

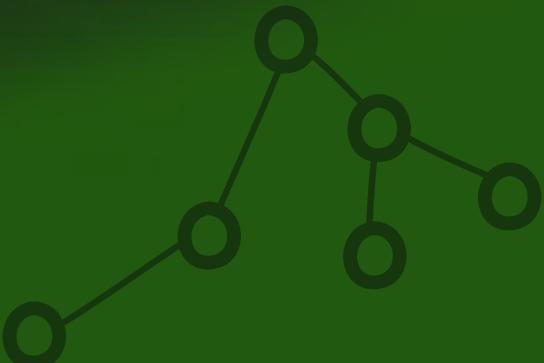
BASIS DATA LANJUT

JEMY ARIESWANTO

SEPTIAN CAHYADI

ANTON SUKAMTO

SUCI SUTJIPTO



Fakultas Informatika dan Pariwisata
Institut Bisnis dan Informatika Kesatuan

KATA PENGANTAR

Puji syukur kehadiran Tuhan Yang Maha Kuasa, yang telah memberikan rahmat-Nya sehingga Modul Ajar Basis Data Lanjut untuk mahasiswa/i Program Studi Teknologi Informasi Fakultas Informatika dan Pariwisata Institut Bisnis dan Informatika Kesatuan ini dapat diselesaikan dengan sebaik-baiknya.

Modul Ajar ini dibuat sebagai pedoman dalam melakukan kegiatan pembelajaran mata kuliah Basis Data Lanjut pada Program Studi Teknologi Informasi Fakultas Informatika dan Pariwisata Institut Bisnis dan Informatika Kesatuan. Modul Ajar ini diharapkan dapat membantu mahasiswa/i dalam mempersiapkan dan melaksanakan praktikum dengan lebih baik, terarah, dan terencana. Pada setiap topik telah ditetapkan tujuan pelaksanaan kegiatan pembelajaran dan semua kegiatan yang harus dilakukan oleh mahasiswa/i serta teori singkat untuk memperdalam pemahaman mahasiswa/i mengenai materi yang dibahas.

Penyusun meyakini bahwa dalam pembuatan Modul Ajar Basis Data Lanjut ini masih jauh dari sempurna. Oleh karena itu penyusun mengharapkan kritik dan saran yang membangun guna penyempurnaan modul ajar ini dimasa yang akan datang. Akhir kata, penyusun mengucapkan banyak terima kasih kepada semua pihak yang telah membantu baik secara langsung maupun tidak langsung.

