MongoDB®
UNTUK INDONESIA

Memahami Konsep dan Implementasi MongoDB

WAHYU KRISTIANTO

Daftar Isi

Pengantar

Selamat datang di buku "MongoDB untuk Indonesia" yang ditulis dengan tujuan untuk memberikan pemahaman yang komprehensif tentang salah satu database paling populer saat ini, yaitu MongoDB.

Penulis berharap buku ini dapat memberikan wawasan yang berharga bagi pembaca dalam membangun aplikasi modern yang memerlukan database yang handal dan fleksibel. Dalam buku ini, pembaca akan menemukan konsep dasar MongoDB, seperti cara menginstal dan mengkonfigurasi, dan juga memperoleh pemahaman mendalam tentang fitur-fitur lainnya yang ditawarkan oleh MongoDB.

Penulis dengan senang hati memberikan buku ini secara gratis sebagai kontribusi untuk komunitas pengembang di Indonesia. Penulis yakin bahwa buku ini akan membantu pembaca memahami MongoDB dan memberikan pemahaman yang dibutuhkan untuk menjadi pengembang yang handal dan terampil dalam menggunakan database yang inovatif ini.

Terima kasih telah membaca buku ini. Semoga buku ini dapat membantu pembaca memperoleh pengetahuan yang berguna tentang MongoDB dan menjadikannya referensi untuk melakukan kontribusi kepada sesama pengembang di Indonesia.

Lisensi

termasuk namun tidak terbatas pada:

- Mengklaim karya ini sebagai karya sendiri atau menghapus kredit penulis yang seharusnya diberikan.
- Menjual atau memperdagangkan karya ini tanpa izin tertulis dari penulis.
- Mengubah atau memodifikasi karya ini tanpa izin tertulis dari penulis.

Pembajakan merupakan tindakan melanggar hak cipta yang merugikan para pencipta dan pelaku industri kreatif. Oleh karena itu, kami sangat mendukung upaya untuk memerangi pembajakan dan menghargai hak cipta. Jika pembaca menemukan buku ini disebarkan secara tidak sah di internet atau tempat lainnya, silakan memberikan informasi lengkap mengenai URL dan informasi tambahan lainnya ke alamat email penulis di `w.kristories@gmail.com`.

Kami sangat menghargai bantuan pembaca dalam memerangi tindakan pembajakan dan menjaga integritas hak cipta. Terima kasih atas dukungan dan penghargaan pembaca terhadap hak cipta penulis dan industri kreatif.

Evolusi Sistem Manajemen Basis Data

Sistem Manajemen Basis Data (DBMS) merupakan perangkat lunak yang dirancang khusus untuk mengatur dan mengelola basis data (database). Seiring dengan perkembangan sistem informasi modern, DBMS telah menjadi bagian yang tidak terpisahkan dan terus berkembang pesat dalam beberapa dekade terakhir. Namun, seperti teknologi lainnya, sejarah panjang DBMS tidak dapat diabaikan.

Awalnya, database dikelola dengan menggunakan file data yang disimpan dalam file teks atau spreadsheet. Namun, metode ini seringkali menghasilkan kekacauan dan sulit diatur serta diakses oleh banyak pengguna. Kemudian, pada tahun 1960-an, Edgar F. Codd memperkenalkan model database relasional yang mengubah cara database disimpan dengan menggunakan tabel yang terdiri dari baris dan kolom. Model ini sangat memudahkan pengguna dalam mengakses dan mengelola data.

Pada tahun 1970-an, DBMS pertama kali diperkenalkan. IBM meluncurkan System R, model database relasional pertama yang berfungsi penuh pada tahun 1970. Perusahaan lain seperti Oracle, Microsoft, dan Informix juga memasuki pasar dengan solusi mereka sendiri selama dekade berikutnya.

Pada tahun 1980-an, teknologi pemrosesan transaksi online (OLTP) diperkenalkan, yang memungkinkan perusahaan untuk mengelola transaksi secara real-time, seperti transaksi penjualan dan pembelian. DBMS mulai menyediakan dukungan untuk bahasa pemrograman seperti SQL, yang memudahkan pengguna dalam memanipulasi data.

Pada tahun 1990-an, perusahaan mulai mencari cara untuk mengakses database dari jaringan yang terdistribusi secara geografis. Solusi yang ditemukan adalah sistem database terdistribusi, yang memungkinkan pengguna untuk mengakses dan memanipulasi data dari jarak jauh melalui jaringan.

Sistem Manajemen Basis Data Relasional

Sistem Manajemen Basis Data Relasional (RDBMS) merupakan jenis DBMS yang paling populer dan banyak digunakan di berbagai industri, seperti perbankan, perusahaan telekomunikasi, e-commerce, dan lainnya. RDBMS menggunakan model database relasional yang diperkenalkan pertama kali oleh Edgar F. Codd pada tahun 1960-an. Dalam RDBMS, data disimpan dalam tabel yang terdiri dari baris dan kolom, memungkinkan pengguna mengelola data secara efektif dan efisien.

Setiap tabel dalam RDBMS memiliki kunci utama yang membedakan setiap baris dan kolom dalam tabel serta memiliki relasi dengan tabel lain. Dengan model database relasional ini, pengguna dapat mengatur, mengelola, dan memanipulasi data secara mudah dan terstruktur. Data yang dimasukkan ke dalam sistem database harus melalui proses validasi dan verifikasi agar data yang dimasukkan valid dan konsisten.

RDBMS menyediakan fitur transaksi ACID (Atomicity, Consistency, Isolation, Durability) yang memastikan bahwa setiap transaksi yang terjadi pada database adalah aman dan konsisten. Fitur ini sangat penting dalam sistem database relasional dan memastikan bahwa integritas data dalam sistem database tetap terjaga dengan baik.

Atomicity merujuk pada sifat transaksi yang tidak dapat dibagi-bagi. Jika satu bagian dari transaksi gagal, maka seluruh transaksi akan dibatalkan dan tidak akan ada perubahan pada data yang terkait dengan transaksi tersebut. Consistency merujuk pada sifat transaksi yang menjaga keselarasan data dalam sistem database. Setiap transaksi harus memastikan bahwa data yang dimasukkan memenuhi kriteria yang ditetapkan sebelumnya.

Isolation merujuk pada sifat transaksi yang terisolasi satu sama lain. Setiap transaksi harus dapat berjalan secara independen dan tidak mempengaruhi transaksi lain yang sedang berlangsung dalam sistem

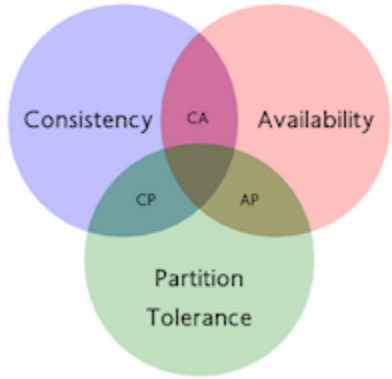
database. Durability merujuk pada sifat transaksi yang bersifat tahan lama. Setiap perubahan yang dilakukan pada data dalam sistem database harus tetap ada dan dapat diakses bahkan setelah sistem database mengalami kegagalan atau kerusakan.

Meskipun RDBMS memiliki banyak keunggulan dalam mengatur data dengan cara yang terstruktur dan memudahkan pengguna untuk mengakses dan memanipulasi data, terdapat beberapa masalah yang sering dihadapi dalam penggunaannya. Salah satunya adalah skalabilitas, yaitu kemampuan sistem database untuk menangani pertumbuhan data yang besar. Dalam konteks bisnis dan industri, pertumbuhan data dapat terjadi dengan sangat cepat, sehingga sistem database harus dapat menangani jumlah data yang besar dengan baik. RDBMS seringkali mengalami masalah dalam menangani pertumbuhan data yang cepat dan besar.

Teorema CAP

Masalah skalabilitas dalam sistem database sering menjadi perhatian utama dalam penggunaan RDBMS. Konsep fundamental dalam sistem database distribusi yang dirancang untuk menghadapi masalah skalabilitas dan toleransi kesalahan adalah Teorema CAP. Teorema ini dirumuskan oleh Eric Brewer pada tahun 2000 dan menyatakan bahwa dalam sebuah sistem database yang dirancang untuk di-scaling secara horizontal, hanya mungkin untuk memenuhi dua sifat dari tiga sifat utama, yaitu Consistency, Availability, dan Partition Tolerance.

Sifat Consistency berarti bahwa setiap operasi pada sistem harus menghasilkan hasil yang konsisten pada setiap node dalam sistem. Namun, konsistensi bertentangan dengan availability yang memastikan sistem distribusi memberikan respons terhadap permintaan klien dalam waktu yang wajar tanpa kegagalan atau error yang berulang. Sementara itu, partition tolerance memastikan bahwa sistem dapat beroperasi dengan baik meskipun terjadi pemisahan atau kegagalan pada komunikasi antar node dalam sistem distribusi.



Dalam memilih dua sifat dalam sistem distribusi, sangat tergantung pada kebutuhan dan tujuan dari sistem. Misalnya, pada sistem e-commerce yang menangani transaksi keuangan, konsistensi dan availability mungkin menjadi sifat yang paling penting, karena kesalahan atau keterlambatan dapat menyebabkan kehilangan uang dan kepercayaan pelanggan. Sebaliknya, pada aplikasi sosial media, availability dan partition tolerance mungkin lebih penting karena prioritasnya adalah memberikan layanan dengan cepat dan tidak terputus meskipun terjadi gangguan jaringan atau pemisahan antar node.

