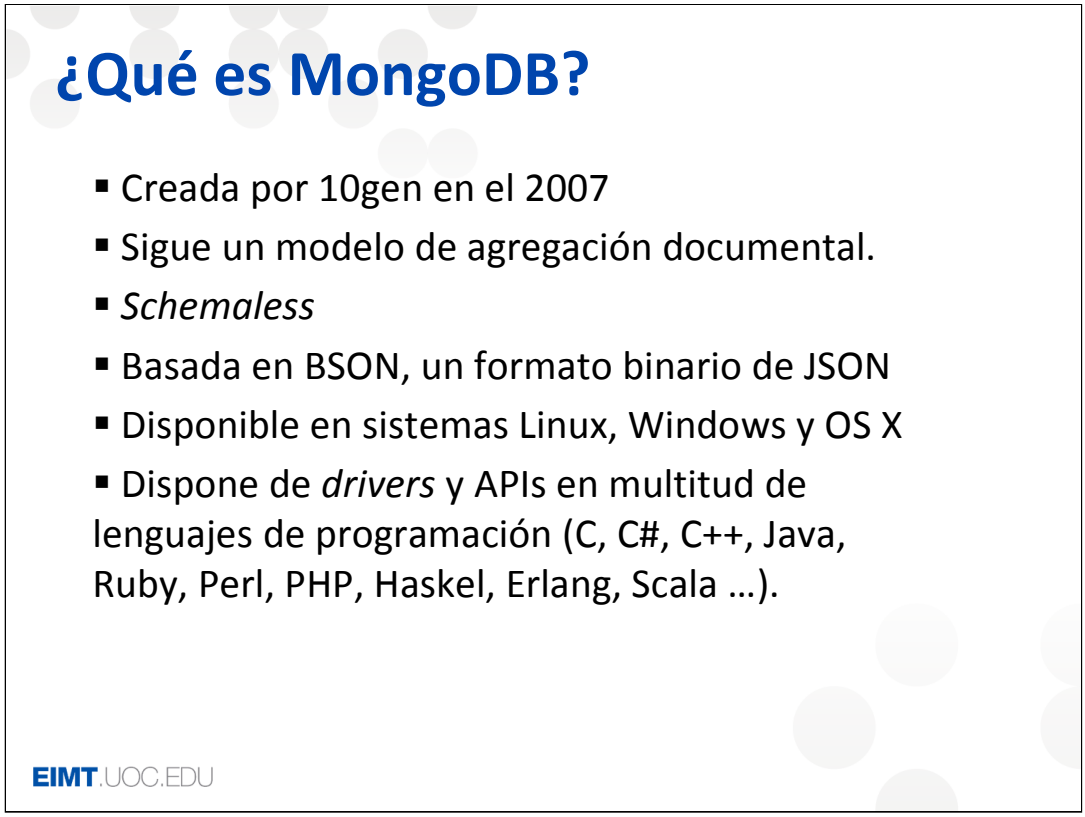
Bienvenidos. En esta presentación vamos a explicar brevemente una de las bases de datos NoSQL documentales más conocidas a día de hoy: MongoDB.

Índice

- Introducción y características
- Modelo de datos
- Operaciones CRUD
- Estrategias de distribución
- Recursos y enlaces

EIMT.UOC.EDU

En la primera parte de la presentación comentaremos qué es MongoDB y explicaremos sus principales características. Posteriormente veremos el modelo de datos que utiliza MongoDB (el documental). En la tercera parte de la presentación explicaremos cómo interactuar con MongoDB para añadir, consultar, actualizar y borrar datos (es decir, documentos) en la base de datos. Finalmente, veremos las posibilidades de fragmentación, replicación y distribución que ofrece esta base de datos y mostraremos un conjunto de enlaces a recursos relativos a MongoDB.



MongoDB es una base de datos documental creada en 2007 por la empresa 10gen, ahora MongoDB Inc. MongoDB fue creada como un componente para un sistema PaS (*Platform as a Service*). En 2009 la empresa decidió evolucionar la base de datos a código abierto, liberando MongoDB y ofreciendo soporte comercial para la misma. Desde ese momento se ha convertido en una de las bases de datos NoSQL más populares. De hecho, actualmente es la 5ª base de datos más popular (de 221) según el *ranking* de db-engines.com (http://db-engines.com/en/ranking) de 2014, siendo la base de datos NoSQL más popular según dicho *ranking*.

MongoDB sigue un modelo de datos de agregación orientado a documentos. Como tal, la unidad básica de almacenamiento son los documentos. Es una base de datos *schemaless*, y por lo tanto, no es necesario crear a priori el esquema de datos que siguen los documentos que se van a almacenar.

En MongoDB los documentos se almacenan en BSON, que es una estructura JSON en formato binario.

Es una base de datos creada en C++ que está disponible en multitud de sistemas operativos: Linux, Windows, OS X y Solaris.

Asimismo MongoDB proporciona APIs y *drivers* para gran cantidad de lenguajes de programación actuales, facilitando su uso sea cual sea el lenguaje que utilice el programa cliente que desea acceder a los datos.

Características de MongoDB

- Modelo de datos basado en documentos:
  - *Schemaless*
  - Dispone de diversos tipos de índices.
- Se puede consultar vía consola o vía API.
- Escrituras atómicas (a nivel de documento) y lecturas consistentes:
  - *Strong* o *eventual consistency* en función de la configuración del sistema
  - No soporta transacciones.
- *Auto-sharding* y replicación *Master-Slave* asíncrona
- Soporta MapReduce.

EIMT.UOC.EDU

La unidad básica de almacenamiento en MongoDB son los documentos, una estructura de datos en un formato adaptado de JSON, con soporte para los tipos de datos básicos y un esquema de datos flexible (*schemaless*). Proporciona varios tipos de índices que permiten indexar, aparte de la información más común, texto y coordenadas geográficas.

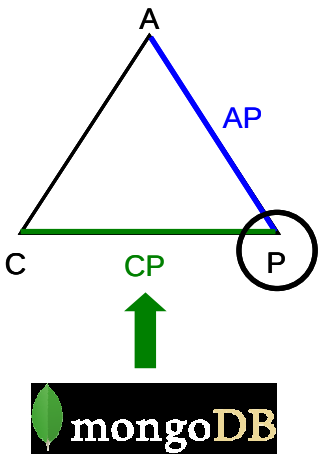
Los datos se pueden consultar bien utilizando la consola o a través de un programa utilizando una de las múltiples APIs que proporciona.