Bogor, September 2023

Penyusun

DAFTAR ISI

COVER

KATA PENGANTAR i

DAFTAR ISI ii

DAFTAR GAMBAR xiii

MODUL 1 SQL & MySQL 1

 1.1. Tujuan Praktikum 1

 1.2. Dasar Teori 1

 1.2.1. Pengenalan SQL & MySQL 1

 1.2.2. *Data Definition Language* (DDL) 2

 1.2.3. *Data Manipulation Language* (DML) 3

 1.2.4. *Data Query Language* (DQL) 3

 1.2.5. *Data Control Language* (DCL) 3

 1.2.6. *Entity Relationship Diagram* - ERD 3

 1.2.7. Kapan perlu Menggambar ERD 4

 1.2.8. Notasi dalam ERD 4

 1.3. *Software* 8

 1.4. Tahapan Kerja 8

 1.4.1. Membuat ERD 8

 1.4.2. Membuat database 11

 1.5. Tugas dan Latihan 14

MODUL 2 DATA MANIPULATION LANGUAGE (DML) 15

 2.1. Tujuan Praktikum 15

 2.2. Dasar Teori 15

 2.2.1. Manipulasi Data MySQL 15

| | |
|---|----|
| 2.3. <i>Software</i> | 16 |
| 2.4. Tahapan Kerja | 16 |
| 2.4.1 Persiapkan database..... | 16 |
| 2.4.2. Membuat database | 21 |
| 2.5. Tugas dan Latihan | 25 |
| MODUL 3 NOSQL & MONGODB..... | 27 |
| 3.1. Tujuan Praktikum | 27 |
| 3.2. Dasar Teori..... | 27 |
| 3.2.1. Mengenal NoSQL..... | 27 |
| 3.2.2. Tipe-tipe NoSQL | 28 |
| 3.2.3. MongoDB | 28 |
| 3.2.3. Terminologi | 29 |
| 3.2.4. Perintah dasar | 29 |
| 3.3. <i>Software</i> | 30 |
| 3.4. Tahapan Kerja | 30 |
| 3.4.1. Instalasi MongoDB..... | 30 |
| 3.4.2. <i>Set up Environment Variable</i> | 32 |
| 3.4.3. Persiapan Basisdata | 35 |
| 3.4.4. Manipulasi Data..... | 37 |
| 3.5. Tugas dan Latihan | 42 |
| MODUL 4 TIPE DATA DAN RELASI PADA MONGODB | 43 |
| 4.1. Tujuan Praktikum | 43 |
| 4.2. Dasar Teori..... | 43 |
| 4.2.1. Tipe Data MongoDB | 43 |
| 4.2.2. Relasi MongoDB | 45 |
| 4.2.3. Lookup..... | 45 |
| 4.2.3. Validasi..... | 46 |

| | |
|--|-----------|
| 4.3. <i>Software</i> | 47 |
| 4.4. Tahapan Kerja | 47 |
| 4.4.1. Tipe Data | 47 |
| 4.4.2. Lookup..... | 48 |
| 4.4.3. Validasi..... | 49 |
| 4.5. Tugas dan Latihan | 51 |
| MODUL 5 DATA MODEL DAN QUERY | 52 |
| 5.1. Tujuan Praktikum..... | 52 |
| 5.2. Dasar Teori..... | 52 |
| 5.2.1. Data Model MongoDB | 52 |
| 5.2.2. JSON dan BSON | 53 |
| 5.2.3. Konsep Method, Filter, Operator..... | 54 |
| 5.4. <i>Software</i> | 54 |
| 5.5. Tahapan Kerja | 54 |
| 5.4.1. Menjalankan MongoDB di Compass dan Command Prompt | 54 |
| 5.4.2. Query Dasar | 55 |
| 5.4.3. Query Kompleks..... | 56 |
| 5.5. Tugas dan Latihan | 68 |
| MODUL 6 PROJECTION DAN AGGREGATION | 69 |
| 6.1. Tujuan Praktikum..... | 69 |
| 6.2. Dasar Teori..... | 69 |
| 6.2.1. <i>Update</i> dan <i>Save Document</i> MongoDB..... | 69 |
| 6.2.2. <i>Remove Document</i> MongoDB | 69 |
| 6.2.3. <i>Limit Record</i> MongoDB | 70 |
| 6.2.4. <i>Indexing</i> MongoDB | 70 |
| 6.2.5. <i>Aggregation</i> MongoDB | 72 |
| 6.4. <i>Software</i> | 74 |

| | |
|--|-----------|
| 6.5. Tahapan Kerja | 74 |
| 6.4.1. Menjalankan MongoDB di Compass dan Command Prompt | 74 |
| 6.4.2. <i>Update</i> dan <i>save document</i> | 75 |
| 6.4.3. <i>Remove Document</i> | 75 |
| 6.4.4. <i>Limit Document</i> | 75 |
| 6.4.5. <i>Index</i> | 76 |
| 6.4.6. <i>Aggregation</i> | 77 |
| 6.4.7. <i>Pipeline</i> | 79 |
| 6.5. Tugas dan Latihan | 82 |
| MODUL 7 MAPREDUCE & TRANSACTION | 83 |
| 7.1. Tujuan Praktikum..... | 83 |
| 7.2. Dasar Teori..... | 83 |
| 7.2.1. <i>Map Reduce</i> | 83 |
| 7.2.2. <i>Transactions</i> | 84 |
| 7.2.3. Gambaran umum transaksi | 84 |
| 7.2.4. Penggunaan transaksi | 84 |
| 7.2.4. Keterbatasan transaksi | 85 |
| 7.3. <i>Software</i> | 85 |
| 7.4. Tahapan Kerja | 85 |
| 7.4.1. MapReduce..... | 85 |
| 7.4.2. <i>Setup</i> akun MongoDb atlas dan membuat cluster | 86 |
| 7.4.3. Mengenal Python..... | 88 |
| 7.4.4. Koneksi Python dengan mongodb..... | 89 |
| 7.4.5. <i>Transaction</i> mongodb..... | 89 |
| 7.5. Tugas dan Latihan | 90 |
| MODUL 8 SECURITY MONGODB | 91 |
| 8.1. Tujuan Praktikum..... | 91 |