Untuk memenuhi sifat availability dalam teorema CAP, sistem distribusi harus dirancang untuk menjamin adanya redundansi pada node-node yang ada. Hal ini memastikan bahwa sistem tetap dapat memberikan layanan kepada klien dengan tingkat ketersediaan yang tinggi, meskipun terjadi kegagalan pada satu atau beberapa node dalam sistem.

Partition tolerance memastikan bahwa sistem dapat beroperasi dengan baik meskipun terjadi pemisahan atau kegagalan pada komunikasi antar node dalam sistem distribusi. Ini berarti bahwa meskipun terjadi pemisahan pada jaringan, setiap node harus tetap dapat menerima dan memproses permintaan dari klien secara mandiri dan kemudian mengembalikan hasilnya dengan benar saat koneksi jaringan terjalin kembali.

Meskipun RDBMS tidak secara langsung dirancang untuk memenuhi teorema CAP, RDBMS masih memegang peran penting dalam manajemen database relasional. Namun, untuk memenuhi sifat-sifat dalam teorema tersebut, NoSQL telah muncul sebagai solusi alternatif yang lebih cocok untuk aplikasi yang membutuhkan skalabilitas dan toleransi kesalahan yang tinggi. Dalam sistem database NoSQL, fokus diberikan pada partition tolerance dan availability dengan cara memperluas jaringan node dan menghindari konsistensi pada level yang sangat tinggi. Meskipun demikian, hal ini memungkinkan sistem untuk memperoleh tingkat skalabilitas yang lebih tinggi, meskipun dengan mengurangi konsistensi data yang tinggi. Oleh karena itu NoSQL sering digunakan dalam aplikasi web yang membutuhkan respons cepat dan skala yang besar.

Pengenalan NoSQL

NoSQL menjadi semakin populer di industri teknologi informasi. Istilah NoSQL sendiri merupakan singkatan dari "Not Only SQL," yang menunjukkan bahwa NoSQL bukan pengganti SQL, tetapi sebuah alternatif untuk database relasional. Konsep NoSQL mengacu pada database yang tidak menggunakan model tabel relasional dan skema tetap seperti pada database relasional.

Salah satu kelebihan utama NoSQL adalah fleksibilitasnya dalam mengelola data dengan struktur yang berbeda-beda tanpa mengganggu keseluruhan database. NoSQL juga dirancang untuk dapat mengelola data dalam skala besar dan dapat di-scaling secara horizontal, menjadikan NoSQL pilihan yang tepat untuk mengelola data semi-struktural atau bahkan tidak struktural, seperti data yang dihasilkan oleh media sosial, log server, atau data sensor.

> Penting untuk diingat bahwa pemilihan jenis database harus didasarkan pada jenis data yang akan disimpan dan kebutuhan aplikasi yang dibangun. Ada beberapa jenis data yang membutuhkan skema yang terstruktur, dan database relasional lebih cocok untuk menangani jenis data ini. Namun, jika data yang diolah memiliki jenis dan struktur yang berbeda-beda, NoSQL dapat membantu pengguna untuk mengelola data dengan lebih efektif.

Salah satu implementasi dari NoSQL yang populer adalah MongoDB, sebuah database document-oriented yang dirancang untuk menangani data semi-struktural. Dengan MongoDB, pengguna dapat mengelola data dalam format document yang fleksibel dan dinamis tanpa mengubah skema database. MongoDB juga dapat di-scale secara horizontal dan menawarkan ketersediaan dan skalabilitas tinggi untuk aplikasi web dan mobile.

Gambaran Umum MongoDB

MongoDB adalah salah satu jenis database NoSQL yang sangat populer dan banyak digunakan oleh perusahaan-perusahaan besar dan startup. MongoDB memberikan keunggulan dalam hal fleksibilitas penyimpanan dan manipulasi data, yang memudahkan pengembang untuk bekerja dengan data yang kompleks dan dinamis.

Keuntungan menggunakan MongoDB adalah fleksibilitas penyimpanan dan manipulasi data yang memudahkan pengembang untuk menyimpan data yang sangat dinamis. Selain itu, MongoDB juga dapat dengan mudah di-scale secara horizontal, yang memungkinkan pengembang untuk menambahkan kapasitas penyimpanan secara efisien saat terjadi peningkatan volume data.

Arsitektur MongoDB terdiri dari beberapa komponen utama yang bekerja bersama-sama untuk memberikan performa yang optimal dan ketersediaan data yang tinggi, serta kemampuan untuk melakukan skalabilitas secara horizontal dengan mudah.

Node

Node pada MongoDB adalah unit terkecil dalam arsitektur MongoDB, yang berisi satu atau beberapa proses MongoDB. Setiap node MongoDB dapat berjalan sebagai node mandiri, atau digabungkan dengan node-node lain untuk membentuk sebuah cluster MongoDB. Setiap node MongoDB memiliki sebuah instance dari MongoDB server yang berjalan, serta memegang data dan konfigurasi khususnya.

Cluster

Cluster pada MongoDB terdiri dari satu atau lebih node MongoDB yang bekerja bersama untuk menyediakan pelayanan untuk aplikasi atau pengguna. Cluster dapat dikonfigurasi untuk memastikan ketersediaan data yang tinggi dan menghindari kerusakan data yang tidak

diinginkan. Cluster juga memungkinkan data untuk disebarkan di beberapa node dan dilakukan replikasi antara node, sehingga memastikan keamanan dan redundansi data.

Replica Set

Replica set pada MongoDB adalah sebuah grup dari node yang menjalankan sebuah proses replikasi data, sehingga memastikan redundansi data dalam cluster. Setiap node dalam sebuah replica set memiliki sebuah peran yang ditentukan sebelumnya, yaitu primary, secondary atau arbiter. Node primary bertanggung jawab untuk menerima permintaan tulis dan memperbarui data, sedangkan node secondary bertanggung jawab untuk melakukan replikasi data dari node primary. Node arbiter berfungsi untuk memastikan ketika terjadi pemilihan node primary.

Sharded Cluster

Sharded cluster pada MongoDB adalah kumpulan beberapa node yang terdiri dari tiga jenis node, yaitu konfigurasi, router dan shard. Node konfigurasi berfungsi untuk menyimpan metadata dan konfigurasi dari sharded cluster, sedangkan node router bertanggung jawab untuk memperluas dan mengarahkan permintaan ke shard yang sesuai. Node shard merupakan node yang memegang data aplikasi dan berfungsi untuk memproses permintaan dari router.

Instalasi

Instalasi pada Windows

Berikut adalah langkah-langkah instalasi MongoDB pada sistem operasi Windows:

1. Unduh installer MongoDB dari website resmi MongoDB (https://www.mongodb.com/download-center/community)
2. Buka file installer yang sudah didownload, dan klik "Next" pada halaman awal.
3. Baca dan setujui persyaratan lisensi MongoDB, lalu klik "Next".
4. Pilih opsi "Complete" untuk menginstal MongoDB dengan semua opsi standar, dan klik "Next".
5. Pilih folder tempat MongoDB akan diinstal pada sistem, kemudian klik "Next".
6. Klik "Install" untuk memulai proses instalasi MongoDB.
7. Setelah instalasi selesai, pengguna dapat memilih opsi "Run MongoDB Compass" untuk membuka MongoDB Compass, atau opsi "Finish" untuk menyelesaikan proses instalasi.

Instalasi pada Mac

Berikut adalah langkah-langkah instalasi MongoDB pada sistem operasi Mac:

1. Unduh installer MongoDB dari website resmi MongoDB (https://www.mongodb.com/download-center/community)
2. Buka file installer yang sudah didownload, dan seret ikon MongoDB ke folder "Applications" pada sistem.
3. Buka aplikasi Terminal pada Mac, dan ketik perintah `mkdir -p /data/db` untuk membuat folder untuk data MongoDB.
4. Jalankan MongoDB dengan perintah `mongod` pada Terminal.
5. Buka aplikasi MongoDB Compass untuk mengakses MongoDB pada Mac.

Instalasi pada Linux

Berikut adalah langkah-langkah instalasi MongoDB pada sistem operasi Linux:

1. Buka terminal pada Linux, dan tambahkan repositori MongoDB ke dalam sistem dengan perintah `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 68818C72E52529D4` (untuk Ubuntu 16.04).
2. Tambahkan repositori MongoDB ke dalam sistem dengan perintah `echo "deb **[http://repo.mongodb.org/apt/ubuntu](http://repo.mongodb.org/apt/ubuntu)** xenial/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list` (untuk Ubuntu 16.04).
3. Jalankan perintah `sudo apt-get update` untuk memperbarui daftar paket pada sistem Anda.
4. Instal MongoDB dengan perintah `sudo apt-get install -y mongodb-org`.
5. Buat folder untuk data MongoDB dengan perintah `sudo mkdir -p /data/db`.
6. Jalankan MongoDB dengan perintah `sudo mongod`.
7. Buka aplikasi MongoDB Compass untuk mengakses MongoDB pada sistem Linux.