Las escrituras de los documentos se realizan de forma atómica en la base de datos para cada documento. No existen transacciones (en el sentido que este término tiene en bases de datos relacionales), por lo tanto, es imposible definir una transacción que incluya diferentes operaciones. La replicación sigue una estrategia *master-slave*. Los cambios se efectúan contra una réplica primaria (que está en el nodo *master*) y se transfieren de forma asíncrona a las réplicas secundarias (que están en los nodos *slave*). La consistencia puede ser fuerte (*strong consistency*) o final en el tiempo (*eventual consistency*), en función de si las consultas se efectúan siempre contra la copia primaria o si se permite acceder a las secundarias. Recordemos que la *strong consistency* garantiza que las aplicaciones (y en consecuencia los usuarios) tengan una visión consistente de la base de datos (bajo su punto de vista todas las réplicas tienen los mismos valores, aunque físicamente, en la base de datos, esto no sea así). Por su parte cuando se provee *eventual consistency* la falta de concordancia en los valores de unas mismas réplicas puede ser visible para las aplicaciones y usuarios (es decir, las inconsistencias pueden ser visibles).

Si se configura convenientemente, la base de datos permite distribuir los documentos entre diferentes nodos de forma automática. Esto se conoce como *auto-sharding*.

MongoDB también facilita las operaciones de agregación sobre los datos vía MapReduce u otros mecanismos.

Enfocado a mantener la consistencia y tolerancia a particiones



Por defecto, el sistema contiene una visión consistente de los datos, aunque no esté totalmente disponible en presencia de particiones.

EIMT.UOC.EDU

Respecto a las tres propiedades del teorema CAP (consistencia, disponibilidad y tolerancia a particiones), MongoDB está enfocado a mantener la consistencia de las réplicas de unos mismos datos y la tolerancia a particiones. Esto quiere decir que en algunas situaciones el sistema puede quedar parcialmente inoperativo en caso de particiones en la red.

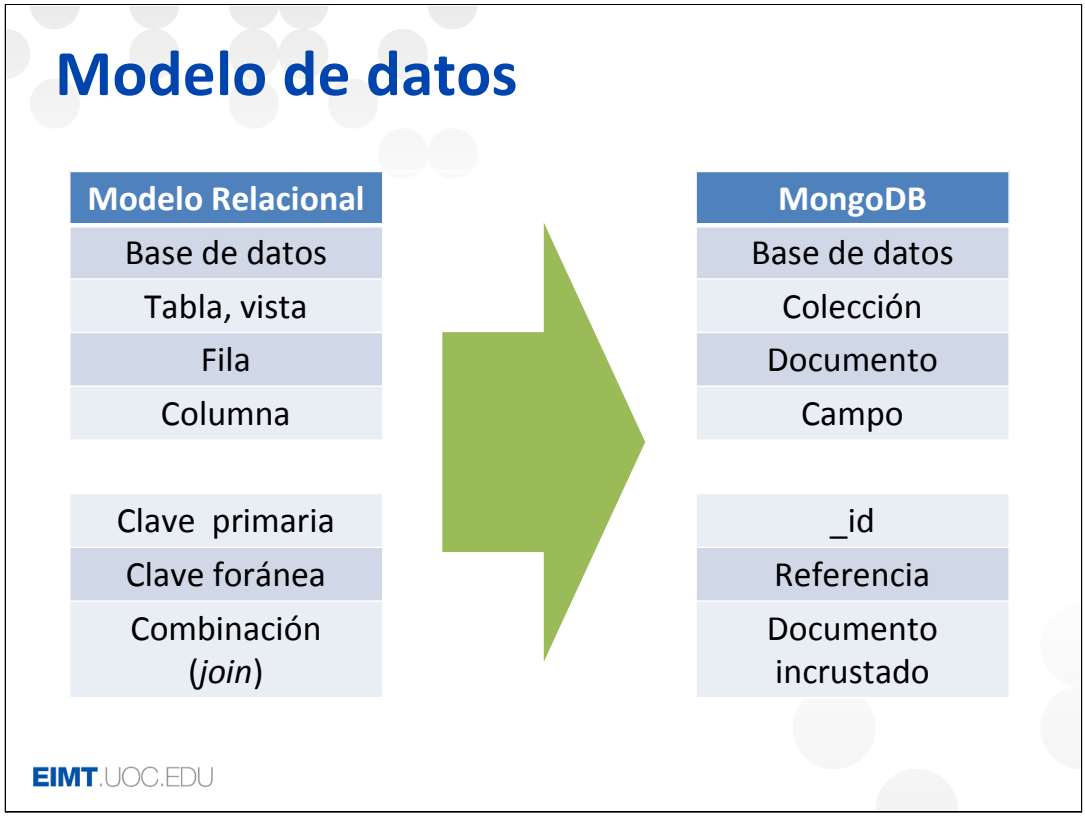
No obstante, como apuntaremos más adelante, puede configurarse para que se comporte de distinta forma, promocionando la disponibilidad a costa de la consistencia.

Índice

- Introducción y características
- Modelo de datos
- Operaciones CRUD
- Estrategias de distribución
- Recursos y enlaces

EIMT.UOC.EDU

Una vez introducida la base de datos MongoDB, vamos a ver más en detalle su modelo de datos, en qué consiste el formato BSON, los índices que incorpora y las equivalencias entre su modelo de datos y el modelo relacional.



Para explicar el modelo de datos de MongoDB utilizaremos el modelo relacional como base.

En MongoDB los datos se almacenan en bases de datos y, dentro de estas, se organizan en colecciones. Las colecciones son el equivalente a las tablas del modelo relacional, y se utilizan para agrupar elementos con semántica o características comunes como, por ejemplo, los pedidos de venta de una empresa. Una colección contiene un conjunto de documentos. Cada documento define un conjunto de datos para un individuo de la colección. En el caso de ejemplo, cada documento equivaldría a un pedido de venta.

