las copias secundarias son principalmente para asegurarse que el sistema continúa funcionando si falla el primario, y MongoDB puede garantizar que cada solicitud de lectura obtiene el valor del documento más reciente. Para aumentar el rendimiento de lectura mance, es posible establecer la preferencia de lectura para que las solicitudes de lectura se puedan procesar en cualquier réplica (primaria o secundaria); sin embargo, una lectura en una secundaria no es garantía teed para obtener la última versión de un documento porque puede haber un retraso en la propagación Gating escribe desde el primario hasta el secundario.

Fragmentación en MongoDB. Cuando una colección contiene una gran cantidad de documentos mentos o requiere un gran espacio de almacenamiento, almacenar todos los documentos en un nodo puede dar lugar a problemas de rendimiento, especialmente si hay muchas operaciones de usuario acceder a los documentos simultáneamente utilizando varias operaciones CRUD. **Fragmentación** de los documentos de la colección (también conocida como partición horizontal) divide los documentos en particiones separadas conocidas como **fragmentos** . Esto permite sistema para agregar más nodos según sea necesario mediante un proceso conocido como **escalado horizontal** de el sistema distribuido (consulte la Sección 23.1.4), y para almacenar los fragmentos de la colección en diferentes nodos para lograr el equilibrio de carga. Cada nodo procesará solo aquellos operaciones pertenecientes a los documentos del fragmento almacenado en ese nodo. Además, cada

Página 13

el fragmento contendrá menos documentos que si toda la colección se almacenara en uno nodo, mejorando así aún más el rendimiento.

Hay dos formas de dividir una colección en fragmentos en MongoDB: **rango particionamiento** y **particionamiento hash** . Ambos requieren que el usuario especifique un particular campo de documento que se utilizará como base para dividir los documentos en fragmentos. El campo de partición, conocido como **clave de** partición en MongoDB, debe tener dos características: debe existir en todos los documentos de la colección, y debe tener un índice. Se puede usar el ObjectId, pero cualquier otro campo que posea estos dos caracteres: istics también se pueden utilizar como base para la fragmentación. Los valores de la clave de fragmento son dividido en **trozos,** ya sea a través de partición de rango o partición hash, y la los documentos se dividen en función de los fragmentos de valores clave de fragmentos.

La partición por rango crea los fragmentos especificando un rango de valores clave; por ejemplo, si los valores de la clave del fragmento oscilaron entre uno y diez millones, es posible crear diez rangos: 1 a 1.000.000; 1.000.001 a 2.000.000; …; 9,000,001 a 10,000,000 — y cada fragmento contendría los valores clave en un rango. La partición hash aplica un función hash h (K) a cada clave de fragmento K, y la partición de claves en fragmentos es basado en los valores hash (discutimos el hash y sus ventajas y desventajas en la Sección 16.8). En general, si las **consultas de rango** se aplican comúnmente a una colección (por ejemplo, recuperar todos los documentos cuyo valor de clave de fragmento esté entre 200 y 400), entonces se prefiere la partición de rango porque cada consulta de rango generalmente se enviará ted a un solo nodo que contiene todos los documentos necesarios en un fragmento. Si la mayoría las búsquedas recuperan un documento a la vez, la partición hash puede ser preferible porque aleatoriza la distribución de los valores de clave de fragmentos en fragmentos.

Cuando se usa la fragmentación, las consultas de MongoDB se envían a un módulo llamado **consulta enrutador** , que realiza un seguimiento de qué nodos contienen qué fragmentos en función de las método de partición lar utilizado en las claves de fragmentos. La consulta (operación CRUD) será enrutado a los nodos que contienen los fragmentos que contienen los documentos que la consulta es solicitando. Si el sistema no puede determinar qué fragmentos contienen la documentación requerida mentos, la consulta se enviará a todos los nodos que contienen fragmentos de la colección. La fragmentación y la replicación se utilizan juntas; fragmentación se centra en mejorar el rendimiento mance a través del equilibrio de carga y la escalabilidad horizontal, mientras que la replicación se centra en asegurar la disponibilidad del sistema cuando fallan ciertos nodos en el sistema distribuido.

Hay muchos detalles adicionales sobre la arquitectura del sistema distribuido y la componentes de MongoDB, pero una discusión completa está fuera del alcance de nuestra presentación. MongoDB también proporciona muchos otros servicios en áreas como administración de sistemas, indexación, seguridad y agregación de datos, pero no discutiremos estas características aquí. La documentación completa de MongoDB está disponible en línea (consulte las notas bibliográficas).

24.4 Almacenes de valores-clave NOSQL

Las tiendas de valor clave se centran en el alto rendimiento, la disponibilidad y la escalabilidad al almacenar datos en un sistema de almacenamiento distribuido. El modelo de datos utilizado en los almacenes de valores-clave es relativo tivamente simple, y en muchos de estos sistemas, no hay un lenguaje de consulta sino más bien un

Página 14

conjunto de operaciones que pueden utilizar los programadores de aplicaciones. La **clave** es un identificador único asociado con un elemento de datos y se utiliza para localizar este elemento de datos rápidamente. El **valor** es el elemento de datos en sí mismo y puede tener formatos muy diferentes para diferentes sistemas de almacenamiento de valores clave. En algunos casos, el valor es solo una cadena de bytes o una matriz de bytes, y la aplicación que utiliza el almacén de valores-clave debe interpretar la estructura del valor de los datos. En otros casos, algunos datos formateados estándar se permitido; por ejemplo, filas de datos estructurados (tuplas) similares a los datos relacionales, o datos semiestructurados utilizando JSON o algún otro formato de datos autodescriptivo. Diferir de- Las tiendas de valores clave ent pueden almacenar datos no estructurados, semiestructurados o estructurados elementos (consulte la Sección 13.1). La principal característica de las tiendas de valores clave es el hecho de que cada valor (elemento de datos) debe estar asociado con una clave única, y que la recuperación de la El valor al suministrar la clave debe ser muy rápido.

Hay muchos sistemas que caen bajo la etiqueta de tienda de valor clave, por lo que en lugar de pro- Para ver muchos detalles sobre un sistema en particular, daremos una breve descripción general ver algunos de estos sistemas y sus características.

24.4.1 Descripción general de DynamoDB

El sistema DynamoDB es un producto de Amazon y está disponible como parte de Amazon Plataformas **AWS** / **SDK** (Amazon Web Services / Software Development Kit). Puede ser utilizado como parte de los servicios de computación en la nube de Amazon, para el componente de almacenamiento de datos.

Modelo de datos de DynamoDB. El modelo de datos básico en DynamoDB usa los conceptos de tablas, elementos y atributos. Una **tabla** en DynamoDB no tiene un **esquema** ; eso contiene una colección de elementos autodescriptivos. Cada **elemento** constará de una serie de Los pares (atributo, valor) y los valores de atributo pueden ser de un solo valor o de varios valores. Entonces Básicamente, una mesa contendrá una colección de elementos, y cada elemento es una descripción registro (u objeto). DynamoDB también permite al usuario especificar los elementos en JSON para- mat, y el sistema los convertirá al formato de almacenamiento interno de DynamoDB.