MongoDB Compass

MongoDB Compass adalah GUI (Graphical User Interface) resmi untuk MongoDB yang menyediakan antarmuka grafis untuk mempermudah pengguna dalam menjelajahi dan memanipulasi data di database MongoDB. Dengan menggunakan MongoDB Compass, pengguna dapat membuat dan menjalankan query dengan lebih mudah, melihat statistik dan metrik performa, serta memvisualisasikan data dalam bentuk grafik dan diagram.

Konfigurasi

Setelah berhasil menginstal MongoDB pada sistem Anda, langkah selanjutnya adalah mengkonfigurasi agar sesuai dengan kebutuhan aplikasi Anda. Dalam bab ini, kita akan membahas beberapa opsi konfigurasi MongoDB yang umum digunakan.

Jaringan

Pertama-tama, Kita perlu mengkonfigurasi jaringan pada MongoDB agar dapat diakses oleh aplikasi atau pengguna yang membutuhkan. Untuk mengkonfigurasinya, tambahkan baris berikut pada file konfigurasi MongoDB yang biasanya berada di direktori `/etc/mongod.conf`:

```
bindIp: 0.0.0.0
```

Baris ini akan mengizinkan MongoDB menerima koneksi dari semua alamat IP pada jaringan. Jika Kita ingin mengizinkan koneksi dari alamat IP tertentu saja, ganti `0.0.0.0` dengan alamat IP yang diizinkan.

Database

Anda juga dapat mengkonfigurasi opsi database pada MongoDB, seperti lokasi penyimpanan database dan opsi pemulihan data. Berikut ini adalah beberapa opsi konfigurasi database yang paling umum:

- `storage.dbPath`: Opsi ini menentukan lokasi penyimpanan database pada sistem Anda. Defaultnya adalah `/data/db`, tetapi bisa diubah sesuai kebutuhan.
- `storage.journal.enabled`: Opsi ini menentukan apakah MongoDB menggunakan jurnal atau tidak. Jika diaktifkan, MongoDB akan menggunakan jurnal untuk memastikan - konsistensi data. Defaultnya adalah `true`.

- `storage.journal.commitIntervalMs`: Opsi ini menentukan interval waktu dalam milidetik untuk melakukan commit jurnal. Defaultnya adalah `100`.

Keamanan

Penting untuk mengkonfigurasi keamanan pada MongoDB agar data Kita terlindungi dari akses yang tidak sah. Beberapa opsi konfigurasi keamanan yang dapat Kita terapkan pada MongoDB adalah:

- `security.authorization`: Opsi ini menentukan apakah MongoDB menggunakan autentikasi atau tidak. Jika diaktifkan, pengguna harus melakukan autentikasi sebelum dapat mengakses data. Defaultnya adalah `disabled`.
- `security.authenticationMechanisms`: Opsi ini menentukan mekanisme autentikasi yang dapat digunakan oleh MongoDB. Defaultnya adalah `SCRAM-SHA-1`.
- `net.ssl.mode`: Opsi ini menentukan apakah MongoDB menggunakan koneksi SSL atau tidak. Jika diaktifkan, koneksi akan dienkripsi untuk melindungi data. Defaultnya adalah `disabled`.

Replikasi

Jika Kita menggunakan MongoDB untuk replikasi data, maka Kita dapat mengkonfigurasi opsi replikasi pada MongoDB. Beberapa opsi konfigurasi replikasi yang dapat Kita terapkan pada MongoDB adalah:

- `replication.replSetName`: Opsi ini menentukan nama set replikasi pada MongoDB. Setiap server pada replikasi harus memiliki nama yang sama untuk dapat saling berkomunikasi. Defaultnya adalah empty string.
- `replication.oplogSizeMB`: Opsi ini menentukan ukuran maksimal oplog pada MongoDB. Oplog digunakan untuk merekam operasi database yang terjadi pada server, dan oplog

yang terlalu kecil dapat menyebabkan hilangnya data pada replikasi. Defaultnya adalah 30 MB.

Sharding

Jika Kita menggunakan MongoDB untuk sharding data, maka Kita dapat mengkonfigurasi opsi sharding pada MongoDB. Beberapa opsi konfigurasi sharding yang dapat Kita terapkan pada MongoDB adalah:

- `sharding.clusterRole`: Opsi ini menentukan peran node pada cluster sharding. Node dapat menjadi konfigurasi router atau shard. Defaultnya adalah config server.
- `sharding.configDB`: Opsi ini menentukan lokasi konfigurasi server pada MongoDB. Konfigurasi server digunakan untuk mengatur metadata pada sharded cluster. Defaultnya adalah `localhost:27019`.

Log

Terakhir, Kita dapat mengkonfigurasi opsi log pada MongoDB untuk memantau aktivitas server dan menemukan masalah jika terjadi. Beberapa opsi konfigurasi log yang dapat Kita terapkan pada MongoDB adalah:

- `systemLog.verbosity`: Opsi ini menentukan tingkat kebisingan log pada MongoDB. Defaultnya adalah `0`, yang hanya mencatat kesalahan kritis.
- `systemLog.path`: Opsi ini menentukan lokasi file log pada MongoDB. Defaultnya adalah `/var/log/mongodb/mongod.log`.

Konsep

Dasar

Database

MongoDB memungkinkan kita untuk membuat beberapa database dalam satu server. Setiap database memiliki nama yang unik dan dapat memiliki satu atau lebih collection. database pada MongoDB mirip dengan database pada database relasional. Contohnya, kita dapat membuat database "aplikasi" yang memiliki collection "pengguna" dan collection "produk" untuk menyimpan data pengguna dan produk dalam aplikasi tersebut.

Collection

Dalam MongoDB, document dikelompokkan ke dalam collection (collection) yang mirip dengan tabel pada database relasional. Collection pada MongoDB memiliki nama yang unik dan hanya dapat menyimpan document dengan struktur yang sama. Contohnya, dalam collection "pengguna", kita hanya dapat menyimpan document yang berisi informasi pengguna yang memiliki field yang sama.

Document

Dalam MongoDB, data disimpan dalam bentuk document yang merepresentasikan entitas atau objek. Document dalam MongoDB diwakili dalam format JSON (JavaScript Object Notation). Document dalam MongoDB dapat berisi banyak field yang berbeda-beda, tergantung pada kebutuhan aplikasi. Contohnya, document pada collection "pengguna" dapat berisi informasi seperti nama pengguna, alamat email, nomor telepon, dan sebagainya.

Field

Field dalam MongoDB adalah elemen terkecil dari document MongoDB

yang berisi nilai data. Field dapat berisi tipe data seperti string, integer, boolean, atau bahkan array dan document bersarang. Contohnya, field pada document yang berisi data pengguna dapat berisi informasi seperti nama, alamat email, usia, dan sebagainya.

> Dalam sistem database tradisional seperti MySQL atau Oracle, data disimpan dalam bentuk tabel yang terdiri dari baris dan kolom. Namun, dalam MongoDB, data disimpan dalam bentuk document yang lebih fleksibel.

Pengenalan JSON dan BSON

MongoDB adalah database yang menggunakan format document untuk menyimpan dan mengelola data. Format document ini memungkinkan MongoDB untuk menyimpan data yang kompleks dan terstruktur dengan mudah. Dalam MongoDB, format document yang digunakan adalah JSON dan BSON.

JSON

JSON (JavaScript Object Notation) adalah format data yang ringan dan mudah dibaca dan ditulis oleh manusia. JSON menggunakan sintaksis yang mirip dengan bahasa pemrograman JavaScript, sehingga mudah dimengerti oleh banyak pengembang. Format JSON terdiri dari dua elemen utama, yaitu field dan nilai. Field dituliskan dalam bentuk string yang diapit oleh tanda kutip ganda (`"`), sedangkan nilai bisa berupa tipe data apapun seperti string, integer, boolean, array, atau object. Contoh format JSON adalah sebagai berikut:

```
{
    "nama": "Andi",
    "umur": 25,
    "alamat": {
        "jalan": "Jl. Pahlawan",
        "kota": "Jakarta",
        "provinsi": "DKI Jakarta"
    },
    "hobi": ["membaca", "bersepeda", "menonton film"]
}
```

BSON

BSON (Binary JSON) adalah format data biner yang juga dapat digunakan untuk menyimpan data di MongoDB. BSON adalah ekstensi dari format JSON, yang memungkinkan penyimpanan data yang lebih efisien dan cepat. Format BSON memiliki struktur yang mirip dengan JSON, namun menggunakan tipe data biner yang memungkinkan penggunaan sumber daya jaringan yang lebih efisien dan manipulasi data yang lebih cepat. Contoh format BSON untuk data yang sama dengan contoh format JSON di atas adalah sebagai berikut:

```
\x34\x00\x00\x00             // Panjang document (52 byte)
\x02                         // Tipe data string
$0                           // Nama field
\x05\x00\x00\x00Andi\x00     // Nilai field (string)
\x10umur\x00\x19\x00\x00\x00  // Nilai field (integer)
\x03alamat\x00               // Nama field
\x26\x00\x00\x00\x02jalan\x00\x0B\x00\x00\x00Jl.
Pahlawan\x00
\x04kota\x00\x07\x00\x00\x00Jakarta\x00
\x08provinsi\x00\x0B\x00\x00\x00DKI Jakarta\x00
\x04hobi\x00                 // Nama field
\x13\x00\x00\x00             // Panjang array (19 byte)
\x05\x00\x00\x00membaca\x00  // Nilai array (string)
\x08\x00\x00\x00bersepeda\x00
\x0F\x00\x00\x00menonton film\x00
```

BSON memiliki beberapa keunggulan dibandingkan JSON, yaitu ukuran document yang lebih kecil dan manipulasi data yang lebih cepat. BSON juga mendukung beberapa tipe data tambahan seperti tanggal dan waktu, byte array, dan geospatial data yang tidak didukung oleh format JSON.