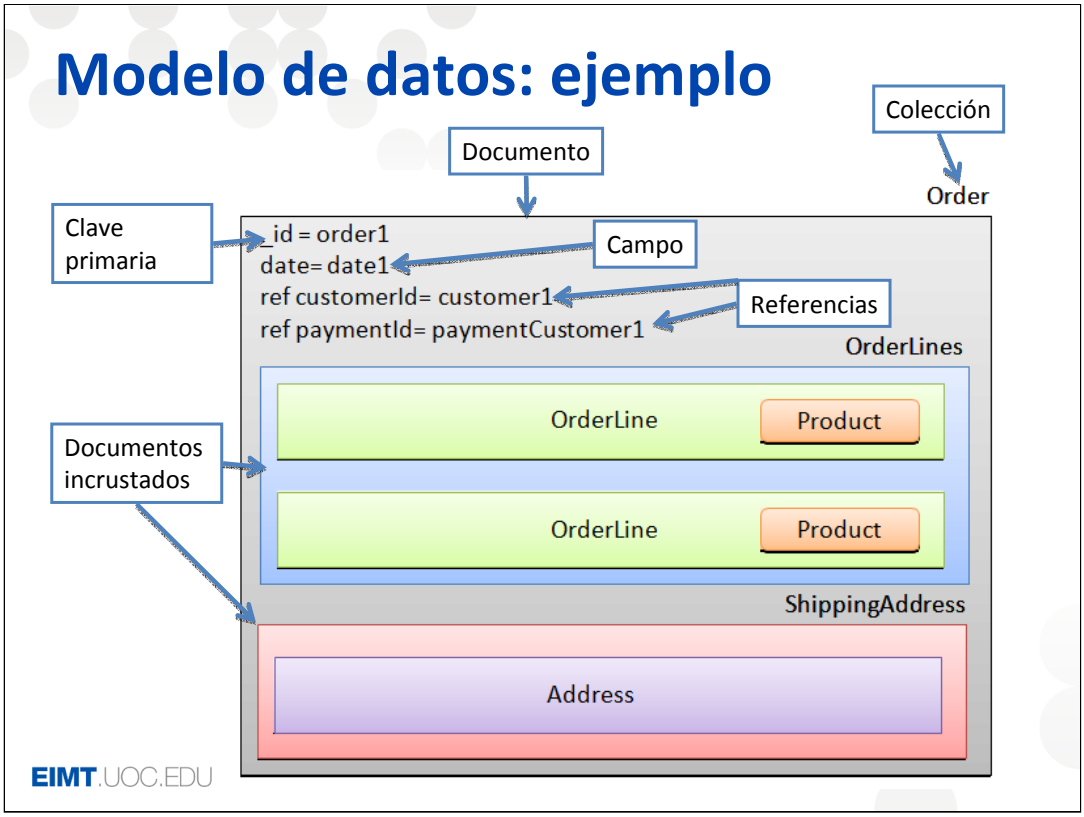
Los documentos se almacenan en formato BSON y contienen un conjunto de campos (es decir, atributos o propiedades, o columnas en términos de una base de datos relacional). Cada campo consta de un nombre de campo y un valor asociado. Los documentos no necesitan seguir ningún tipo de esquema predeterminado. Es decir, distintos documentos de una misma colección pueden tener estructuras y campos diferentes.

Todo documento tiene una campo que lo identifica unívocamente de los demás documentos de la colección (lo que conocemos como clave primaria en el modelo relacional). Esta clave primaria se representa mediante el campo *_id*. Su valor puede ser informado por el usuario o, en caso contrario, será asignado por la base de datos de forma automática.

MongoDB permite utilizar campos en los documentos que sean referencias a otros documentos, sería el equivalente a las claves foráneas del modelo relacional.

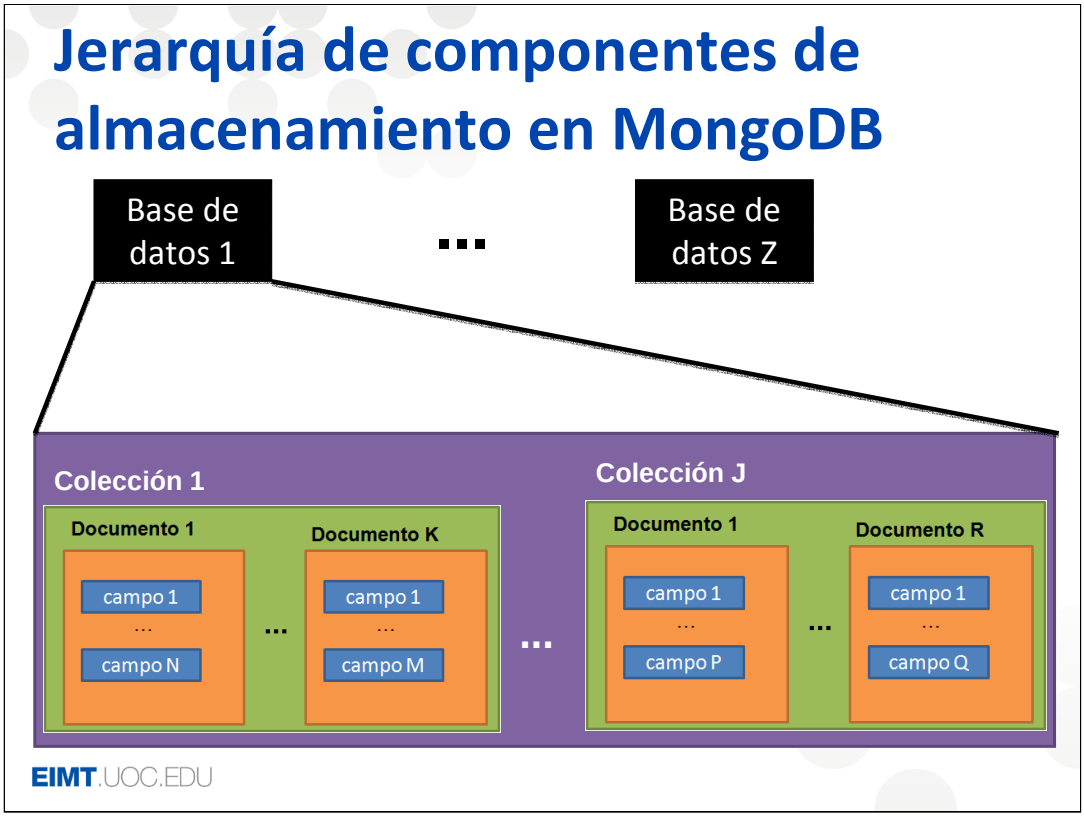
MongoDB también permite definir (o si se prefiere incrustar) (sub)documentos dentro de un documento (*embedded subdocument*). Estos (sub)documentos son conjuntos de datos relacionados con el elemento a representar. Como consecuencia, los documentos que incorporan (sub)documentos pueden ser vistos como documentos (u objetos) compuestos, de tal manera que estamos aplicando técnicas de desnormalización de los datos que permiten evitar la ejecución de operaciones de *join* del álgebra relacional.