Cuando se crea una tabla, es necesario especificar un **nombre de tabla** y una **clave primaria** ; la clave principal se utilizará para localizar rápidamente los elementos de la tabla. Por tanto, el pri- mary key es la **clave** y el elemento es el **valor** del almacén de clave-valor de DynamoDB. El atributo de clave principal debe existir en todos los elementos de la tabla. La clave primaria puede ser uno de los siguientes dos tipos:

- **Un solo atributo.** El sistema DynamoDB utilizará este atributo para crear un índice hash en los elementos de la tabla. Esto se denomina clave primaria de tipo hash. Los artículos no se ordenan en almacenamiento según el valor del atributo hash.

- **Un par de atributos.** Esto se denomina clave primaria de tipo hash y rango. los La clave principal será un par de atributos (A, B): el atributo A se utilizará para hashing, y debido a que habrá varios elementos con el mismo valor de A, el B Los valores se utilizarán para ordenar los registros con el mismo valor A. Una mesa con este tipo de clave puede tener índices secundarios adicionales definidos en su atributos. Por ejemplo, si queremos almacenar varias versiones de algún tipo de elementos en una tabla, podríamos usar ItemID como hash y fecha o marca de tiempo (cuando la versión fue creada) como rango en una clave primaria de tipo hash y rango.

Página 15

Características distribuidas de DynamoDB. Dado que DynamoDB es propietario, en En la siguiente subsección analizaremos los mecanismos utilizados para la replicación, fragmentación, y otros conceptos de sistemas distribuidos en un sistema de clave-valor de código abierto llamado Voldemort. Voldemort se basa en muchas de las técnicas propuestas para DynamoDB.

24.4.2 Almacén de datos distribuidos de valores-clave de Voldemort

Voldemort es un sistema de código abierto disponible a través de la licencia de código abierto Apache 2.0. reglas de ing. Está basado en DynamoDB de Amazon. La atención se centra en el alto rendimiento y escalabilidad horizontal, así como en proporcionar replicación para alta disponibilidad y fragmentación para mejorar la latencia (tiempo de respuesta) de las solicitudes de lectura y escritura. Los tres de esas características (replicación, fragmentación y escalabilidad horizontal) se realizan a través de una técnica para distribuir los pares clave-valor entre los nodos de una distribución clúster uted; esta distribución se conoce como **hash consistente** . Voldemort ha sido utilizado por LinkedIn para el almacenamiento de datos. Algunas de las características de Voldemort son las siguientes:

- **Operaciones básicas sencillas.** Una colección de pares (clave, valor) se mantiene en un **Tienda** Voldemort . En nuestra discusión, asumiremos que la tienda se llama s. los La interfaz básica para el almacenamiento y la recuperación de datos es muy simple e incluye tres operaciones: obtener, poner y eliminar. La operación s.put (k, v) inserta un elemento como un par clave-valor con clave k y valor v. La operación s.delete (k) borra el artículo cuya clave es k de la tienda, y la operación v = s.get (k) recupera el valor v asociado con la clave k. La aplicación puede utilizar estos operaciones básicas para construir sus propios requisitos. En el nivel de almacenamiento básico, tanto las claves como los valores son matrices de bytes (cadenas).
- **Valores de datos formateados de alto nivel.** Los valores v en los elementos (k, v) se pueden especificado en JSON (notación de objetos JavaScript), y el sistema convertirá entre JSON y el formato de almacenamiento interno. Otros formatos de objetos de datos pueden También se especificará si la aplicación proporciona la conversión (también conocida como **serialización**) entre el formato de usuario y el formato de almacenamiento como serializador clase. La clase de serializador debe ser proporcionada por el usuario e incluirá aciones para convertir el formato de usuario en una cadena de bytes para el almacenamiento como valor,

y volver a convertir una cadena (matriz de bytes) recuperada a través de s.get (k) en el usuario formato. Voldemort tiene algunos serializadores integrados para formatos distintos a JSON.

- **Hash consistente para distribuir pares (clave, valor).** Una variación del El algoritmo de distribución de datos conocido como **hash consistente** se utiliza en Voldemort para la distribución de datos entre los nodos en el clúster distribuido de nodos. Se aplica una función hash h (k) a la clave k de cada par (k, v), y h (k) determina dónde se almacenará el artículo. El método asume que h (k) es un valor entero, generalmente en el rango de 0 a Hmax = 2^{n-1}, donde n es elegido en función del rango deseado para los valores hash. Este método es el mejor visualizado considerando el rango de todos los posibles valores hash enteros 0 a Hmáx debe distribuirse uniformemente en un círculo (o anillo). Los nodos en la distribución los sistemas uted también se ubican en el mismo anillo; normalmente cada nodo tienen varias ubicaciones en el anillo (consulte la Figura 24.2). El posicionamiento del puntos en el anillo que representan los nodos se realiza de manera pseudoaleatoria.

Página 16

Se almacenará un elemento (k, v) en el nodo cuya posición en el anillo sigue la posición de h (k) en el anillo en el sentido de las agujas del reloj. En la figura 24.2 (a), suponga que hay tres nodos en el clúster distribuido etiquetados A, B y C, donde el nodo C tiene una capacidad mayor que los nodos A y B. En un sistema típico, habrá muchos más nodos. En el círculo, dos instancias de A y B se colocan, y tres instancias de C (debido a su mayor capacidad), en un manera pseudoaleatoria para cubrir el círculo. La figura 24.2 (a) indica qué Los elementos (k, v) se colocan en qué nodos según los valores de h (k).

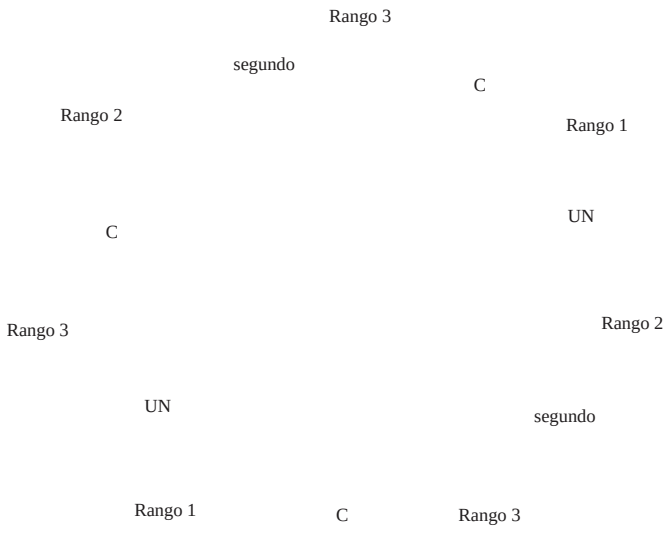
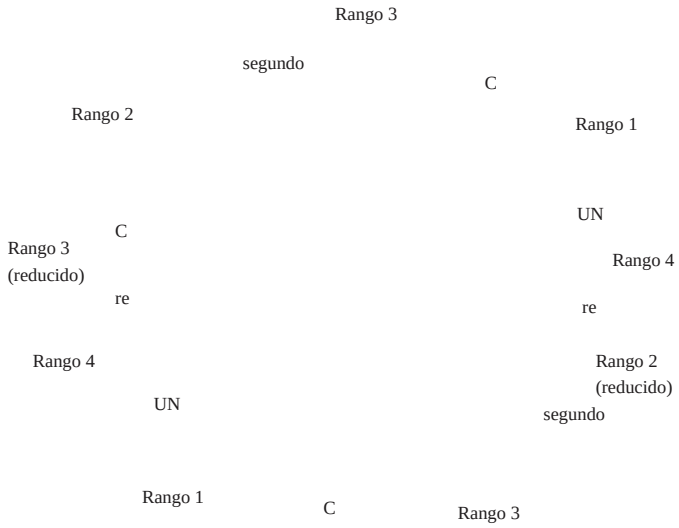