Namun, penggunaan format BSON juga memiliki kelemahan, yaitu kurang mudah dibaca dan ditulis oleh manusia. Karena format BSON menggunakan representasi biner, pengembang yang tidak terbiasa dengan format ini mungkin akan kesulitan dalam memahami dan mengelola data di MongoDB.

Penggunaan format JSON atau BSON di MongoDB tergantung pada kebutuhan penggunaan dan preferensi pengembang. JSON cocok digunakan untuk pengembangan aplikasi yang sederhana atau ketika manusia masih memerlukan kemampuan untuk membaca dan menulis document. Sementara BSON lebih cocok digunakan untuk aplikasi yang

memerlukan manipulasi data yang cepat dan efisien, atau ketika ukuran document menjadi masalah seperti pada penggunaan sumber daya jaringan yang terbatas.

Fleksibilitas

Fleksibilitas Skema

Salah satu keuntungan utama dari menggunakan document pada MongoDB adalah fleksibilitas skema. Skema adalah aturan yang menentukan struktur dan tipe data pada database. Pada sistem database tradisional, skema harus didefinisikan sebelum data dapat disimpan. Namun, pada MongoDB, skema dapat berubah seiring waktu tanpa memerlukan pengaturan yang rumit.

Sebagai contoh, mari kita lihat skema dasar untuk menyimpan data produk:

```
{
  "nama": "Sepatu Sneakers",
  "harga": 500000,
  "berat": 0.5,
  "stok": 10
}
```

Namun, suatu saat kita ingin menambahkan field `warna` pada document tersebut. Dalam MongoDB, kita hanya perlu menambahkan field baru tersebut ke document tanpa perlu mengubah skema secara keseluruhan.

```
{
  "nama": "Sepatu Sneakers",
  "harga": 500000,
  "berat": 0.5,
  "stok": 10,
  "warna": "Merah"
}
```

Fleksibilitas Query

MongoDB menyediakan berbagai jenis query yang memungkinkan pengguna untuk mengambil data dari database sesuai dengan kebutuhan. Salah satu fitur query yang paling fleksibel pada MongoDB adalah kemampuan untuk melakukan query berdasarkan isi field yang terdapat dalam document.

Sebagai contoh, mari kita ingin mencari semua produk yang memiliki `harga` di atas 500 ribu:

```
db.produk.find(
  {
    harga: {
      $gt: 500000
    }
  }
)
```

Dalam contoh di atas, kita menggunakan operator `$gt` (greater than) untuk mencari semua document yang memiliki nilai field `harga` di atas 500 ribu.

Kemampuan untuk melakukan query berdasarkan isi field pada

document memungkinkan pengguna untuk mengambil data secara fleksibel dan efisien. Selain itu, MongoDB juga menyediakan berbagai jenis query lain seperti query teks dan query spasial yang memungkinkan pengguna untuk melakukan pencarian data dengan lebih kompleks.

Fleksibilitas Indexing

Indexing adalah proses membuat indeks pada database untuk mempercepat proses pencarian data. Pada MongoDB, indexing dapat diterapkan pada field-field pada document untuk meningkatkan performa query. Indexing pada MongoDB sangat fleksibel karena dapat diterapkan pada field-field yang berbeda pada setiap document.

Sebagai contoh, mari kita ingin membuat indeks pada field `nama` pada document `karyawan`:

```
db.karyawan.createIndex(
  {
    nama: 1
  }
)
```

Dalam contoh di atas, kita menggunakan method `createIndex` untuk membuat indeks pada field `nama` pada document `karyawan`. Indeks ini akan meningkatkan performa query yang melakukan pencarian berdasarkan field `nama`.

Dengan adanya fleksibilitas indexing pada MongoDB, pengguna dapat membuat indeks pada field-field yang paling sering diakses pada setiap document, sehingga performa query dapat ditingkatkan secara signifikan.

MongoDB Query Language (MQL)

Salah satu membedakan MongoDB dari sistem database lainnya adalah kemampuan untuk melakukan query data dengan menggunakan bahasa pemrograman yang disebut dengan MongoDB Query Language atau MQL. MQL adalah sebuah bahasa yang dirancang khusus untuk MongoDB dan memungkinkan pengguna untuk melakukan query data dengan lebih mudah dan efektif.

MQL memiliki sintaks yang mirip dengan sintaks SQL pada sistem database relasional, namun memiliki beberapa perbedaan terkait dengan struktur data document pada MongoDB. Pada MQL, kita dapat melakukan query untuk mencari document yang sesuai dengan kriteria tertentu. Kriteria ini dapat berupa nilai dari field-field pada document, atau bahkan query yang berhubungan antara document dalam collection yang berbeda.

Tipe Data

MongoDB memungkinkan penyimpanan dan pengambilan data yang sangat fleksibel, terutama karena penggunaannya yang mengandalkan tipe data document yang dinamis. Dalam MongoDB, sebuah document dapat memiliki tipe data yang berbeda untuk setiap atributnya, dan atribut dapat berubah dari satu document ke document lainnya. Hal ini berbeda dengan database relasional tradisional yang mengharuskan tipe data yang seragam di seluruh baris tabel.

Double

Tipe data double digunakan untuk menyimpan bilangan desimal. Dalam MongoDB, tipe data double dinyatakan dalam format angka desimal atau notasi 64-bit. Contohnya: `3.14`.

String

Tipe data string digunakan untuk menyimpan karakter atau teks. Dalam

MongoDB, tipe data string harus diapit oleh tanda kutip ganda (`"`). Contohnya: `"Ini adalah sebuah string"`.

Object

Tipe data abject digunakan untuk menyimpan document yang kompleks. Dalam MongoDB, tipe data abject dinyatakan dalam bentuk JSON. Contohnya:

```
{
   "nama": "Andi",
   "umur": 25,
   "alamat": {
      "jalan": "Jl. Merdeka",
      "kota": "Jakarta"
   }
}
```

Array

Tipe data array digunakan untuk menyimpan beberapa nilai atau informasi dalam satu field. Dalam MongoDB, tipe data array dinyatakan dalam bentuk kurung siku (`[]`). Contohnya:

```
["apel", "jeruk", "pisang"]
```

Binary Data

Tipe data binary digunakan untuk menyimpan data biner seperti gambar atau file. Dalam MongoDB, tipe data binary dinyatakan dalam bentuk data BSON.

ObjectId

Tipe data `ObjectId` digunakan untuk menyimpan nilai ID unik yang digunakan oleh MongoDB untuk mengidentifikasi document. Setiap document di MongoDB harus memiliki `_id` field yang menyimpan nilai `ObjectId`:

```
ObjectId("61d95e27b601561c1a10a29a")
```

Boolean

Tipe data boolean digunakan untuk menyimpan nilai `true` atau `false`.

Date

Tipe data date digunakan untuk menyimpan informasi tanggal dan waktu. Dalam MongoDB, tipe data date dinyatakan dalam bentuk timestamp milidetik sejak 1 Januari 1970. Contohnya:

```
ISODate("2017-12-31T23:59:59.000Z")
```

Null

Tipe data null digunakan untuk menyimpan nilai `null` atau kosong. Dalam MongoDB, tipe data null dituliskan dengan `null`.

Regular Expression

Tipe data regular expression digunakan untuk menyimpan pola yang digunakan untuk pencarian dan pemrosesan teks.

JavaScript

Tipe data JavaScript digunakan untuk menyimpan kode JavaScript.

Dalam MongoDB, tipe data JavaScript dituliskan dalam bentuk objek dengan satu field, yaitu `$code` yang berisi kode JavaScript. Contohnya:

```
{
  "$code": "function tambah(a, b) { return a + b; }"
}
```

32-bit Integer

Tipe data 32-bit Integer digunakan untuk menyimpan bilangan bulat dengan presisi 32-bit. Dalam MongoDB, tipe data 32-bit Integer dinyatakan dalam format angka desimal atau notasi 32-bit.

64-bit Integer

Tipe data 64-bit Integer digunakan untuk menyimpan bilangan bulat dengan presisi 64-bit. Dalam MongoDB, tipe data 64-bit Integer dinyatakan dalam format angka desimal atau notasi 64-bit.

Timestamp

Tipe data timestamp digunakan untuk menyimpan informasi waktu yang spesifik dalam bentuk 64-bit integer. Dalam MongoDB, tipe data timestamp dinyatakan dengan menggunakan nilai yang terdiri dari dua bagian, yaitu timestamp dan ordinal:

```
Timestamp(1646151931, 1)
```

Decimal128

Tipe data Decimal128 digunakan untuk menyimpan bilangan pecahan yang memiliki presisi tinggi. Dalam MongoDB, tipe data Decimal128 dinyatakan dalam format angka desimal atau notasi 128-bit. Contohnya:

```
NumberDecimal("3.14159265358979323846264338327 9")
```

Min Key

Tipe data Min Key digunakan untuk merepresentasikan nilai terkecil yang mungkin dalam MongoDB. Min Key sering digunakan dalam pengurutan dan pencarian data. Dalam MongoDB, Min Key dituliskan dengan `MinKey`.