Vamos a ver con un ejemplo los distintos elementos de MongoDB que acabamos de describir.



En esta transparencia vemos un ejemplo de documento que representa un pedido de venta. Los diferentes pedidos se almacenarían en una colección denominada *Order*. Cada pedido estará representado por un documento. El documento de ejemplo contiene una clave primaria (*_id*) con valor *order1*, un campo que permite indicar la fecha en que se efectúa el pedido y un par de referencias que apuntan a los documentos que contienen información del cliente que realizó el pedido y de los datos del pago. Estas referencias podrían haberse definido de forma distinta, ya que podría haber incluido toda la información del cliente y de los datos de pago dentro del documento actual. No obstante, por decisiones de diseño se ha preferido utilizar referencias y gestionar los datos de clientes y de pago mediante documentos (y en consecuencia, colecciones) aparte.

En el pedido también podemos ver ejemplos de (sub)documentos incrustados, como por ejemplo, las líneas del pedido y la dirección de envío del pedido.



Una vez visto el modelo de datos de MongoDB, vamos a ver cómo se estructuran los componentes de almacenamiento.

Los componentes de almacenamiento de MongoDB se estructuran de forma jerárquica tal y como se indica en la figura. Una instanciación de MongoDB puede tener diferentes bases de datos. Cada base de datos, a su vez, puede contener distintas colecciones. Cada colección puede contener documentos, que a su vez, pueden contener distintos números de campos (es decir, pueden tener diferente estructura). Recordemos que los documentos de cada colección comparten semántica, es decir, se corresponden a objetos de un mismo tipo del mundo real.

Índices

- MongoDB utiliza distintos tipos de índices para optimizar el acceso a datos.
- Los índices se definen a nivel de colección.
- Por defecto siempre se crea un índice sobre la clave primaria.
- Tipos de índice disponibles:
  - Sobre un campo
  - Sobre un conjunto de campos
  - Sobre datos espaciales
  - Indexación de texto
- Los índices pueden ser únicos y *sparse*.

EIMT.UOC.EDU

MongoDB utiliza distintos índices para optimizar el acceso a datos y evitar tener que consultar todos los documentos de una colección para satisfacer una consulta.

Los índices se definen a nivel de colección. Por defecto, para cada colección, siempre se crea un índice único sobre su clave primaria, es decir, sobre el campo *_id*. Eso evita que se puedan dar de alta en una colección dos documentos con el mismo identificador.

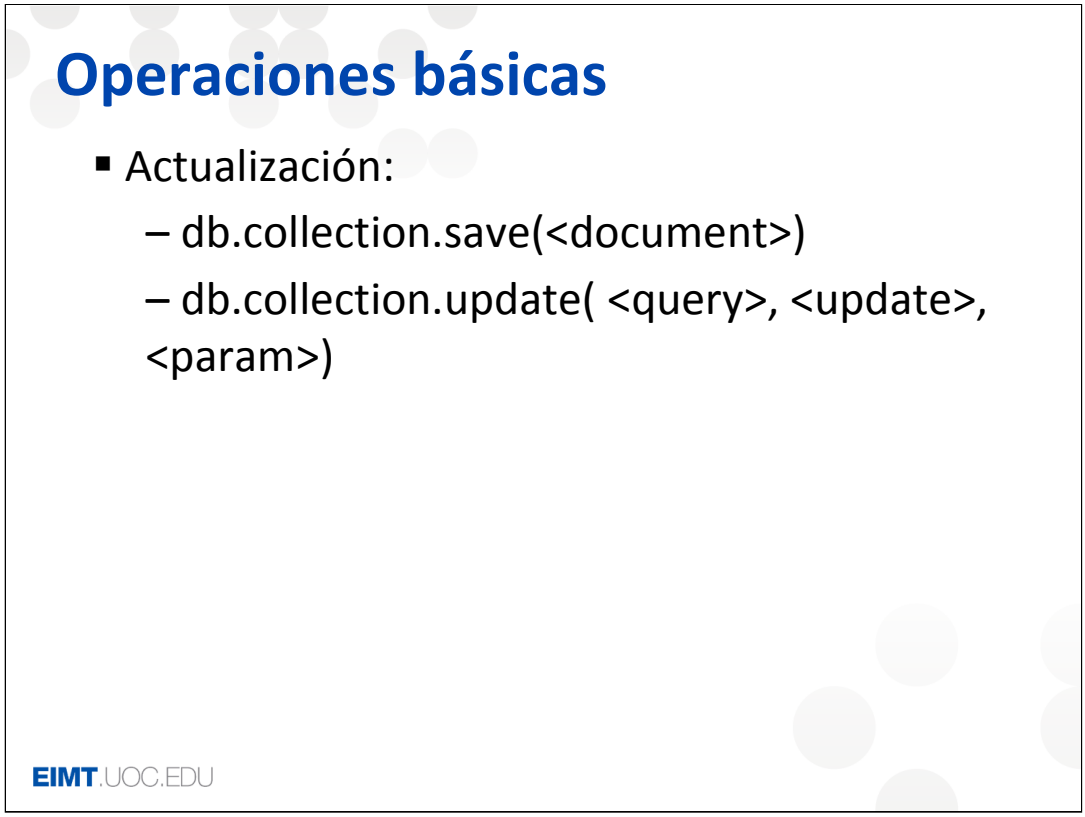
MongoDB contiene una gran variedad de índices que pueden clasificarse según el número de campos indexados (sobre un campo, sobre un conjunto de campos o sobre un campo que es una lista de valores —un array o (sub)documento incrustado—) y según el tipo de datos indexado (índices espaciales o de texto). Los índices creados pueden ser únicos y *sparse*. Los índices únicos no permiten valores duplicados. Los índices *sparse* no almacenan entradas en el índice para los documentos que no tienen valores para los campos indexados. Este tipo de índice es conveniente debido a la falta de esquema de los documentos.

Índice

- Introducción y características
- Modelo de datos
- Operaciones CRUD
- Estrategias de distribución
- Recursos y enlaces

EIMT.UOC.EDU

Hasta aquí la introducción a MongoDB y a su modelo de datos. A continuación vamos a presentar algunas de las operaciones de creación, lectura, actualización y borrado de datos en MongoDB.



Las operaciones de lectura y escritura de MongoDB se ejecutan siempre en el ámbito de una colección y permiten crear, modificar, consultar y borrar documentos. Recordad que en MongoDB las operaciones se realizan siempre a nivel de documento.