Figura 24.2
Ejemplo de consistente hash. (un anillo tener tres nodos A, B y C, con C teniendo mayor capacidad. los h (K) valores que se asignan a el círculo apunta en rango 1 tiene su (k, v) elementos almacenados en el nodo A, rango 2 en el nodo B, rango 3 en el nodo C. (b) Agregar un nodo D a el anillo. Elementos en el rango 4 se mueve a el nodo D del nodo B (el rango 2 se reduce) y el nodo C (el rango 3 es reducido).



Página 17

- Los valores de h (k) que caen en las partes del círculo marcadas como rango 1 en la Figura ure 24.2 (a) tendrá sus elementos (k, v) almacenados en el nodo A porque ese es el nodo cuya etiqueta sigue a h (k) en el anillo en el sentido de las agujas del reloj; aquellos en el rango 2 se almacenan en el nodo B; y los del rango 3 se almacenan en el nodo C. Este esquema permite la escalabilidad horizontal porque cuando se agrega un nuevo nodo a la distribución sistema uted, se puede agregar en una o más ubicaciones en el anillo dependiendo en la capacidad del nodo. Solo se reasignará un porcentaje limitado de los elementos (k, v) firmado al nuevo nodo desde los nodos existentes en función del hash consistente ing algoritmo de ubicación. Además, los elementos asignados al nuevo nodo pueden no todos provienen de solo uno de los nodos existentes porque el nuevo nodo puede tener múltiples ubicaciones en el ring. Por ejemplo, si se agrega un nodo D y tiene dos colocaciones en el anillo como se muestra en la Figura 24.2 (b), luego algunos de los elementos de los nodos B y C se moverían al nodo D. Los elementos cuyas claves rango 4 en el círculo (ver Figura 24.2 (b)) se migraría al nodo D. Este El esquema también permite la replicación colocando el número de réplicas especificadas de un elemento en sucesivos nodos del anillo en el sentido de las agujas del reloj. La fragmentación está integrado en el método, y los diferentes elementos de la tienda (archivo) se encuentran en diferentes nodos en el clúster distribuido, lo que significa que los elementos son horizon- recuento particionado (fragmentado) entre los nodos del sistema distribuido. Cuando un nodo falla, su carga de elementos de datos se puede distribuir a los otros nodos existentes cuyas etiquetas siguen las etiquetas del nodo fallido en el anillo. Y nodos con

mayor capacidad puede tener más ubicaciones en el anillo, como lo ilustra el nodo C en la figura 24.2 (a) y, por lo tanto, almacenan más elementos que los nodos de menor capacidad.

- **Coherencia y control de versiones.** Voldemort usa un método similar al desarrollado para DynamoDB para mantener la coherencia en presencia de réplicas. Basi- En términos generales, diferentes procesos permiten operaciones de escritura simultáneas, por lo que podrían existir dos o más valores diferentes asociados con la misma clave en diferentes diferentes nodos cuando los elementos se replican. La consistencia se logra cuando el El elemento se lee mediante una técnica conocida como control de versiones y reparación de lectura. Estafa- se permiten escrituras actuales, pero cada escritura está asociada con un reloj vectorial valor. Cuando se produce una lectura, es posible que diferentes versiones del mismo Los valores (asociados con la misma clave) se leen desde diferentes nodos. Si el el sistema puede reconciliarse con un único valor final, pasará ese valor a la lectura; de lo contrario, se puede devolver más de una versión a la aplicación, que reconciliará las diversas versiones en una versión basada en el semántica de la aplicación y devolver este valor conciliado a los nodos.

24.4.3 Ejemplos de otros almacenes de valores clave

En esta sección, revisamos brevemente otros tres almacenes de valores-clave. Es importante tener en cuenta que hay muchos sistemas que se pueden clasificar en esta categoría, y solo podemos mencione algunos de estos sistemas.

Almacén de valores-clave de Oracle. Oracle tiene uno de los datos relacionales SQL más conocidos: sistemas base, y Oracle también ofrece un sistema basado en el concepto de tienda de valor clave; este sistema se llama **Oracle NoSQL Database** .

Página 18

Almacenamiento y caché de valores-clave de Redis. **Redis se** diferencia de los otros sistemas distribuidos maldecido aquí porque almacena en caché sus datos en la memoria principal para mejorar aún más el rendimiento mance. Ofrece replicación maestro-esclavo y alta disponibilidad, y también ofrece persistencia haciendo una copia de seguridad de la caché en el disco.

Apache Cassandra. **Cassandra** es un sistema NOSQL que no se categoriza fácilmente en una categoría; a veces se incluye en la categoría NOSQL basada en columnas (consulte Sección 24.5) o en la categoría clave-valor. Si ofrece funciones de varios NOSQL categorías y es utilizado por Facebook, así como por muchos otros clientes.

24.5 Columna ancha o basada en columnas Sistemas NOSQL

Otra categoría de sistemas NOSQL se conoce como **columna ancha** o **basada en columnas.** sistemas. El sistema de almacenamiento distribuido de Google para big data, conocido como **BigTable** , es

un ejemplo bien conocido de esta clase de sistemas NOSQL, y se utiliza en muchos
Aplicaciones de Google que requieren una gran cantidad de almacenamiento de datos, como Gmail. Grande-
Table utiliza el **sistema de archivos de Google** (**GFS**) para el almacenamiento y la distribución de datos. Un
El sistema de código abierto conocido como **Apache Hbase** es algo similar a Google Big-
Tabla, pero normalmente utiliza **HDFS** (**Hadoop Distributed File System**) para el almacenamiento de datos.
años. HDFS se utiliza en muchas aplicaciones de computación en la nube, como veremos en
Capítulo 25. Hbase también puede utilizar el **sistema de almacenamiento simple** de Amazon (conocido como **S3**)
para el almacenamiento de datos. Otro ejemplo conocido de sistemas NOSQL basados en columnas es
Cassandra, que discutimos brevemente en la Sección 24.4.3 porque también puede
acterizado como un almacén de valores-clave. Nos centraremos en Hbase en esta sección como ejemplo
de esta categoría de sistemas NOSQL.