Max Key

Tipe data Max Key digunakan untuk merepresentasikan nilai terbesar yang mungkin dalam MongoDB. Max Key sering digunakan dalam pengurutan dan pencarian data. Dalam MongoDB, Max Key dituliskan dengan `MaxKey`.

Query

MongoDB Query Language (MQL)

Sintaksis MQL mirip dengan sintaksis SQL pada sistem database relasional, meskipun ada beberapa perbedaan yang terkait dengan struktur data document pada MongoDB. Dalam MQL, kita dapat melakukan query untuk mencari Document yang sesuai dengan kriteria tertentu. Kriteria ini dapat berupa nilai dari field-field pada Document, atau bahkan query yang berhubungan antara Document dalam collection yang berbeda.

Meskipun terdapat perbedaan dalam sintaksis dan fitur antara SQL dan MQL, konsep dasar query pada kedua bahasa tetap sama. Oleh karena itu, kita akan membahas cara menggunakan query untuk memahami konsep-konsep tersebut dengan lebih baik.

SELECT

Pada SQL, perintah `SELECT` digunakan untuk mengambil data dari tabel. Berikut contoh penggunaan perintah `SELECT` untuk mengambil semua kolom dari tabel `pengguna` yang memiliki umur lebih besar dari "18":

```
SELECT *
FROM
  pengguna
WHERE
  umur > 18;
```

Sedangkan pada MongoDB, untuk mengambil data dari sebuah collection, kita menggunakan perintah `find()` seperti contoh berikut:

```
db.pengguna.find(
  {
    umur: { $gt: 18 }
  }
)
```

Pada dasarnya, sebuah query di MongoDB terdiri dari dua bagian yaitu selector dan projection. Selector adalah bagian yang digunakan untuk memilih data yang akan ditampilkan atau diolah. Projection, di sisi lain, digunakan untuk menentukan bagian data mana yang akan ditampilkan.

Selector

Selector pada MongoDB digunakan untuk memilih data berdasarkan kriteria tertentu. Kriteria ini dapat berupa satu atau beberapa kondisi. Berikut adalah beberapa contoh selector yang umum digunakan pada MongoDB:

- `$eq`: Memilih data yang sama dengan nilai tertentu.
- `$gt`: Memilih data yang lebih besar dari nilai tertentu.
- `$lt`: Memilih data yang lebih kecil dari nilai tertentu.
- `$in`: Memilih data yang nilainya terdapat dalam sebuah array tertentu.

Berikut adalah contoh penggunaan selector yang akan memilih semua data dari collection `pengguna` yang memiliki nilai `umur` lebih besar dari 25:

```
db.pengguna.find(
  {
    umur: { $gt: 25 }
  }
)
```

Projection

Projection pada MongoDB digunakan untuk menentukan bagian data mana yang akan ditampilkan. Hal ini berguna jika Kita hanya ingin menampilkan bagian-bagian tertentu dari sebuah document, atau jika Kita ingin mengurangi jumlah data yang harus ditampilkan.

Berikut ini adalah contoh penggunaan projection pada MongoDB yang akan memilih semua data dari collection `pengguna` yang memiliki nilai `umur` lebih besar dari 25, dan hanya menampilkan kolom `nama` dan `umur` dari data tersebut:

```
db.pengguna.find(
  {
    umur: { $gt: 25 }
  },
  {
    nama: 1,
    umur: 1
  }
)
```

INSERT

Dalam SQL, perintah `INSERT` digunakan untuk menambahkan data ke dalam tabel. Berikut adalah contoh untuk menambahkan data ke dalam

tabel:

```
INSERT INTO pengguna (nama, email, umur)
VALUES ('Joko', 'joko@example.com', 30);
```

Sedangkan pada MongoDB, untuk menambahkan Document ke dalam collection, kita menggunakan `insertOne()` atau `insertMany()`:

```
db.pengguna.insertOne({
  nama: 'Joko',
  email: 'joko@example.com',
  umur: 30
});
```

UPDATE

Dalam SQL, `UPDATE` digunakan untuk memperbarui data yang sudah ada dalam tabel. Berikut adalah contoh penggunaan `UPDATE` untuk mengubah nilai pada kolom `umur` menjadi "26" untuk data dengan `nama` Joko dalam tabel `pengguna`:

```
UPDATE pengguna
SET umur = 26
WHERE nama = 'Joko';
```

Sedangkan pada MongoDB, untuk memperbarui Document dalam collection, kita menggunakan `updateOne()` atau `updateMany()`:

```
db.pengguna.updateOne(
  { nama: 'Joko' },
  {
    $set: { umur: 26 }
  }
);
```

DELETE

Pada SQL, perintah `DELETE` digunakan untuk menghapus data dari tabel. Sebagai contoh, kita akan menghapus data dengan nama Joko dari tabel pengguna:

```
DELETE FROM pengguna
WHERE nama = 'Joko';
```

Sementara pada MongoDB, untuk menghapus Document dari collection, kita dapat menggunakan `deleteOne()` atau `deleteMany()`:

```
db.pengguna.deleteOne(
  { nama: 'Joko' }
);
```

Studi Kasus

Dalam bisnis apapun, mengumpulkan data yang akurat dan terorganisir adalah sangat penting. Menjaga data produk dan penjualan yang terbaru dan teratur dapat membantu mengoptimalkan proses bisnis. Oleh karena itu, dalam studi kasus ini, kita akan mengeksplorasi bagaimana perusahaan "Electroindo" dapat membuat database yang tepat untuk menyimpan data produk mereka, termasuk informasi stok, harga, dan penjualan. Dengan memahami bagaimana mengelola data dengan benar, perusahaan dapat mengambil keputusan yang lebih cerdas, meningkatkan efisiensi operasional, dan memperkuat posisi mereka di pasar elektronik yang kompetitif. Mari kita mulai dengan mempertimbangkan contoh data produk penjualan yang ingin disimpan oleh "Electroindo".

Schema Design

Pertama-tama kita perlu menentukan schema atau struktur collection yang akan digunakan untuk menyimpan data produk penjualan. Berikut adalah contoh schema yang akan dibuat:

```
{
    "_id": ObjectId,
    "product_name": String,
    "description": String,
    "price": Number,
    "quantity": Number,
    "category": String,
    "tags": [String],
    "created_at": Date,
    "updated_at": Date
}
```

Penjelasan dari masing-masing field:

Field	Deskripsi
`_id`	Primary key yang digenerate oleh MongoDB
`product_name`	Nama dari produk
`description`	Deskripsi dari produk
`price`	Harga dari produk
`quantity`	Jumlah stok dari produk
`category`	Kategori dari produk
`tags`	Tag-tag terkait dengan produk
`created_at`	Tanggal pembuatan produk
`updated_at`	Tanggal terakhir kali produk diperbarui atau diupdate ulang

Insert Data

Setelah schema telah ditentukan, selanjutnya kita akan melakukan insert data ke dalam database MongoDB. Berikut adalah contoh data produk penjualan yang akan kita insert:

```
db.products.insertMany([
    {
        "product_name": "Handphone Samsung Galaxy S21",
        "description": "Handphone dengan spesifikasi tinggi,
layar OLED 6.2 inch, kamera 108MP",
        "price": 15000000,
        "quantity": 50,
        "category": "Handphone",
        "tags": ["Samsung", "Galaxy", "S21"],
        "created_at": ISODate("2017-03-01T10:00:00Z"),
        "updated_at": ISODate("2017-03-01T10:00:00Z")
    },
    {
        "product_name": "Laptop ASUS ROG Strix G17",
        "description": "Laptop gaming dengan spesifikasi
tinggi, prosesor Intel Core i7-11800H, kartu grafis NVIDIA
GeForce RTX 3070",
        "price": 25000000,
        "quantity": 20,
        "category": "Laptop",
        "tags": ["ASUS", "ROG", "Gaming"],
        "created_at": ISODate("2017-03-02T11:30:00Z"),
        "updated_at": ISODate("2017-03-02T11:30:00Z")
    },
    {
        "product_name": "Smart TV LG OLED CX",
        "description": "Smart TV dengan layar OLED 4K, suara
```

```
    Dolby Atmos, dan fitur pintar LG ThinQ",
            "price": 20000000,
            "quantity": 30,
            "category": "TV",
            "tags": ["LG", "OLED", "4K", "ThinQ"],
            "created_at": ISODate("2017-03-03T14:00:00Z"),
            "updated_at": ISODate("2017-03-03T14:00:00Z")
        }
    ])
```

Mencari Data

Setelah data berhasil di-insert, kita dapat melakukan query untuk melihat data yang telah disimpan di dalam database dengan perintah `find`:

```
db.products.find()
```

Selector

Selector Perbandingan

Menampilkan produk dengan kategori "Handphone":

```
db.products.find({
  category: "Handphone"
})
```

Menampilkan produk dengan tag "Samsung":

```
db.products.find({
  tags: "Samsung"
})
```

Menampilkan produk dengan harga lebih besar dari 20 juta:

```
db.products.find({
  price: {
    $gt: 20000000
  }
})
```