En esta transparencia podemos ver una simplificación de la sintaxis propuesta para la consola de MongoDB. Las llamadas a las API en algunos lenguajes pueden diferir ligeramente de esta sintaxis.

Empezaremos hablando de las operaciones de actualización, que pueden ejecutarse mediante llamadas a las funciones *save* y *update*. La función *save* actualiza el documento pasado por parámetro. Para que pueda realizarse dicha actualización el documento deberá incluir un campo *_id*. En el caso de que el documento no incluya este campo, MongoDB realizará una inserción del documento en la colección. Esta doble funcionalidad (actualizar un documento cuando existe y sino crearlo) es lo que se denomina *upsert* en MongoDB.

La otra operación para actualizar los datos de los documentos es la operación *update*. Esta operación identifica los documentos a modificar utilizando la condición indicada en el parámetro *query*. Posteriormente, actualiza dichos documentos en función del parámetro *update*. Este parámetro permite indicar un documento (en caso de que queramos cambiar un documento por otro) o información relativa a sus campos (en caso de que sólo queramos cambiar algunos campos del documento).

Antes de continuar con las siguientes operaciones creemos que es interesante hablar de cómo gestiona MongoDB las transacciones y la concurrencia.

Operaciones básicas

- ¿Cómo funcionan las operaciones de escritura?
  - Transacciones a nivel de documento
  - Gestión de concurrencia:
    - Reservas S, X
    - Bloqueo a nivel de base de datos y diferentes estrategias de escritura

EIMT.UOC.EDU

Las transacciones en MongoDB incorporan una única operación (y esta operación actúa sobre documentos de una misma colección). Si queremos crear una transacción que incorpore múltiples operaciones deberemos gestionarlas desde nuestros programas.

Respecto a la gestión de concurrencia, MongoDB utiliza el típico sistema pesimista de bloqueos (o reservas) S, X, donde las operaciones de escritura bloquean los datos hasta su finalización para asegurar la consistencia. Según la documentación, el bloqueo se realiza a nivel de instancia de base de datos, eso quiere decir que si realizamos una operación de actualización sobre un documento, el resto de documentos y colecciones de la base de datos quedaran inaccesibles para otros usuarios hasta que la operación finalice. Eso podría suponer un problema en caso de que existieran transacciones complejas y que necesitaran un gran tiempo de proceso. No obstante, según la documentación de MongoDB, sus transacciones son muy cortas en el tiempo y el bloqueo a nivel de base de datos no debería suponer un problema.

Respecto a las operaciones de actualización, borrado e inserción también es importante destacar que MongoDB permite que el usuario determine cómo desea que se ejecute su operación: no requerir confirmación de que la escritura se ha realizado –*unacknowledged*–, esperar respuesta de que la escritura se ha realizado correctamente – *acknowledged*–, esperar que la operación se registre en memoria y en el dietario de la base de datos antes de recibir la confirmación de lectura –*ournaled*– o esperar a que la operación se realice sobre las réplicas antes de recibir la confirmación –*replica aknowledged*–. Por defecto la forma de trabajo es *acknowledged*.

Operaciones básicas

- Creación:
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, {upsert:true})
- Borrado:
  - db.collection.remove(<query>, <just one>)
- Consulta:
  - db.collection.query(<query>, <projection>)

EIMT.UOC.EDU

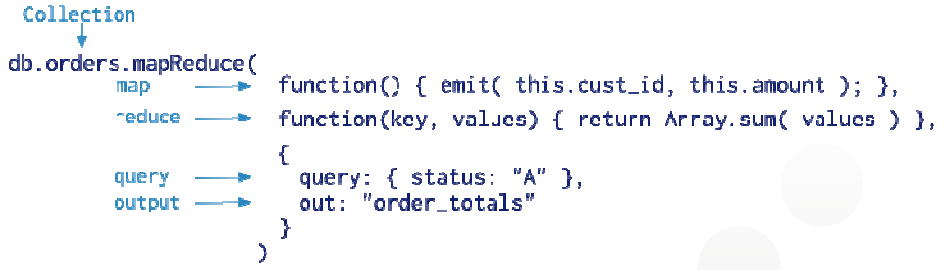
La creación de nuevos documentos en una colección puede hacerse mediante la operación *insert*, cuyo parámetro indicará el documento a insertar. Este documento puede contener un valor para el campo *_id* o no. Alternativamente, se pueden añadir documentos vía las operaciones *save* y *update*. La operación *save* realizará una inserción cuando el documento pasado por parámetro no tenga definido el campo *_id* o bien cuando lo haya pasado pero no exista dicho documento en la colección. La operación *update* podrá realizar una operación de inserción cuando se le añada el parámetro *upsert:true*, tal y como ya hemos comentado anteriormente.

Para borrar documentos de una colección se utiliza la operación *remove*. Esta operación puede recibir como parámetro un conjunto de condiciones (*query*) que permita encontrar los documentos a eliminar y un parámetro para indicar si se debe eliminar sólo un documento o si se pueden eliminar múltiples documentos (opción por defecto). Asímismo, la ejecución de la operación *remove* sin parámetros, causará la eliminación de todos los documentos de la colección.

Para consultar los documentos de una colección se utiliza la operación *query*, cuyos parámetros más importantes permiten definir las condiciones de búsqueda a partir del parámetro *query* y qué campos del documento se deben devolver a partir del parámetro *projection*. El parámetro *query* es el equivalente a la cláusula *WHERE* de SQL y el parámetro *projection* es el equivalente a la cláusula *SELECT* de SQL.

Otras operaciones

- Gestión de índices
- Agregación:
  - db.collection.count()
  - db.collection.distinct()
  - db.collection.group()
- MapReduce:



EIMT.UOC.EDU

Además de las operaciones básicas que acabamos de ver, hay otras operaciones que se pueden ejecutar sobre las colecciones en MongoDB. Algunas de ellas son las relativas a la creación y la gestión de índices, la agregación de datos y la definición de tareas MapReduce.

Las operaciones de agregación básicas son *count,* que cuenta el número de documentos de la colección o de una determinada consulta, *distinct,* que evita devolver valores duplicados y *group,* que agrupa los datos de una colección —o de una consulta— en función de un conjunto de campos (el equivalente a la cláusula *GROUP BY* de SQL). Según la configuración de MongoDB la operación *group* puede ser ejecutada utilizando el *framework* MapReduce o mediante otros mecanismos.