BigTable (y Hbase) a veces se describe como una distribución multidimensional dispersa
uted mapa ordenado persistente, donde la palabra mapa significa una colección de (clave, valor)
pares (la clave se asigna al valor). Una de las principales diferencias que distinguen
sistemas basados en columnas de almacenes de valores-clave (consulte la Sección 24.4) es la naturaleza del
llave. En sistemas basados en columnas como Hbase, la clave es multidimensional y por lo tanto
tiene varios componentes: normalmente, una combinación de nombre de tabla, clave de fila, columna,
y marca de tiempo. Como veremos, la columna se compone típicamente de dos componentes
nents: familia de columnas y calificador de columnas. Discutimos estos conceptos en más
detalle a continuación, ya que se realizan en Apache Hbase.

24.5.1 Versión y modelo de datos de Hbase

Modelo de datos de Hbase. El modelo de datos en Hbase organiza los datos usando los conceptos
de espacios de nombres, tablas, familias de columnas, calificadores de columna, columnas, filas y datos
células. Una columna se identifica mediante una combinación de (familia de columnas: calificador de columna).
Los datos se almacenan en forma autodescriptiva asociando columnas con valores de datos,
donde los valores de datos son cadenas. Hbase también almacena varias versiones de un elemento de datos,
con una marca de tiempo asociada con cada versión, por lo que las versiones y las marcas de tiempo también se

Página 19

parte del modelo de datos de Hbase (esto es similar al concepto de control de versiones de atributos en
bases de datos temporales, que discutiremos en la Sección 26.2). Como con otros NOSQL
sistemas, claves únicas se asocian con elementos de datos almacenados para un acceso rápido, pero el
Las claves identifican las celdas en el sistema de almacenamiento. Porque el foco está en el alto rendimiento
cuando se almacenan grandes cantidades de datos, el modelo de datos incluye algunos datos relacionados con el almacenamiento
conceptos. Discutimos los conceptos de modelado de datos de Hbase y definimos la terminología
ogy siguiente. Es importante notar que el uso de las palabras tabla, fila y columna es
no es idéntico a su uso en bases de datos relacionales, pero los usos están relacionados.

- **Tablas y Filas.** Los datos en Hbase se almacenan en **tablas** , y cada tabla tiene un
  nombre de la tabla. Los datos de una tabla se almacenan como **filas** autodescriptivas . Cada fila tiene un

clave de fila única , y las claves de fila son cadenas que deben tener la propiedad que pueden ordenarse lexicográficamente, por lo que los caracteres que no tienen un lexico El orden cográfico del conjunto de caracteres no se puede utilizar como parte de una clave de fila.

- **Familias de columnas, calificadores de columnas y columnas.** Una tabla esta asociada con una o más **familias de columnas** . Cada familia de columnas tendrá un nombre, y las familias de columnas asociadas con una tabla deben especificarse cuando el La tabla se crea y no se puede cambiar más tarde. La figura 24.3 (a) muestra cómo una tabla puede ser creado; El nombre de la tabla va seguido de los nombres de la familia de columnas. se encuentra asociado con la mesa. Cuando los datos se cargan en una tabla, cada La familia de **columnas** se puede asociar con muchos **calificadores de columna** , pero la columna los calificadores no se especifican como parte de la creación de una tabla. Entonces los calificadores de columna hacer del modelo un modelo de datos autodescriptivo porque los calificadores pueden ser dinámicamente especificado a medida que se crean nuevas filas y se insertan en la tabla. UN **column** se especifica mediante una combinación de ColumnFamily: ColumnQualifier. Básicamente, las familias de columnas son una forma de agrupar columnas relacionadas (atributos en terminología relacional) para fines de almacenamiento, excepto que el los nombres de los calificadores de columna no se especifican durante la creación de la tabla. Más bien, ellos se especifican cuando los datos se crean y almacenan en filas, por lo que los datos se describir ya que cualquier nombre de calificador de columna se puede usar en una nueva fila de datos (vea la Figura 24.3 (b)). Sin embargo, es importante que el programa de aplicación Los usuarios saben qué calificadores de columna pertenecen a cada familia de columnas, incluso aunque tienen la flexibilidad de crear nuevos calificadores de columna sobre la marcha cuando se crean nuevas filas de datos. El concepto de familia de columnas es algo similar a la partición vertical (ver Sección 23.2), porque las columnas (atributo butes) a los que se accede juntos porque pertenecen a la misma columna family se almacenan en los mismos archivos. Cada familia de columnas de una tabla se almacena en sus propios archivos utilizando el sistema de archivos HDFS.

- **Versiones y marcas de tiempo.** Hbase puede mantener varias **versiones** de un elemento de datos, junto con la **marca de tiempo** asociada con cada versión. La marca de tiempo es una número entero largo que representa la hora del sistema cuando la versión fue creado, por lo que las versiones más nuevas tienen valores de marca de tiempo más grandes. Hbase utiliza mid- night '1 de enero de 1970 UTC' como valor de marca de tiempo cero y utiliza un entero largo que mide el número de milisegundos desde ese momento cuando el sistema valor de marca de tiempo (esto es similar al valor devuelto por la utilidad Java java.util.Date.getTime () y también se usa en MongoDB). También es posible

Página 20

902 Capítulo 24 Bases de datos NOSQL y sistemas de almacenamiento de Big Data

Figura 24.3

Ejemplos en Hbase. (a) Crear una tabla llamada EMPLEADO con tres familias de columnas: Nombre, Dirección y Detalles. (b) Insertar algunos en la tabla EMPLEADO; diferentes filas pueden tener diferentes calificadores de columna autodescriptivos (Fname, Lname, Nickname, Mname, Minit, Suffix,… para la columna Nombre de la familia; Trabajo, Revisión, Supervisor, Salario para los detalles de la familia de columnas). (c) Algunas operaciones CRUD de Hbase.