| | |
|---|-----|
| 8.2. Dasar Teori..... | 91 |
| 8.2.1. <i>Security MongoDB</i> | 91 |
| 8.2.2. <i>Authentication</i> | 91 |
| 8.2.3. <i>Authorization</i> | 92 |
| 8.2.4. TLS/SSL (<i>Transport Encryption</i>)..... | 94 |
| 8.3. <i>Software</i> | 95 |
| 8.4. Tahapan Kerja | 95 |
| 8.4.1. <i>Authentication MongoDB</i> | 95 |
| 8.4.2. <i>Authorization MongoDB</i> | 96 |
| 8.5. Tugas dan Latihan | 100 |
| MODUL 9 MENGENAL GRAPH DATABASE..... | 101 |
| 9.1. Tujuan Praktikum..... | 101 |
| 9.2. Dasar Teori..... | 101 |
| 9.2.1. Graph Database..... | 101 |
| 9.2.2. RDBMS Vs Graph Database | 101 |
| 9.2.3. Neo4j..... | 102 |
| 9.2.4. Kelebihan dari Neo4j | 102 |
| 9.2.5. Fitur dari Neo4j..... | 103 |
| 9.3. <i>Software</i> | 104 |
| 9.4. Tahapan Kerja | 104 |
| 9.4.1. Instalasi Neo4j | 104 |
| 9.4.2. Membuat Graph..... | 105 |
| 9.5. Tugas dan Latihan | 112 |
| MODUL 10 CYPHER CLAUSE DI GRAPH..... | 113 |
| 10.1. Tujuan Praktikum..... | 113 |
| 10.2. Dasar Teori..... | 113 |
| 10.2.1. MATCH | 113 |

| | |
|--|------------|
| 10.2.2. OPTIONAL MATCH | 113 |
| 10.2.3. FOREACH | 114 |
| 10.2.4. UNION | 114 |
| 10.2.5. WITH | 114 |
| 10.2.6. DETACH DELETE | 115 |
| 10.3. <i>Software</i> | 115 |
| 10.4. Tahapan Kerja | 115 |
| 10.5. Tugas dan Latihan | 125 |
| MODUL 11 SENARAI DAN MAP DI GRAPH | 126 |
| 11.1. Tujuan Praktikum | 126 |
| 11.2. Dasar Teori | 126 |
| 11.2.1. Senarai | 126 |
| 11.2.2. Map | 127 |
| 11.3. <i>Software</i> | 128 |
| 11.4. Tahapan Kerja | 128 |
| 11.5. Tugas dan Latihan | 136 |
| MODUL 12 FUNGSI-FUNGSI AGREGAT | 138 |
| 12.1. Tujuan Praktikum | 138 |
| 12.2. Dasar Teori | 138 |
| 12.2.1. Fungsi agregat | 138 |
| 12.2.2. Persentil | 139 |
| 12.3. <i>Software</i> | 139 |
| 12.4. Tahapan Kerja | 139 |
| 12.5. Tugas dan Latihan | 143 |
| DAFTAR PUSTAKA | 144 |