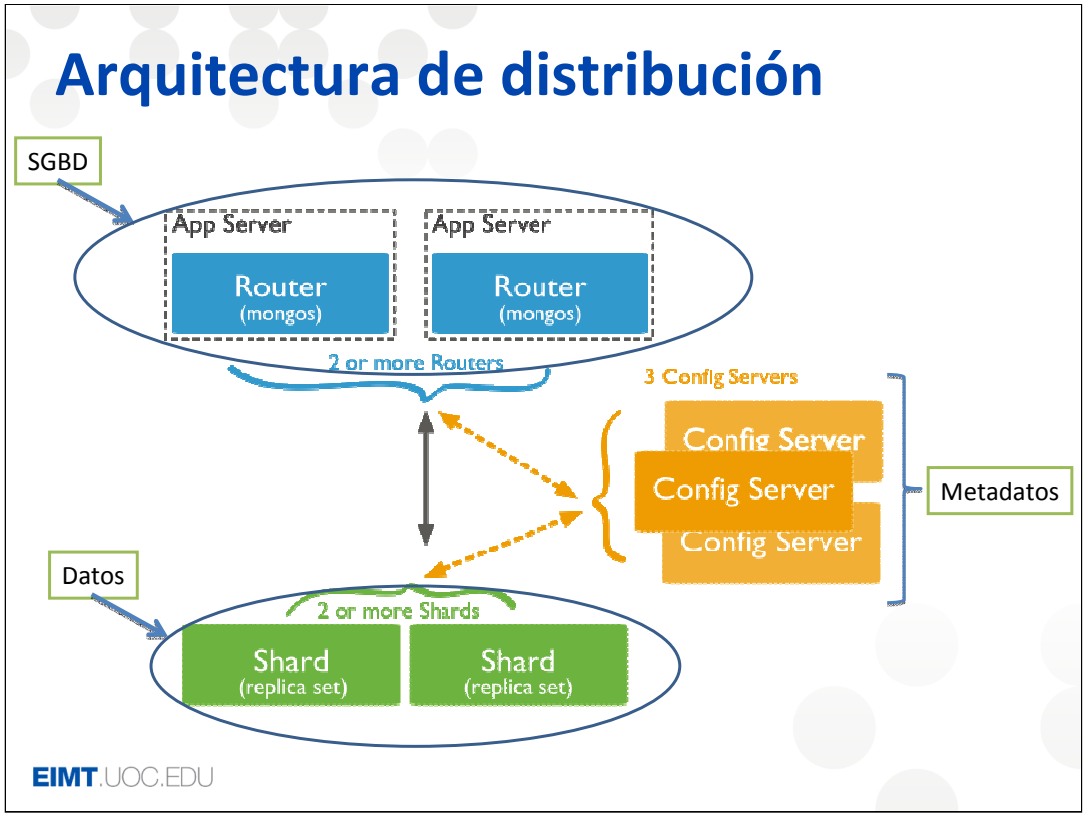
Finalmente, MongoDB permite utilizar el *framework* MapReduce para resolver un determinado problema, a través de la operación *mapReduce*. Esta operación es compleja y puede variar en diferentes versiones de MongoDB, por lo tanto, no entraremos en detalle en su sintaxis. Simplemente comentar que la operación de ejemplo mostrada en la transparencia se ejecuta sobre la colección de pedidos y permite calcular el número de pedidos totales con estatus “A” para cada cliente.

Índice

- Introducción y características
- Modelo de datos
- Operaciones CRUD
- Estrategias de distribución
- Recursos y enlaces

EIMT.UOC.EDU

Hasta aquí hemos visto las características y funcionamiento de MongoDB. En este apartado vamos a estudiar cómo funciona su mecanismo de fragmentación y replicación de datos.

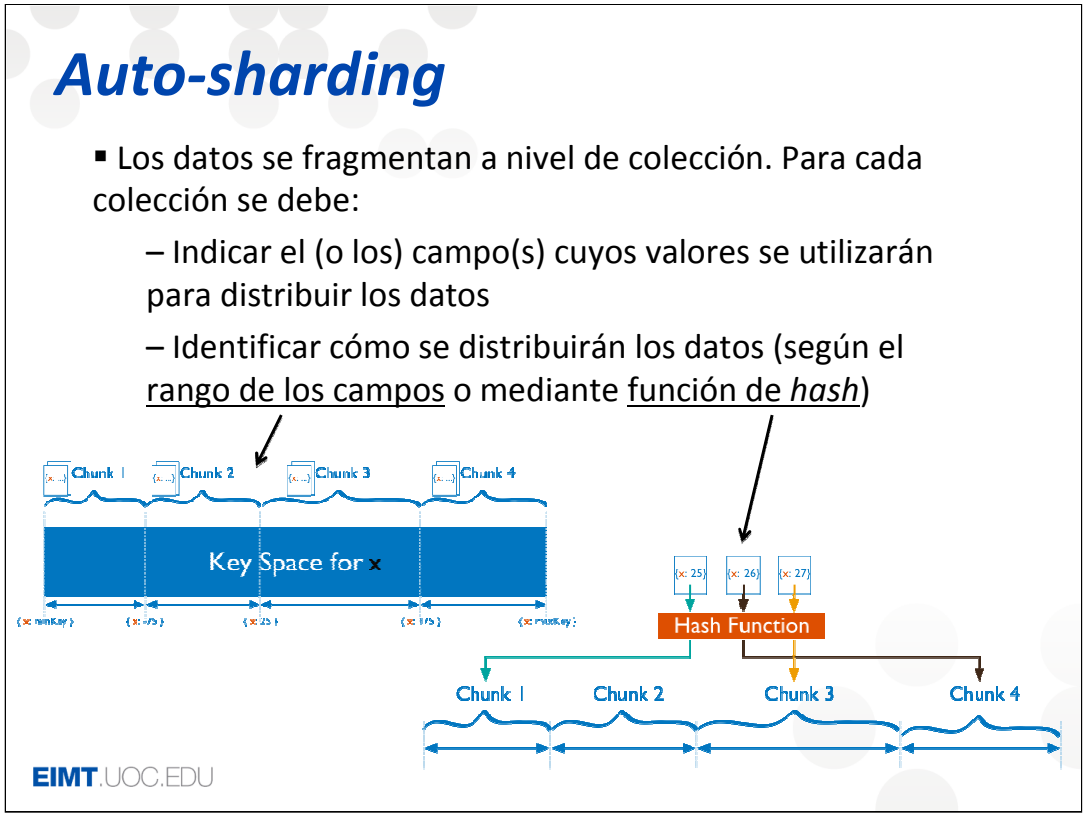


MongoDB utiliza *sharding* (fragmentación horizontal) para distribuir los datos de la base de datos en distintos nodos. En esta transparencia podemos ver los distintos componentes que utiliza MongoDB para gestionar la fragmentación de datos.