(a) creando una tabla:
crear 'EMPLEADO', 'Nombre', 'Dirección', 'Detalles'

(b) insertar algunos datos de fila en la tabla EMPLOYEE:
poner 'EMPLEADO', 'fila1', 'Nombre: Fnombre', 'Juan'
poner 'EMPLEADO', 'fila1', 'Nombre: Lname', 'Smith'
poner 'EMPLEADO', 'fila1', 'Nombre: Apodo', 'Johnny'
poner 'EMPLEADO', 'fila1', 'Detalles: Trabajo', 'Ingeniero'
poner 'EMPLEADO', 'fila1', 'Detalles: Revisión', 'Bueno'
poner 'EMPLEADO', 'fila2', 'Nombre: Fnombre', 'Alicia'
poner 'EMPLEADO', 'fila2', 'Nombre: Lnombre', 'Zelaya'
poner 'EMPLEADO', 'fila2', 'Nombre: MName', 'Jennifer'
poner 'EMPLEADO', 'fila2', 'Detalles: Trabajo', 'DBA'
poner 'EMPLEADO', 'fila2', 'Detalles: Supervisor', 'James Borg'
poner 'EMPLEADO', 'fila3', 'Nombre: Fname', 'James'
poner 'EMPLEADO', 'fila3', 'Nombre: Minit', 'E'
poner 'EMPLEADO', 'fila3', 'Nombre: Lname', 'Borg'
poner 'EMPLEADO', 'fila3', 'Nombre: Sufijo', 'Jr.'
poner 'EMPLEADO', 'fila3', 'Detalles: trabajo', 'CEO'
poner 'EMPLEADO', 'fila3', 'Detalles: Salario', '1,000,000'

(c) Algunas operaciones CRUD básicas de Hbase:
Creando una tabla: cree <tablename>, <column family>, <column family>,…
Insertar datos: poner <tablename>, <rowid>, <column family>: <column qualifier>, <value>
Leyendo datos (todos los datos en una tabla): escanear <tablename>
Recuperar datos (un elemento): obtener <tablename>, <rowid>

que el usuario defina el valor de la marca de tiempo explícitamente en un formato de fecha en lugar de utilizando la marca de tiempo generada por el sistema.

- **Células.** Una **celda** contiene un elemento de datos básico en Hbase. La clave (dirección) de una celda es especificado por una combinación de (table, rowid, columnfamily, columnqualifier, marca de tiempo). Si se omite la marca de tiempo, se recupera la última versión del elemento a menos que se especifique un número predeterminado de versiones, digamos las últimas tres versiones. El número predeterminado de versiones que se recuperarán, así como el número predeterminado de versiones que el sistema necesita mantener, son parámetros que pueden especificarse fied durante la creación de la tabla.
- **Espacios de nombres.** Un **espacio de nombres** es una colección de tablas. Un espacio de nombres básicamente especifica una colección de una o más tablas que normalmente se utilizan juntas por aplicaciones de usuario, y corresponde a una base de datos que contiene una colección de tablas en terminología relacional.

Página 21

24.5.2 Operaciones CRUD de Hbase

Hbase tiene operaciones CRUD (crear, leer, actualizar, eliminar) de bajo nivel, como en muchos los sistemas NOSQL. Los formatos de algunas de las operaciones CRUD básicas en Hbase se muestran en la Figura 24.3 (c).

Hbase solo proporciona operaciones CRUD de bajo nivel. Es responsabilidad del programas de aplicación para implementar operaciones más complejas, como combinaciones entre filas en diferentes tablas. La operación de creación crea una nueva tabla y especifica identifica una o más familias de columnas asociadas con esa tabla, pero no especifica los calificadores de columna, como comentamos anteriormente. La operación put se utiliza para insertar ing nuevos datos o nuevas versiones de elementos de datos existentes. La operación de obtención es para recuperar ing los datos asociados con una sola fila en una tabla, y la operación de escaneo recupera todas las filas.

24.5.3 Conceptos de sistemas distribuidos y almacenamiento Hbase

Cada tabla Hbase se divide en varias **regiones** , donde cada región contendrá una rango de las claves de fila en la tabla; es por eso que las claves de fila deben ser lexicográficamente ordenado. Cada región tendrá un número de **tiendas** , donde cada familia de columnas es asignado a una tienda dentro de la región. Las regiones se asignan a los **servidores de la región** (nodos de almacenamiento) para almacenamiento. Un **servidor maestro** (nodo maestro) es responsable de monitorear Toring los servidores de la región y para dividir una tabla en regiones y asignar regiones a los servidores de la región.

Hbase utiliza el sistema de código abierto **Apache Zookeeper** para servicios relacionados con envejecimiento del nombre, distribución y sincronización de los datos de Hbase en la pantalla nodos de servidor Hbase tributados, así como para servicios de coordinación y replicación. Hbase también utiliza Apache HDFS (Hadoop Distributed File System) para distribuciones servicios de archivo. Entonces, Hbase está construido sobre HDFS y Zookeeper. El guardián del zoológico puede tiene varias réplicas en varios nodos para disponibilidad, y mantiene los datos que necesita en la memoria principal para acelerar el acceso a los servidores maestros y servidores de la región.

No cubriremos los muchos detalles adicionales sobre la arquitectura del sistema distribuido. y componentes de Hbase; una discusión completa está fuera del alcance de nuestra presentación. Completo La documentación de Hbase está disponible en línea (ver las notas bibliográficas).

24.6 Bases de datos de gráficos NOSQL y Neo4j

Otra categoría de sistemas NOSQL se conoce como **bases de datos de gráficos** o **gráficos** sistemas **NOSQL orientados** . Los datos se representan como un gráfico, que es una colección de vértices (nodos) y aristas. Tanto los nodos como los bordes se pueden etiquetar para indicar el tipos de entidades y relaciones que representan, y generalmente es posible almacenar datos asociados con nodos individuales y bordes individuales. Muchos sistemas Los elementos se pueden clasificar como bases de datos de gráficos. Centraremos nuestra discusión en uno sistema particular, Neo4j, que se utiliza en muchas aplicaciones. Neo4j es un abierto sistema fuente, y está implementado en Java. Discutiremos el modelo de datos de Neo4j

en la Sección 24.6.1, y dar una introducción a las capacidades de consulta de Neo4j en Sección 24.6.2. La sección 24.6.3 ofrece una descripción general de los sistemas distribuidos y algunas otras características de Neo4j.

24.6.1 Modelo de datos de Neo4j

El modelo de datos en Neo4j organiza los datos utilizando los conceptos de **nodos** y **relaciones barcos** . Tanto los nodos como las relaciones pueden tener **propiedades** , que almacenan los elementos de datos asociado con nodos y relaciones. Los nodos pueden tener **etiquetas** ; los nodos que tienen la misma etiqueta se agrupan en una colección que identifica un subconjunto de los nodos en el gráfico de la base de datos para fines de consulta. Un nodo puede tener cero, uno o varios etiquetas. Las relaciones están dirigidas; cada relación tiene un nodo de inicio y un nodo final como así como un **tipo de relación** , que cumple una función similar a una etiqueta de nodo al identificar relaciones similares que tienen el mismo tipo de relación. Las propiedades se pueden especificar a través de un **patrón de mapa** , que está formado por uno o más pares "nombre: valor" incluidos entre llaves; por ejemplo {Lname: 'Smith', Fname: 'John', Minit: 'B'} .