Menampilkan produk dengan jumlah stok kurang dari 25:

```
db.products.find({
  quantity: {
    $lt: 25
  }
})
```

Menampilkan produk dengan harga di antara 15 juta dan 25 juta:

```
db.products.find({
  price: {
    $gte: 15000000,
    $lte: 25000000
  }
})
```

Menampilkan produk dengan harga tidak sama dengan 20 juta:

```
db.products.find({
  price: {
    $ne: 20000000
  }
})
```

Menampilkan produk yang dibuat setelah tanggal 2 Maret 2017:

```
db.products.find({
  created_at: {
    $gt: ISODate("2017-03-02T00:00:00Z")
  }
})
```

Selector Logika

Operator `$and` dapat digunakan untuk mencari data yang memenuhi beberapa kriteria pada waktu yang bersamaan. Berikut ini adalah contoh penggunaannya untuk mencari produk yang harganya lebih dari 10 juta dan stoknya lebih dari 20:

```
db.products.find({
  $and: [
    {
      price: {
        $gt: 10000000
      }
    },
    {
      quantity: {
        $gt: 20
      }
    }
  ]
})
```

Operator `$or` dapat digunakan untuk mencari data yang memenuhi salah satu dari beberapa kriteria yang diberikan. Berikut ini adalah contoh penggunaannya untuk mencari produk dengan kategori "Handphone" atau "Laptop":

```
db.products.find({
  $or: [
    { category: "Handphone" },
    { category: "Laptop" }
  ]
})
```

Operator `$not` dapat digunakan untuk mencari data yang tidak memenuhi kriteria yang diberikan. Berikut ini adalah contoh penggunaannya untuk mencari produk dengan stok kurang dari atau sama dengan 30:

```
db.products.find({
  quantity: {
    $not: {
      $gt: 30
    }
  }
})
```

Operator `$nor` dapat digunakan untuk mencari data yang tidak memenuhi salah satu atau semua kriteria yang diberikan. Berikut ini adalah contoh penggunaannya untuk mencari produk dengan harga lebih dari 20 juta atau stok kurang dari 10:

```
db.products.find({
  $nor: [
    {
      price: {
        $gt: 20000000
      }
    },
    {
      quantity: {
        $lt: 10
      }
    }
  ]
})
```

Selector Pencocokan

Operator `$in` dapat digunakan untuk mencari data yang memiliki nilai

tertentu dalam suatu array. Berikut ini adalah contoh penggunaannya untuk mencari produk dengan kategori "Handphone" atau "TV":

```
db.products.find({
  category: {
    $in: ["Handphone", "TV"]
  }
})
```

Operator `$nin` dapat digunakan untuk mencari data yang tidak memiliki nilai tertentu dalam suatu array. Berikut ini adalah contoh penggunaannya untuk mencari produk yang tidak memiliki `tag` "OLED":

```
db.products.find({
  tags: {
    $nin: ["OLED"]
  }
})
```

Operator `$regex` dapat digunakan untuk mencari data yang memiliki pola tertentu dalam suatu field. Berikut ini adalah contoh penggunaannya untuk mencari produk yang memiliki "Galaxy" dalam nama produk:

```
db.products.find({
  product_name: {
    $regex: "Galaxy"
  }
})
```

Operator `$elemMatch` dapat digunakan untuk mencari data yang

memenuhi beberapa kriteria pada suatu array. Berikut ini adalah contoh penggunaannya untuk mencari produk yang memiliki tag "Samsung" dan "S21" dalam array tags:

```
db.products.find({
  tags: {
    $elemMatch: {
      $in: ["Samsung", "S21"]
    }
  }
})
```

Proyeksi

Operator `$project` dapat digunakan untuk memproyeksikan hanya beberapa field dari suatu document yang ingin ditampilkan. Berikut ini adalah contoh penggunaannya untuk memproyeksikan hanya field `product_name` dan `price`:

```
db.products.find(
  {},
  {
    product_name: 1,
    price: 1
  }
)
```

Output dari perintah di atas adalah sebagai berikut:

```
[
  {
    _id: "6404a82ade6192c967c02297",
    product_name: "Handphone Samsung Galaxy S21",
    price: 15000000,
  },
  {
    _id: "6404a82ade6192c967c02299",
    product_name: "Laptop ASUS ROG Strix G17",
    price: 25000000,
  },
  {
    _id: "6404a60e53a61769ceb5a87b",
    product_name: "Smart TV LG OLED CX",
    price: 20000000,
  },
]
```

Operator `$slice` dapat digunakan untuk memproyeksikan hanya beberapa elemen dalam suatu array. Berikut ini adalah contoh penggunaannya untuk memproyeksikan dua elemen pertama dalam array `tags`:

```
db.products.find(
  {},
  {
    product_name: 1,
    price: 1,
    tags: {
      $slice: 2
    }
  }
)
```

Output dari perintah di atas adalah sebagai berikut:

```
[
  {
    "_id": "6404a82ade6192c967c02297",
    "product_name": "Handphone Samsung Galaxy S21",
    "price": 15000000,
    "tags": ["Samsung", "Galaxy"]
  },
  {
    "_id": "6404a82ade6192c967c02299",
    "product_name": "Laptop ASUS ROG Strix G17",
    "price": 25000000,
    "tags": ["ASUS", "ROG"]
  },
  {
    "_id": "6404a60e53a61769ceb5a87b",
    "product_name": "Smart TV LG OLED CX",
    "price": 20000000,
    "tags": ["LG", "OLED"]
  }
]
```

Operator `$elemMatch` dapat digunakan untuk memproyeksikan hanya elemen dalam suatu array yang memenuhi beberapa kriteria tertentu. Berikut ini adalah contoh penggunaannya untuk memproyeksikan elemen dalam array `tags` yang sama dengan "Galaxy":

```
db.products.find(
  {},
  {
    product_name: 1,
    price: 1,
    tags: {
      $elemMatch: {
        $eq: "Galaxy"
      }
    }
  }
)
```

Output dari perintah di atas adalah sebagai berikut:

```
[
  {
    _id: "6404a82ade6192c967c02297",
    product_name: "Handphone Samsung Galaxy S21",
    price: 15000000,
    tags: ["Galaxy"]
  },
  {
    _id: "6404a82ade6192c967c02299",
    product_name: "Laptop ASUS ROG Strix G17",
    price: 25000000
  },
  {
    _id: "6404a60e53a61769ceb5a87b",
    product_name: "Smart TV LG OLED CX",
    price: 20000000
  },
]
```

Aggregation

Operator `$group` dapat digunakan untuk mengelompokkan document berdasarkan suatu field tertentu dan melakukan operasi agregasi pada field yang lain. Berikut ini adalah contoh penggunaannya untuk mengelompokkan produk berdasarkan kategori dan menghitung rata-rata harga dan jumlah stok untuk setiap kategori:

```
db.products.aggregate([
  {
    $group: {
      _id: "$category",
      avgPrice: {
        $avg: "$price"
      },
      totalQuantity: {
        $sum: "$quantity"
      }
    }
  }
])
```

Output dari perintah di atas adalah sebagai berikut:

```
[
  {
    _id: "Handphone",
    avgPrice: 15000000,
    totalQuantity: 50,
  },
  {
    _id: "TV",
    avgPrice: 20000000,
    totalQuantity: 30,
  },
  {
    _id: "Laptop",
    avgPrice: 25000000,
    totalQuantity: 20,
  },
]
```

Operator `$match` dapat digunakan untuk melakukan filtering pada document berdasarkan kriteria tertentu. Berikut ini adalah contoh penggunaannya untuk mencari produk dengan harga lebih dari atau sama dengan 20 juta:

```
db.products.aggregate([
  {
    $match: {
      price: {
        $gte: 20000000
      }
    }
  }
])
```

Operator `$sort` dapat digunakan untuk mengurutkan document berdasarkan suatu field tertentu. Berikut ini adalah contoh penggunaannya untuk mengurutkan produk berdasarkan harga secara menaik:

```
db.products.aggregate([
  {
    $sort: {
      price: 1
    }
  }
])
```

Operator `$limit` dapat digunakan untuk membatasi jumlah document yang ditampilkan. Berikut ini adalah contoh penggunaannya untuk membatasi hanya menampilkan 2 produk:

```
db.products.aggregate([
  {
    $limit: 2
  }
])
```

Operator `$skip` dapat digunakan untuk melewati sejumlah document tertentu. Berikut ini adalah contoh penggunaannya untuk melewati 2 document dan memulai dari document ke-3:

```
db.products.aggregate([
  {
    $skip: 2
  }
])
```

Operator `$lookup` dapat digunakan untuk melakukan join dengan document dari collection yang lain. Berikut ini adalah contoh penggunaannya untuk melakukan join antara produk dan penjualan berdasarkan nama produk:

```
db.products.aggregate([
  {
    $lookup: {
      from: "sales",
      localField: "product_name",
      foreignField: "product_name",
      as: "sales_data"
    }
  }
])
```

Manipulasi Data

Operator `$set` dapat digunakan untuk menambah atau mengubah nilai field dalam suatu document. Berikut ini adalah contoh penggunaannya untuk menambah field "discount" pada produk dengan harga lebih dari 20 juta:

```
db.products.updateMany(
  {
    price: {
      $gt: 20000000
    }
  },
  {
    $set: {
      discount: 0.1
    }
  }
)
```