DAFTAR GAMBAR

| | |
|--|----|
| Gambar 1. SQL commands..... | 2 |
| Gambar 2. Contoh entitas | 4 |
| Gambar 3. Contoh atribut entitas | 5 |
| Gambar 4. Contoh Primary Key | 6 |
| Gambar 5. Contoh symbol relasi | 6 |
| Gambar 6. Contoh relasi one to one antara table user dan UserProfile | 6 |
| Gambar 7. Simbol garis one to many..... | 7 |
| Gambar 8. Contoh relasi one to many dari table departemen ke student | 7 |
| Gambar 9. Contoh relasi many to many | 7 |
| Gambar 10. Contoh ERD | 8 |
| Gambar 23. Sample database diagram..... | 15 |
| Gambar 28. Vertical dan Horizontal Scaling..... | 28 |
| Gambar 29. Tipe SQL dan NoSQL database..... | 28 |
| Gambar 30. Terminologi..... | 29 |
| Gambar 31. Perintah dasar MongoDB | 30 |
| Gambar 33. Instalasi MongoDB | 31 |
| Gambar 36. Opsi installasi MongoDB Compass | 32 |
| Gambar 37. Aplikasi MongoDB | 32 |
| Gambar 61. Model database MongoDB | 52 |
| Gambar 62. Gambar Map Reducing | 83 |
| Gambar 63. Pustaka native TLS/SSL OS | 94 |

MODUL 1

SQL & MySQL

1.1. Tujuan Praktikum

1. Mengenal Bahasa SQL
2. Mahasiswa memahami konsep ERD
3. Mahasiswa mampu menerapkan DDL dan DML

1.2. Dasar Teori

1.2.1. Pengenalan SQL & MySQL

Database atau basis data merupakan kumpulan berbagai informasi yang disimpan secara sistematis pada komputer agar informasi tersebut dapat diolah kembali dengan mudah. Dalam pengembangan sebuah website, *Database* berperan penting sebagai penyimpanan dan penyedia informasi yang ditampilkan ke pengguna.

Semakin kompleks logika bisnis dan semakin banyak jumlah pengguna sebuah website maka akan membutuhkan *database* yang baik dalam menyimpan dan menyediakan data.

Kriteria database yang baik ditinjau dari beberapa faktor yakni:

- kecepatan
- performa
- akurasi
- availability
- terhindar dari duplikasi dan inkonsistensi data
- keamanan

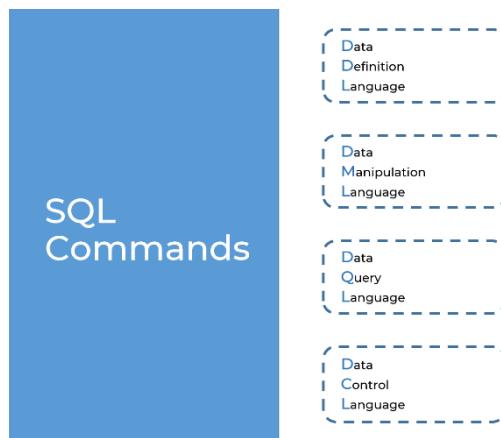
Database ditinjau dari sifat relasional nya terbagi menjadi dua yaitu: ***relational database*** dan ***non-relational database***. Pada kesempatan kali ini kita akan mempelajari tentang *relational database* SQL.

Structured Query Language (SQL) adalah sebuah bahasa yang digunakan untuk mengakses data dalam basis data relasional. Bahasa ini secara *de facto* merupakan bahasa standar yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua server basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya. wikipedia

Pada awal tahun 1970-an, tercetus ide untuk mengembangkan basis data relasional. Niatnya adalah membuat standar bahasa yang dapat digunakan oleh siapa saja. Karena hal tersebut munculah istilah SEQUEL (*Structured English Query Language*). Sebelum adanya

SEQUEL ini, pada tahun 1969 IBM terlebih dahulu melahirkan SQUARE sebagai cikal bakal dari SEQUEL. Nama tersebut lalu disingkat menjadi SQL karena adanya permasalahan merk dagang dengan perusahaan lain.