En la teoría de grafos convencional, los nodos y las relaciones se denominan generalmente vértices. y bordes. El modelo de datos de gráficos de Neo4j se parece un poco a cómo se representan los datos. presentado en los modelos ER y EER (véanse los capítulos 3 y 4), pero con algunos diferencias. Comparando el modelo gráfico de Neo4j con los conceptos ER / EER, los nodos responder a entidades, las etiquetas de nodo corresponden a tipos y subclases de entidad, relación Los barcos corresponden a instancias de relación, los tipos de relación corresponden a los tipos de relación y las propiedades corresponden a los atributos. Una diferencia notable es que una relación está dirigida en Neo4j, pero no en ER / EER. Otro es que un El nodo puede no tener etiqueta en Neo4j, lo cual no está permitido en ER / EER porque cada entidad debe pertenecer a un tipo de entidad. Una tercera diferencia crucial es que el gráfico El modelo de Neo4j se utiliza como base para una distribución de datos real de alto rendimiento. sistema base mientras que el modelo ER / EER se utiliza principalmente para el diseño de bases de datos.

La figura 24.4 (a) muestra cómo se pueden crear algunos nodos en Neo4j. Hay varios formas en que se pueden crear nodos y relaciones; por ejemplo, llamando a apro- Priate las operaciones de Neo4j desde varias API de Neo4j. Solo mostraremos el alto nivel sintaxis para crear nodos y relaciones; para hacerlo, usaremos Neo4j CREATE comando, que es parte del lenguaje de consulta declarativo de alto nivel **Cypher** . Neo4j tiene muchas opciones y variaciones para crear nodos y relaciones usando varios interfaces de secuencias de comandos, pero una discusión completa está fuera del alcance de nuestra presentación.

- **Etiquetas y propiedades.** Cuando se crea un nodo, se puede especificar la etiqueta del nodo. fied. También es posible crear nodos sin etiquetas. En la Figura 24.4 (a), el las etiquetas de nodo son EMPLEADO, DEPARTAMENTO, PROYECTO y UBICACIÓN, y los nodos creados corresponden a algunos de los datos de la EMPRESA base de datos en la Figura 5.6 con algunas modificaciones; por ejemplo, usamos EmpId en lugar de SSN, y solo incluimos un pequeño subconjunto de los datos para ilustrar propósitos. Las propiedades están entre llaves {…}. Es posible que algunos nodos tienen múltiples etiquetas; por ejemplo, el mismo nodo se puede etiquetar como PERSONA y EMPLEADO y GERENTE enumerando todas las etiquetas separadas por el símbolo de dos puntos de la siguiente manera: PERSONA: EMPLEADO: GERENTE. Teniendo múltiples etiquetas es similar a una entidad que pertenece a un tipo de entidad (PERSONA)

más algunas subclases de PERSONA (a saber, EMPLEADO y GERENTE) en el modelo EER (consulte el Capítulo 4) pero también se puede utilizar para otros fines.

- **Relaciones y tipos de relaciones.** La figura 24.4 (b) muestra algunos ejemplos relaciones en Neo4j basadas en la base de datos EMPRESA en la Figura 5.6. El → especifica la dirección de la relación, pero la relación puede ser atravesado en cualquier dirección. Los tipos de relaciones (etiquetas) en la Figura 24.4 (b) son WorksFor, Manager, LocationIn y WorksOn; solo relaciones con el tipo de relación WorksOn tiene propiedades (Horas) en la Figura 24.4 (b).

- **Caminos.** Una **ruta** especifica un recorrido de parte del gráfico. Normalmente se utiliza como parte de una consulta para especificar un patrón, donde la consulta se recuperará del gráficos de datos que coincidan con el patrón. Una ruta generalmente se especifica mediante un inicio nodo, seguido de una o más relaciones, que conducen a uno o más extremos nodos que satisfacen el patrón. Es algo similar a los conceptos de camino. expresiones que discutimos en los Capítulos 12 y 13 en el contexto de la consulta lenguajes para bases de datos de objetos (OQL) y XML (XPath y XQuery).

- **Esquema opcional.** Un **esquema** es opcional en Neo4j. Se pueden crear gráficos y se utiliza sin un esquema, pero en Neo4j versión 2.0, algunos esquemas relacionados se agregaron funciones. Las principales características relacionadas con la creación de esquemas involucran creando índices y restricciones basados en las etiquetas y propiedades. por ejemplo, es posible crear el equivalente de una restricción de clave en una propiedad erty de una etiqueta, por lo que todos los nodos de la colección de nodos asociados con la etiqueta debe tener valores únicos para esa propiedad.

- **Indexación e identificadores de nodos.** Cuando se crea un nodo, el sistema Neo4j crea un identificador interno único definido por el sistema para cada nodo. A recuperar nodos individuales utilizando otras propiedades de los nodos de manera eficiente, el usuario puede crear **índices** para la colección de nodos que tienen un particular etiqueta. Normalmente, una o más de las propiedades de los nodos de esa colección se puede indexar. Por ejemplo, Empid se puede utilizar para indexar nodos con el Etiqueta EMPLEADO, Dno para indexar los nodos con la etiqueta DEPARTAMENTO, y Pno para indexar los nodos con la etiqueta PROYECTO.

24.6.2 El lenguaje de consulta cifrado de Neo4j

Neo4j tiene un lenguaje de consulta de alto nivel, Cypher. Hay comandos declarativos para crear nodos y relaciones (ver Figuras 24.4 (a) y (b)), así como para encontrar nodos y relaciones basadas en patrones específicos. Eliminación y modificación de los datos también son posibles en Cypher. Introdujimos el comando CREATE en el anterior , por lo que ahora daremos una breve descripción de algunas de las otras características de Cypher.

Una consulta Cypher se compone de cláusulas. Cuando una consulta tiene varias cláusulas, el resultado de una cláusula puede ser la entrada a la siguiente cláusula de la consulta. Daremos un flavor del lenguaje discutiendo algunas de las cláusulas usando ejemplos. Nuestra presentación no pretende ser una presentación detallada de Cypher, sólo una introducción a algunas de las funciones de idiomas. La figura 24.4 (c) resume algunas de las cláusulas principales que puede ser parte de una consulta cibernética. El lenguaje cibernético puede especificar consultas complejas y actualizaciones en una base de datos de gráficos. Daremos algunos ejemplos para ilustrar simples Consultas cibernéticas en la Figura 24.4 (d).

Página 24

Figura 24.4
Ejemplos en Neo4j usando el lenguaje Cypher. (a) Creación de algunos nodos. (b) Crear algunas relaciones.