Los *routers* o servidores de aplicaciones son las instancias de la base de datos y por tanto son los puntos de entrada de los usuarios y de los programas a la base de datos. Son los encargados de recibir las peticiones de los usuarios y programas y resolverlas.

Los datos de la base de datos se distribuyen en *shards* o fragmentos que podemos distribuir en distintos nodos.

Los servidores de configuración contienen metadatos que los *routers* utilizan para identificar en qué *shard* está la información requerida y, en consecuencia, son necesarios para que las consultas se puedan distribuir eficientemente entre los distintos nodos de la base de datos y evitar así hacer peticiones de datos a nodos que no contengan esos datos.



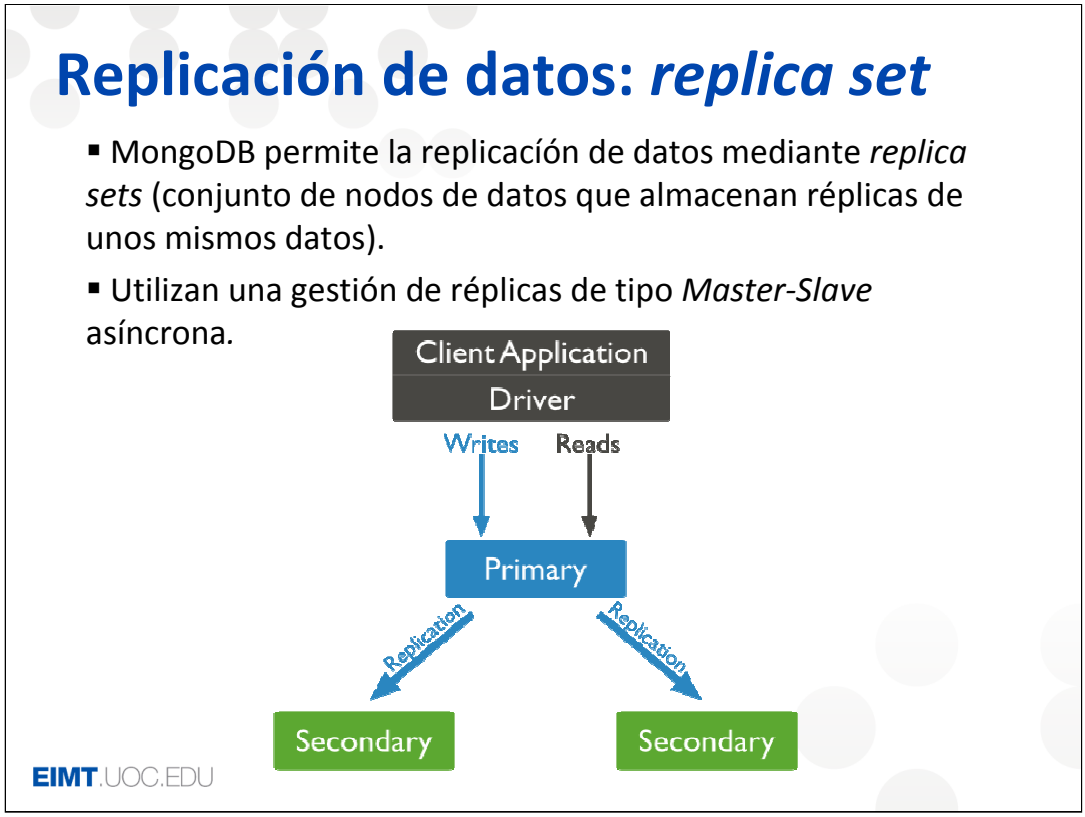
MongoDB permite distribuir los datos entre los distintos nodos de la base de datos de forma automática. Los datos que se utilizan para realizar esta distribución de datos y la forma en que se distribuyen puede ser configurada por el usuario.

La fragmentación en MongoDB se realiza a nivel de colección. Para cada colección se debe informar de qué campo, o conjunto de campos, se tendrán en cuenta para distribuir los datos. Aparte, se deberá informar de cómo se debe realizar la distribución. Existen dos opciones: mediante una función de *hash* y según los rangos de valores de los campos seleccionados. Una vez identificada esta información, debe crearse una clave de distribución (denominada *shard key*) sobre los campos elegidos para definir la función de *shard*. La función de *shard* deberá distribuir los valores en un número predefinido de *chunks* (o montones). Un *chunk* es un conjunto de valores que pertenece al mismo grupo y determinará al *shard* donde se enviarán los datos.

En la figura podemos ver las dos opciones mediante dos ejemplos. En ambos ejemplos se ha utilizado un campo numérico para la distribución de datos. En el primer caso, la fragmentación se realiza en función del valor de los atributos. Por tanto, se crean un conjunto de rangos en función de los posibles valores del campo {(-infinito, -75), [-75,25), [25,175), [175,infinito)}. Nótese que los valores cercanos caerán en el mismo *chunk*, mientras que valores lejanos es más probable que caigan en *chunks* distintos, y por lo tanto, nodos distintos. Eso puede agilizar las consultas por rangos, pero puede comportar distribuciones poco equilibradas en otros.

En el ejemplo de la derecha podemos ver la distribución de datos mediante una función de *hash*. En este caso, documentos con valores cercanos del campo a utilizar se asignarían en distintos nodos, consiguiendo, en el caso general, un reparto más equilibrado de los datos entre los distintos nodos. A pesar de ello, podrían existir problemas de rendimiento en algunos casos donde hubiera muchas consultas secuenciales o por rangos.

En el caso de utilizar MongoDB es importante estudiar atentamente sus manuales para hacer una buena distribución de datos y mantener una carga de datos adecuada. Hay distintas opciones para controlar dichos procesos en MongoDB, pero explicarlos en detalle queda fuera del alcance de esta presentación.