Operator `$unset` dapat digunakan untuk menghapus field dalam suatu document. Berikut ini adalah contoh penggunaannya untuk menghapus field `description` pada semua produk:

```
db.products.updateMany(
  {},
  {
    $unset: {
      description: ""
    }
  }
)
```

Operator `$inc` dapat digunakan untuk menambah atau mengurangi nilai field yang bertipe data angka dalam suatu document. Berikut ini adalah contoh penggunaannya untuk menambah stok produk dengan harga lebih dari 20 juta sebanyak 10 buah:

```
db.products.updateMany(
  {
    price: {
      $gt: 20000000
    }
  },
  {
    $inc: {
      quantity: 10
    }
  }
)
```

Operator `$rename` dapat digunakan untuk mengubah nama field dalam

suatu document. Berikut ini adalah contoh penggunaannya untuk mengubah nama field `product_name` menjadi `name` pada semua produk:

```
db.products.updateMany(
  {},
  {
    $rename: {
      product_name: "name"
    }
  }
)
``
```

GridFS

GridFS adalah sebuah spesifikasi untuk menyimpan dan mengambil file yang berukuran besar pada MongoDB. Dalam MongoDB, file yang berukuran besar dapat dipecah menjadi beberapa bagian atau chunk yang kemudian disimpan pada collection yang berbeda dengan dokumen utama. GridFS juga mendukung streaming file, sehingga file dapat diambil atau dikirim dalam potongan-potongan kecil. Hal ini membuat GridFS sangat berguna untuk aplikasi yang membutuhkan penyimpanan dan pengambilan file dengan ukuran yang besar. GridFS juga menyimpan metadata dari setiap file yang disimpan, seperti nama file, tipe file, dan ukuran file.

Mengupload File

Untuk mengupload file menggunakan GridFS, kita dapat menggunakan perintah `mongofiles` pada MongoDB Shell. Contoh di bawah ini akan mengupload file dari direktori lokal ke database MongoDB. Setelah file diupload, MongoDB akan menyimpan file tersebut pada dua koleksi yaitu `fs.files` dan `fs.chunks`:

```
mongofiles put /lokasi/ke/file
```

Mengambil File

Untuk mengambil file dari database menggunakan GridFS, kita dapat menggunakan perintah `mongofiles` pada MongoDB Shell. Perintah di bawah ini akan mengambil file dengan nama filename dari database MongoDB dan menyimpannya pada direktori lokal:

```
mongofiles get filename
```

Melihat Daftar File

Untuk melihat daftar file yang tersimpan pada database menggunakan GridFS, kita dapat menggunakan perintah `mongofiles` pada MongoDB Shell:

```
mongofiles list
```

Menghapus File

Untuk menghapus file dari database menggunakan GridFS, kita dapat menggunakan perintah `mongofiles` pada MongoDB Shell:

```
mongofiles delete filename
```

Perintah di atas akan menghapus file dengan nama `filename` dari database MongoDB.

Capped collections

Capped collections adalah salah satu fitur pada MongoDB yang memungkinkan kita untuk membuat collection dengan ukuran maksimal tertentu dan data yang lebih lama secara otomatis akan dihapus saat ukuran maksimal tersebut tercapai. Fitur ini berguna untuk mengumpulkan data yang berkaitan dengan log atau data sensor yang dihasilkan secara terus-menerus tanpa perlu khawatir kehabisan ruang penyimpanan.

Membuat Capped Collections

Untuk membuat capped collections, kita dapat menggunakan perintah `createCollection` dengan parameter `capped: true` dan `size` untuk menentukan ukuran maksimal collection tersebut. Contoh dibawah ini akan membuat capped collection `logs` dengan ukuran maksimal 1 MB:

```
db.createCollection("logs", {
  capped: true,
  size: 1000000
})
```

Setelah collection berhasil dibuat, kita dapat mengaksesnya dengan perintah `find` seperti biasa.

Memperbarui Capped Collections

Kita juga dapat memperbarui ukuran maksimal atau konfigurasi lainnya pada capped collections menggunakan perintah `collMod`. Dibawah ini akan memperbarui ukuran maksimal collection `logs` menjadi 2 MB

```
db.runCommand({
  collMod: "logs",
  size: 2000000
})
```

Menghapus Capped Collections

Capped collections dapat dihapus seperti collection lainnya menggunakan perintah `drop`:

```
db.logs.drop()
```

Capped collections dapat membantu mempermudah pengelolaan data pada MongoDB dengan cara yang lebih efektif dan efisien. Namun, perlu diingat bahwa penggunaan capped collections harus dipertimbangkan dengan cermat terlebih dahulu terutama dalam hal ukuran maksimal dan jumlah data yang dihasilkan agar tidak mengalami kekurangan penyimpanan atau kehilangan data yang penting.

Backup dan Restore

Backup dan restore adalah proses penting dalam pengelolaan database. Backup digunakan untuk membuat salinan data yang disimpan dalam collection agar dapat diakses kembali jika terjadi kehilangan atau kerusakan data. Sedangkan, restore digunakan untuk mengembalikan data dari salinan yang dibuat sebelumnya.

Backup

MongoDB dump adalah fitur bawaan dari MongoDB yang dapat digunakan untuk melakukan backup database. Perintah `mongodump` dapat digunakan untuk membuat salinan data dari database ke file BSON atau JSON.

```
mongodump --db <nama_database> --out <direktori_tujuan>
```

Restore

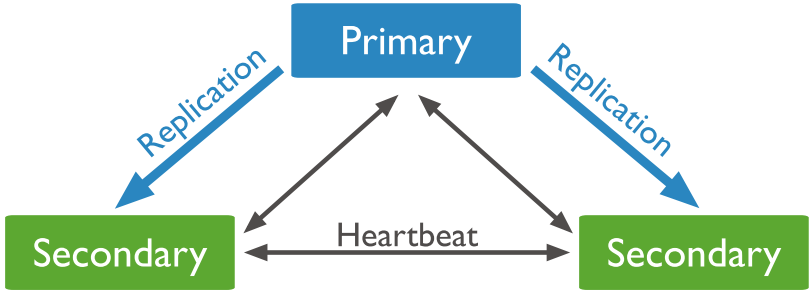
Setelah melakukan backup, kita dapat menggunakan data yang disimpan untuk melakukan restore. Perintah `mongorestore` dapat digunakan untuk melakukan restore data dari file BSON atau JSON.

```
mongorestore --db <nama_database> <direktori_sumber>
```

Dalam pengelolaan database, backup dan restore adalah proses penting untuk menjaga keamanan dan ketersediaan data. Dengan melakukan backup secara teratur, kita dapat meminimalkan risiko kehilangan atau kerusakan data, serta memudahkan proses recovery jika terjadi kejadian yang tidak diinginkan.

Replication

Replication adalah fitur yang memungkinkan untuk melakukan replikasi atau duplikasi data pada beberapa node atau server MongoDB. Dengan melakukan replikasi data, maka akan memungkinkan untuk membuat cadangan data (backup), meningkatkan toleransi kesalahan (fault tolerance), serta memungkinkan untuk mengakses data secara lokal.



Cara Kerja

Pada MongoDB, replication dilakukan dengan cara membuat replica set yang terdiri dari beberapa server atau node. Pada replica set, terdapat satu node utama (primary node) dan beberapa node sekunder (secondary node). Data pada primary node akan secara otomatis diteruskan atau disalin ke node sekunder, sehingga terjadi duplikasi data yang identik di setiap node.

Apabila terjadi kegagalan pada primary node, maka salah satu node sekunder akan secara otomatis dipilih untuk menggantikan primary node tersebut. Proses penggantian ini disebut dengan failover. Setelah primary node kembali online, maka node tersebut akan menjadi secondary node dan secara otomatis menyesuaikan diri dengan data yang ada di primary node.

Konfigurasi

Untuk melakukan konfigurasi replication pada MongoDB, langkah-langkah yang dapat dilakukan adalah sebagai berikut:

Mengaktifkan replica set

Pertama-tama, aktifkan replica set pada node yang ingin dijadikan primary node, dengan menjalankan perintah berikut:

```
rs.initiate()
```

Setelah menjalankan perintah tersebut, node akan menjadi primary node dan akan secara otomatis membuat sebuah replica set.

Menambahkan node sekunder

Selanjutnya, tambahkan node sekunder ke dalam replica set dengan menjalankan perintah `rs.add()` pada primary node. Perintah ini harus dijalankan pada setiap node sekunder yang ingin ditambahkan:

```
rs.add("node2:27017")
```

Node sekunder akan secara otomatis terhubung dengan primary node dan mulai melakukan replikasi data.

Mengecek status replica set

Untuk mengecek status replica set, jalankan perintah di bawah ini pada primary node:

```
rs.status()
```

Perintah ini akan menampilkan status replica set beserta informasi tentang setiap node dalam replica set.