(a) crear algunos nodos para los datos de la COMPAÑÍA (de la Figura 5.6):

```
CREAR (e1: EMPLEADO, {Empid: '1', Lname: 'Smith', Fname: 'John', Minit: 'B'})
CREAR (e2: EMPLEADO, {Empid: '2', Lname: 'Wong', Fname: 'Franklin'})
CREAR (e3: EMPLOYEE, {Empid: '3', Lname: 'Zelaya', Fname: 'Alicia'})
CREATE (e4: EMPLOYEE, {Empid: '4', Lname: 'Wallace', Fname: 'Jennifer', Minit: 'S'})
...
CREAR (d1: DEPARTAMENTO, {Dno: '5', Dname: 'Investigación'})
CREAR (d2: DEPARTMENT, {Dno: '4', Dname: 'Administration'})
...
CREAR (p1: PROYECTO, {Pno: '1', Pname: 'ProductX'})
CREAR (p2: PROYECTO, {Pno: '2', Pname: 'ProductY'})
CREAR (p3: PROYECTO, {Pno: '10', Pname: 'Informatización'})
CREAR (p4: PROYECTO, {Pno: '20', Pname: 'Reorganización'})
...
CREAR (loc1: UBICACIÓN, {Lname: 'Houston'})
CREAR (loc2: UBICACIÓN, {Lname: 'Stafford'})
CREAR (loc3: UBICACIÓN, {Lname: 'Bellaire'})
CREAR (loc4: UBICACIÓN, {Lname: 'Sugarland'})
...
```

(b) crear algunas relaciones para los datos de la EMPRESA (de la Figura 5.6):

```
CREAR (e1) - [: WorksFor] -> (d1)
CREAR (e3) - [: WorksFor] -> (d2)
...
CREAR (d1) - [: Gerente] -> (e2)
CREAR (d2) - [: Gerente] -> (e4)
...
CREAR (d1) - [: Ubicado en] -> (loc1)
CREAR (d1) - [: Ubicado en] -> (loc3)
CREAR (d1) - [: Ubicado en] -> (loc4)
CREAR (d2) - [: Ubicado en] -> (loc2)
...
CREAR (e1) - [: WorksOn, {Horas: '32 .5 '}] -> (p1)
CREAR (e1) - [: WorksOn, {Hours: '7.5'}] -> (p2)
CREAR (e2) - [: WorksOn, {Hours: '10 .0 '}] -> (p1)
CREAR (e2) - [: WorksOn, {Horas: 10.0}] -> (p2)
CREAR (e2) - [: WorksOn, {Hours: '10 .0 '}] -> (p3)
CREAR (e2) - [: WorksOn, {Hours: 10.0}] -> (p4)
...
```

Página 25

Figura 24.4 (continuación)
Ejemplos en Neo4j usando el lenguaje Cypher. (c) Sintaxis básica de consultas Cypher. (d) Ejemplos de consultas Cypher.

(c) Sintaxis básica simplificada de algunas cláusulas cifradas comunes:

Encontrar nodos y relaciones que coincidan con un patrón: COINCIDIR <patrón>
Especificación de agregados y otras variables de consulta: CON <especificaciones>
Especificación de condiciones sobre los datos a recuperar: DONDE <condición>
Especificando los datos a devolver: RETURN <data>
Ordenar los datos a devolver: ORDER BY <data>
Limitar el número de elementos de datos devueltos: LIMIT <número máximo>
Creación de nodos: CREATE <nodo, etiquetas y propiedades opcionales>
Crear relaciones: CREAR <relación, tipo de relación y propiedades opcionales>
Eliminación: DELETE <nodos o relaciones>
Especificar valores de propiedad y etiquetas: SET <valores de propiedad y etiquetas>
Eliminando valores de propiedad y etiquetas: REMOVE <valores de propiedad y etiquetas>

(d) Ejemplos de consultas cifradas simples:

1. COINCIDIR (d: DEPARTAMENTO {Dno: '5'}) - [: Ubicado en] → (loc)
   RETORNO d.Dname, loc.Lname
2. COINCIDIR (e: EMPLEADO {Empid: '2'}) - [w: WorksOn] → (p)
   VOLVER e.Ename, w.Hours, p.Pname
3. COINCIDIR (e) - [w: WorksOn] → (p: PROJECT {Pno: 2})
   VOLVER p.Pname, e.Ename, w.Hours
4. COINCIDIR (e) - [w: WorksOn] → (p)
   VOLVER e.Ename, w.Hours, p.Pname
   PEDIR POR e.Ename
5. COINCIDIR (e) - [w: WorksOn] → (p)
   VOLVER e.Ename, w.Hours, p.Pname
   PEDIR POR e.Ename
   LÍMITE 10
6. COINCIDIR (e) - [w: WorksOn] → (p)
   CON e, CUENTA (p) COMO numOfprojs
   DONDE numOfprojs> 2
   VOLVER e.Ename, numOfprojs
   ORDEN POR numOfprojs
7. COINCIDIR (e) - [w: WorksOn] → (p)
   VOLVER e, w, p
   PEDIR POR e.Ename
   LÍMITE 10
8. MATCH (e: EMPLOYEE {Empid: '2'})
   SET e.Job = 'Ingeniero'

La consulta 1 de la figura 24.4 (d) muestra cómo utilizar las cláusulas MATCH y RETURN en una consulta, y la consulta recupera las ubicaciones del departamento número 5. Coincide con la especificación

fies el patrón y las variables de consulta (d y loc) y RETURN especifica la consulta
resultado que se recuperará haciendo referencia a las variables de consulta. La consulta 2 tiene tres variables
(e, w y p), y devuelve los proyectos y las horas semanales que el empleado con

Página 26

Empid = 2 funciona. La consulta 3, por otro lado, devuelve los empleados y las horas
por semana que trabajan en el proyecto con Pno = 2. La consulta 4 ilustra el ORDER BY
cláusula y devuelve todos los empleados y los proyectos en los que trabajan, ordenados por Ename. Eso
También es posible limitar el número de resultados devueltos utilizando la cláusula LIMIT como
en la consulta 5, que solo devuelve las primeras 10 respuestas.

La consulta 6 ilustra el uso de WITH y agregación, aunque la cláusula WITH puede
utilizarse para separar cláusulas en una consulta incluso si no hay agregación. La consulta 6 también ilustra
procesa la cláusula WHERE para especificar condiciones adicionales, y la consulta devuelve el
empleados que trabajan en más de dos proyectos, así como el número de proyectos de cada uno
empleado trabaja. También es común devolver los nodos y las relaciones.
en el resultado de la consulta, en lugar de los valores de propiedad de los nodos como en el
consultas. La consulta 7 es similar a la consulta 5 pero solo devuelve los nodos y las relaciones,
y así el resultado de la consulta se puede mostrar como un gráfico usando la herramienta de visualización de Neo4j. Está
también es posible agregar o eliminar etiquetas y propiedades de los nodos. La Consulta 8 muestra cómo
agregue más propiedades a un nodo agregando una propiedad de trabajo a un nodo de empleado.

Lo anterior da una idea breve del lenguaje de consulta Cypher de Neo4j. El idioma completo
guage manual está disponible en línea (ver las notas bibliográficas).

24.6.3 Interfaces Neo4j y características del sistema distribuido

Neo4j tiene otras interfaces que se pueden usar para crear, recuperar y actualizar nodos y
relaciones en una base de datos gráfica. También tiene dos versiones principales: la edi-
ción, que viene con capacidades adicionales, y la edición comunitaria. Dis-
maldice algunas de las características adicionales de Neo4j en esta subsección.