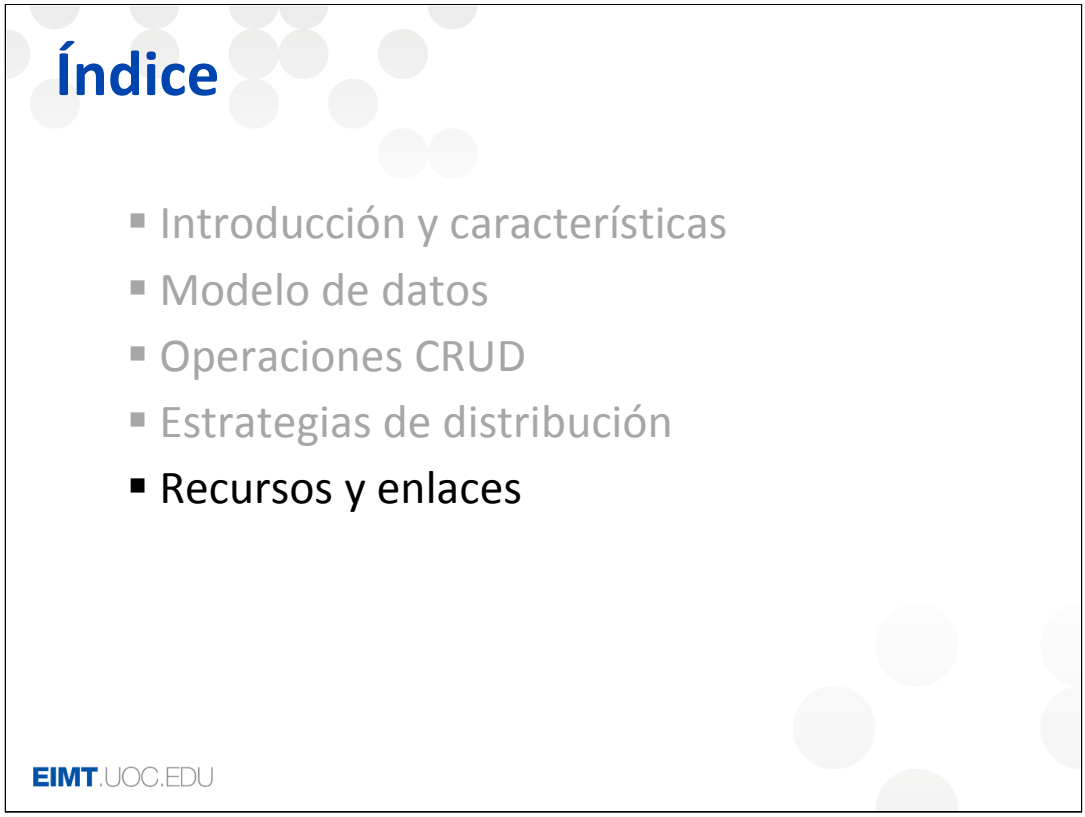
MongoDB permite la replica de datos para garantizar redundancia, proveer de copias de seguridad y garantizar la recuperación automática en caso de caídas de nodos. En MongoDB los nodos que comparten los mismos datos se denominan *replica set*. En la figura podemos ver un *replica set* con tres nodos, eso quiere decir que el sistema tiene 3 copias (o réplicas) de unos mismos datos.

Por defecto, las réplicas en MongoDB siguen una política de gestión de replicas *máster-slave* asíncrona, donde:

• Existe un solo maestro (o replica primaria) por cada *replica set*.
• Las operaciones de lectura y escritura se realizan sólo sobre la copia primaria.
• Los datos de las copias (o réplicas) secundarias se sincronizan a partir de las operaciones realizadas en la copia primaria de forma asíncrona.
• En caso de que el nodo que almacena la copia primaria caiga, se promociona una copia secundaria (o *slave*) como nueva copia primaria.
• Opcionalmente, pueden añadirse nodos de tipo "Arbitro" a un *replica set*. Estos árbitros permitirán desempatar las votaciones que puedan surgir al escoger nuevas copias primarias.
• Se pueden definir prioridades sobre las distintas copias secundarias para incrementar (o decrementar) su probabilidad de ser escogidas como nodo primario.

Las anteriores características hacen que las réplicas permitan a MongoDB incrementar la disponibilidad de los datos. Según la documentación de MongoDB, su uso con la configuración por defecto garantiza la consistencia estricta. Esto, desde un punto de vista teórico no es del todo exacto. Realmente garantiza *strong consistency*, es decir, las aplicaciones tienen siempre una visión consistente de los datos, aunque a nivel físico, mientras se propagan los cambios de la copia primaria a las secundarias, no todas las réplicas de unos mismos datos contienen los mismos valores.

Pero MongoDB también permite que se realicen consultas directas sobre los nodos secundarios, siempre y cuando se configure adecuadamente. Como la sincronización entre el nodo primario y los secundarios se realiza de forma asíncrona, realizar consultas sobre nodos secundarios puede implicar leer datos obsoletos y por tanto rebajaría la consistencia a consistencia final en el tiempo (*eventual consistency*).



Y hasta aquí esta introducción a MongoDB. En esta presentación hemos presentado las principales características de esta base de datos, su modelo de datos, las operaciones que permiten añadir, borrar, modificar y consultar datos en bases de datos y cómo se realiza el *sharding* y la replicación de datos. La presentación tiene por objetivo ser eminentemente introductoria. Para cualquier desarrollo real en MongoDB aconsejamos consultar su manual de referencia para obtener información actualizada de sus funcionalidades, operaciones y arquitectura.

Ya para finalizar presentaremos un conjunto de enlaces y referencias de interés.

Recursos y enlaces

- Página oficial de MongoDB:
  http://www.mongodb.org
- Documentación oficial: http://docs.mongodb.org/manual/
- Listado de *drivers*:
  http://docs.mongodb.org/ecosystem/drivers/
- MongoDB en StackOverflow:
  http://stackoverflow.com/questions/tagged/mongodb
- Grupo de usuarios de MongoDB-Spain:
  http://www.meetup.com/mongodb-spain/

EIMT.UOC.EDU

A continuación podemos encontrar un conjunto de enlaces de interés sobre MongoDB.

Referencias

R. Copeland (2013). *MongoDB Applied Design Patterns*. O'Really Media.

E. Redmond, J. Wilson (2012). *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement.* Pragmatic Bookshelf.

K. Chodorow (2011). *50 Tips for MongoDB Developers* O'Really Media.

K. Banker (2011). *MongoDB in Action*. Manning.

P.J. Sadalage & M. Fowler. (2013). *NoSQL Distilled. A brief Guide to the Emerging World of Polyglot Persistence*, Pearson Education.

EIMT.UOC.EDU

Y hasta aquí esta presentación dedicada a MongoDB. Esperamos que os haya sido de ayuda.

Que tengáis un buen día.