Menghapus node

Jika diperlukan, Kita dapat menghapus node sekunder dari replica set dengan menjalankan perintah di bawah ini pada primary node:

```
rs.remove("node2:27017")
```

Mematikan replication

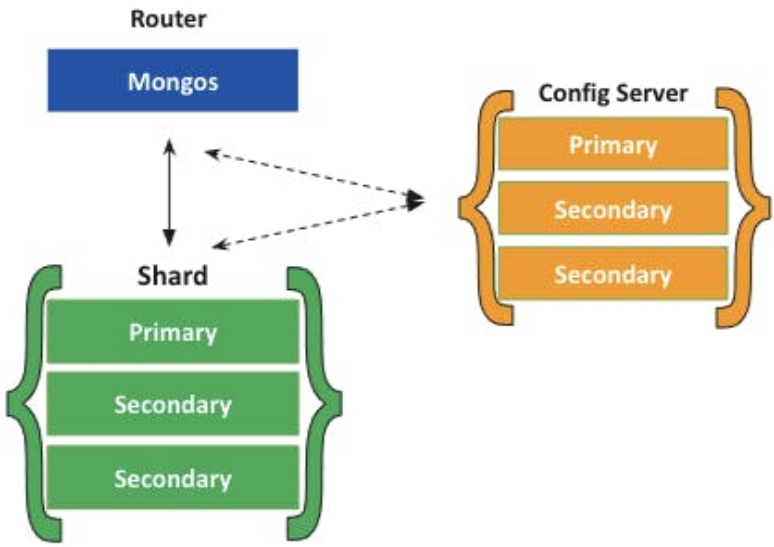
Jika Kita ingin mematikan replication sementara waktu, Kita dapat menjalankan perintah `rs.stepDown()` pada primary node. Perintah ini akan menurunkan primary node menjadi secondary node dan memilih satu dari node sekunder sebagai primary node.

```
rs.stepDown()
```

Sharding

Sharding merupakan sebuah teknik yang digunakan untuk memecah data menjadi beberapa bagian atau shard dan menyimpannya pada beberapa node atau server yang berbeda. Teknik ini memungkinkan MongoDB untuk menangani jumlah data yang sangat besar dan meningkatkan skalabilitas database.

Dalam sharding, setiap shard berisi sebagian kecil data dan dapat beroperasi secara independen, sehingga dapat mengurangi beban pada satu server dan meningkatkan performa database secara keseluruhan. MongoDB juga menyediakan mekanisme untuk mengembangkan kapasitas sharding dengan mudah, sehingga memungkinkan perusahaan atau organisasi untuk mengelola data yang terus bertambah dengan lebih efisien.



Cara Kerja

MongoDB melakukan sharding dengan cara mempartisi data menjadi

beberapa bagian atau chunk dan mendistribusikannya ke dalam beberapa shard. Setiap shard berisi sejumlah chunk yang saling terkait.

Ketika sebuah permintaan query dilakukan, MongoDB akan mengirimkan query tersebut ke koordinator atau mongos, yang akan menentukan shard mana yang harus digunakan untuk memenuhi permintaan tersebut. Mongos akan mengembalikan hasil query kepada aplikasi setelah mendapatkan semua hasil dari masing-masing shard.

Komponen

Sebelum kita membahas cara konfigurasi sharding, kita perlu memahami terlebih dahulu beberapa komponen yang diperlukan dalam sharding pada MongoDB, yaitu:

Config Server

Config server digunakan untuk menyimpan metadata dan konfigurasi sharding pada MongoDB. MongoDB biasanya menggunakan tiga server config untuk memastikan ketersediaan dan keandalan data.

Shard Server

Shard server digunakan untuk menyimpan data secara terpisah dan memiliki kemampuan replikasi. Setiap shard harus berjalan pada beberapa mesin agar data tetap aman dan terdistribusi dengan baik.

Mongos

Mongos berfungsi sebagai koordinator antara aplikasi dan shard server. Mongos menerima permintaan query dari aplikasi dan mengirimkan permintaan tersebut ke shard server yang sesuai.

Konfigurasi

Persiapkan Server Config

Sebelum melakukan sharding, pastikan server config telah terkonfigurasi dengan baik. Pada server config, kita harus menentukan opsi `sharding` pada konfigurasi MongoDB, yaitu:

```
sharding:
  clusterRole: configsvr
```

Persiapkan Shard Server

Setelah server config siap, kita perlu menyiapkan shard server. Shard server harus berjalan pada beberapa mesin dan memiliki kemampuan replikasi untuk menjaga data tetap aman. Untuk mempersiapkan shard server, kita harus melakukan beberapa hal, yaitu:

- Menginstal MongoDB pada semua mesin yang akan digunakan sebagai shard server.
- Menentukan opsi sharding pada konfigurasi MongoDB pada setiap mesin shard server, yaitu:

  ```
  sharding:
    clusterRole: shardsvr
  ```

- Menjalankan server MongoDB pada setiap mesin yang akan digunakan sebagai shard server.

Persiapan Mongos

Setelah server config dan shard server siap, kita perlu mempersiapkan Mongos. Mongos berfungsi sebagai koordinator antara aplikasi dan shard server. Untuk mempersiapkan Mongos, kita harus melakukan beberapa hal, yaitu:

- Menginstal MongoDB pada semua mesin yang akan digunakan sebagai Mongos.
- Menentukan opsi sharding pada konfigurasi MongoDB pada mesin Mongos, yaitu:

```
sharding:
  configDB:
configserver1.example.net:27019,configserver2.example.
net:27019,configserver3.example.net:27019
```

- Menjalankan Mongos pada mesin yang telah diinstal MongoDB.

Menambahkan Shard ke dalam Cluster

Setelah Mongos siap, kita perlu menambahkan shard ke dalam cluster MongoDB. Untuk menambahkan shard, kita harus melakukan beberapa hal, yaitu:

- Jalankan perintah `mongos` pada mesin Mongos, kemudian masuk ke shell MongoDB dengan perintah `mongo`.
- Tambahkan shard dengan perintah `sh.addShard()`, misalnya: `sh.addShard("shard1.example.net:27017")`.
- Ulangi langkah ini untuk setiap shard server yang ingin ditambahkan ke dalam cluster.

Membuat Database

Setelah semua shard ditambahkan ke dalam cluster, kita dapat membuat database yang akan digunakan. Database dapat dibuat dengan perintah `use <nama_database>` pada shell MongoDB. Setelah database dibuat, kita perlu menentukan key yang akan digunakan untuk sharding pada collection yang akan dibuat. Key tersebut harus berbeda dengan `_id`, karena `_id` sudah digunakan oleh MongoDB sebagai default key untuk sharding.

Sharding pada Collection

Setelah database dan collection dibuat, kita dapat melakukan sharding pada collection. Sharding dapat dilakukan dengan perintah `sh.enableSharding(<nama_database>)` untuk mengaktifkan sharding pada database, dan `sh.shardCollection("<nama_database>.<nama_collection>", <key_sharding>)` untuk mengaktifkan sharding pada collection.

Misalnya, untuk mengaktifkan sharding pada collection `product` pada database `store` dengan menggunakan key `category`, kita dapat menggunakan perintah berikut:

```
sh.enableSharding("store")
sh.shardCollection("store.product", {"category": 1})
```

Setelah sharding diaktifkan pada collection, MongoDB secara otomatis akan mempartisi data ke dalam beberapa shard yang tersedia. Ketika kita menambahkan atau mengakses data pada collection tersebut, MongoDB akan secara otomatis memindahkan data ke shard yang tepat.

> Dalam memilih menggunakan sharding, perusahaan atau organisasi harus memperhatikan beberapa hal seperti kapasitas data, performa database, dan ketersediaan server. Dengan memahami konsep dan cara kerja sharding, pengguna MongoDB dapat meningkatkan performa dan efisiensi database pada perusahaan atau organisasinya.

Penutup

Dengan mempelajari buku ini, diharapkan pembaca dapat memahami dasar-dasar MongoDB dan bagaimana cara menggunakannya untuk membangun aplikasi dengan data yang skalabel dan cepat. MongoDB adalah salah satu solusi database NoSQL yang populer dan sangat berguna untuk memenuhi kebutuhan aplikasi modern yang kompleks dan membutuhkan skala besar.

Namun, perlu diingat bahwa penggunaan NoSQL, khususnya MongoDB, tidak untuk menggantikan database relasional seperti MySQL atau PostgreSQL. Meskipun MongoDB memiliki kelebihan dalam hal skala, kecepatan, dan fleksibilitas, namun database relasional masih memiliki tempatnya tersendiri, terutama dalam aplikasi yang membutuhkan integritas data yang tinggi dan keamanan yang kuat.

Oleh karena itu, sebagai pengembang atau arsitek perangkat lunak, penting untuk mempertimbangkan kebutuhan dan karakteristik aplikasi dan data yang akan diolah, sebelum memutuskan untuk menggunakan database NoSQL atau relasional. Dalam beberapa kasus, solusi hybrid dari kedua jenis database dapat menjadi pilihan yang tepat untuk memenuhi kebutuhan yang beragam dan kompleks.

Terakhir, semoga buku ini dapat membantu pembaca dalam mempelajari MongoDB dan memberikan wawasan baru tentang solusi database modern yang dapat membantu meningkatkan efisiensi dan performa aplikasi. Terima kasih telah membaca buku ini dan selamat menjelajahi dunia MongoDB!

"MongoDB adalah jembatan antara data yang kompleks dan solusi yang mudah. Sebuah teknologi yang memudahkan kita untuk memanipulasi data dengan skala dan kecepatan yang luar biasa."
~ Wahyu Kristianto