- **Edición empresarial frente a edición comunitaria.** Ambas ediciones son compatibles con Neo4j
  modelo de datos de gráfico y sistema de almacenamiento, así como la consulta de gráfico Cypher
  lenguaje y varias otras interfaces, incluida una interfaz nativa de alto rendimiento
  API, controladores de lenguaje para varios lenguajes de programación populares, como
  Java, Python, PHP y la API REST (Representational State Transfer). En
  Además, ambas ediciones admiten propiedades ACID. La edición empresarial
  admite funciones adicionales para mejorar el rendimiento, como el almacenamiento en caché
  y agrupación de datos y bloqueo.
- **Interfaz de visualización de gráficos.** Neo4j tiene una interfaz de visualización de gráficos, por lo que
  que un subconjunto de los nodos y bordes en un gráfico de base de datos se puede mostrar como un
  grafico. Esta herramienta se puede utilizar para visualizar los resultados de la consulta en una representación gráfica.

- **Replicación maestro-esclavo.** Neo4j se puede configurar en un clúster de instancias
  Replicación maestro-esclavo. Neo4j se puede configurar en un clúster de instancias
  nodo. Los datos y los índices se replican por completo en cada nodo del clúster.
  Varias formas de sincronizar los datos entre los nodos maestro y esclavo pueden
  configurarse en el clúster distribuido.
- **Almacenamiento en caché.** Se puede configurar una memoria caché principal para almacenar los datos del gráfico.
  para un mejor rendimiento.
- **Registros lógicos.** Los registros se pueden mantener para recuperarse de fallas.

Página 27

Una discusión completa de todas las características e interfaces de Neo4j está fuera del alcance de nuestra presentación. La documentación completa de Neo4j está disponible en línea (consulte la biblio-notas gráficas).

24.7 Resumen

En este capítulo, discutimos la clase de sistemas de bases de datos conocidos como sistema NOSQL.
tems, que se centran en el almacenamiento y la recuperación eficientes de grandes cantidades de "macrodatos".
Las aplicaciones que utilizan este tipo de sistemas incluyen redes sociales, enlaces web, usuarios
perfiles, marketing y ventas, publicaciones y tweets, hojas de ruta y datos espaciales, y
correo electrónico. El término NOSQL generalmente se interpreta como No solo SQL, en lugar de
NO a SQL, y está destinado a transmitir que muchas aplicaciones necesitan otros sistemas
que los sistemas SQL relacionales tradicionales para aumentar sus necesidades de gestión de datos.
Estos sistemas son bases de datos distribuidas o sistemas de almacenamiento distribuidos, con un enfoque
sobre almacenamiento de datos semiestructurados, alto rendimiento, disponibilidad, replicación de datos,
y escalabilidad en lugar de un énfasis en la coherencia inmediata de los datos,
lenguajes de consulta y almacenamiento de datos estructurados.

En la Sección 24.1, comenzamos con una introducción a los sistemas NOSQL, sus características
características y en qué se diferencian de los sistemas SQL. Cuatro categorías generales de NOSQL
Los sistemas están basados en documentos, almacenes de valores clave, basados en columnas y basados en gráficos.
En la Sección 24.2, discutimos cómo los sistemas NOSQL abordan el tema de la coherencia
tency entre múltiples réplicas (copias) utilizando el paradigma conocido como eventual
consistencia. Discutimos el teorema CAP, que se puede utilizar para comprender el
énfasis de los sistemas NOSQL en la disponibilidad. En las Secciones 24.3 a 24.6,
presentó una descripción general de cada una de las cuatro categorías principales de sistemas NOSQL:
comenzando con los sistemas basados en documentos en la Sección 24.3, seguido de clave-valor
almacena en la Sección 24.4, luego los sistemas basados en columnas en la Sección 24.5, y finalmente
sistemas basados en gráficos en la Sección 24.6. También notamos que algunos sistemas NOSQL
Puede que no caiga claramente en una sola categoría, sino que utilice técnicas que abarquen dos
o más categorías.

Preguntas de revisión

24.1. ¿Para qué tipos de aplicaciones se desarrollaron los sistemas NOSQL?

24.2. ¿Cuáles son las principales categorías de sistemas NOSQL? Enumere algunos de los NOSQL sistemas en cada categoría.

24.3. ¿Cuáles son las principales características de los sistemas NOSQL en las áreas relacionadas con modelos de datos y lenguajes de consulta?

24,4. ¿Cuáles son las principales características de los sistemas NOSQL en las áreas relacionadas con sistemas distribuidos y bases de datos distribuidas?

24,5. ¿Qué es el teorema CAP? ¿Cuál de las tres propiedades (consistencia, disponibilidad, tolerancia de partición) son las más importantes en los sistemas NOSQL?

Página 28

24,6. ¿Cuáles son las similitudes y diferencias entre el uso de la coherencia en CAP versus usar consistencia en ACID?

24,7. ¿Cuáles son los conceptos de modelado de datos que se utilizan en MongoDB? Cuales son los principales ¿Operaciones CRUD de MongoDB?

24,8. Analice cómo se realizan la replicación y la fragmentación en MongoDB.

24,9. Analice los conceptos de modelado de datos en DynamoDB.

24.10. Describir el esquema hash coherente para la distribución, replicación y y fragmentación. ¿Cómo se manejan la consistencia y el control de versiones en Voldemort?

24.11. ¿Cuáles son los conceptos de modelado de datos utilizados en sistemas NOSQL basados en columnas? tems y Hbase?

24.12. ¿Cuáles son las principales operaciones CRUD en Hbase?

24.13. Analice los métodos de almacenamiento y sistemas distribuidos que se utilizan en Hbase.

24.14. ¿Cuáles son los conceptos de modelado de datos utilizados en NOSQL orientado a gráficos? sistema Neo4j?

24.15. ¿Cuál es el lenguaje de consulta de Neo4j?

24.16. Analice las interfaces y las características de los sistemas distribuidos de Neo4j.

Bibliografía seleccionada

El documento original que describía el sistema de almacenamiento distribuido Google BigTable es Chang et al. (2006) y el artículo original que describía Amazon Dynamo
El sistema de almacenamiento de valores clave es DeCandia et al. (2007). Hay numerosos artículos que

comparar varios sistemas NOSQL con SQl (sistemas relacionales); por ejemplo,
Parker y col. (2013). Otros artículos comparan los sistemas NOSQL con otros sistemas NOSQL.
tems; por ejemplo Cattell (2010), Hecht y Jablonski (2011) y Abramova y
Bernardino (2013).

La documentación, los manuales de usuario y los tutoriales de muchos sistemas NOSQL se pueden que se encuentran en la Web. Aquí están algunos ejemplos:

Tutoriales de MongoDB: docs.mongodb.org/manual/tutorial/

Manual de MongoDB: docs.mongodb.org/manual/

Documentación de Voldemort: docs.project-voldemort.com/voldemort/

Sitio web de Cassandra: cassandra.apache.org

Sitio web de Hbase: hbase.apache.org

Documentación de Neo4j: neo4j.com/docs/

Además, numerosos sitios web clasifican los sistemas NOSQL en sub-
categorías basadas en el propósito; nosql-database.org es un ejemplo de un sitio de este tipo.