Pada tahun 1986, *ANSI (American National Standards Institute)*, sebuah badan yang membuat standarisasi merancang sebuah standar untuk SQL. Tidak lama setelahnya, *ISO (International Organization for Standardization)* juga mengeluarkan standar untuk SQL. Ini dikerjakan dengan tujuan untuk menyeragamkan query yang digunakan di pada SQL dikarenakan perkembangan dari SQL yang begitu cepat sehingga banyak sekali perusahaan yang melakukan pengembangan sendiri SQL dengan menambahkan beberapa fitur tambahan. Walaupun begitu, perintah-perintah dasar masih memiliki kesamaan.



Gambar 1. SQL commands

1.2.2. Data Definition Language (DDL)

Perintah ini digunakan untuk membuat struktur sebuah *database*. Terdapat lima perintah utama, diantaranya:

- Perintah **Create**: sebuah perintah yang bisa kamu gunakan ketika membuat sebuah basisdata maupun tabel baru.
- Perintah **Alter**: biasa digunakan ketika seseorang ingin mengubah struktur tabel yang sebelumnya sudah ada.
- Perintah **Rename**: dapat kamu gunakan untuk mengubah sebuah nama di sebuah tabel ataupun kolom yang ada.
- Perintah **Drop**: Bisa kamu gunakan dalam menghapus baik itu berupa *database*, table maupun kolom hingga index.

- Perintah **Show**: perintah DDL ini digunakan untuk menampilkan sebuah data tabel atau basis data

1.2.3. Data Manipulation Language (DML)

Perintah dasar SQL ini bertujuan untuk memanipulasi data yang ada dalam sebuah *database*. Beberapa perintah dalam DML di antaranya *adalah insert, update, dan delete*.

- Perintah **Insert**: Kamu bisa menggunakan perintah ini untuk memasukkan sebuah *record* baru di dalam sebuah tabel *database*.
- Perintah **update**: Ini dapat kamu gunakan ketika ingin melakukan pembaruan data di sebuah tabel. Contohnya saja jika ada kesalahan ketika memasukkan sebuah record. Kamu tidak perlu menghapusnya dan bisa diperbaiki menggunakan perintah ini.
- Perintah **Delete**: Perintah DML ini dapat digunakan ketika kamu ingin menghapus sebuah record yang ada dalam sebuah tabel.

1.2.4. Data Query Language (DQL)

Perintah yang digunakan untuk melakukan query data pada SQL.

- Perintah **Select**: Pada perintah ini kamu dapat menggunakannya dalam menampilkan maupun mengambil sebuah data pada tabel. Data yang diambil pun tidak hanya terbatas pada satu jenis saja melainkan lebih dari satu tabel dengan memakai relasi.

1.2.5. Data Control Language (DCL)

Perintah yang digunakan untuk melakukan query data pada SQL.

- Perintah **Select**: Pada perintah ini kamu dapat menggunakannya dalam menampilkan maupun mengambil sebuah data pada tabel. Data yang diambil pun tidak hanya terbatas pada satu jenis saja melainkan lebih dari satu tabel dengan memakai relasi.

1.2.6. Entity Relationship Diagram - ERD

Saat kita mengembangkan perangkat lunak atau pun mengerjakan sebuah proyek yang menggunakan banyak data sehingga membutuhkan penggunaan database khusus maka kemampuan untuk dapat merancang ERD akan sangat membantu untuk menaikkan kapabilitas dan efisiensi sistem database kita.

ERD merupakan kepanjangan dari Entity Relationship Diagram, merupakan salah satu tipe diagram struktural yang digunakan pada saat kita mendesain database. pada ERD terdapat

beberapa simbol dan konektor yang digunakan untuk menggambarkan dua hal, yaitu semua Entitas dalam sistem dan hubungan antar entitas tersebut.

1.2.7. Kapan perlu Menggambar ERD

Beberapa hal yang menjadi alasan kenapa menggambar ERD itu perlu diantaranya adalah:

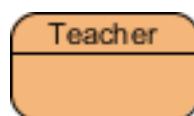
- Mengetahui bagaimana database tersebut bekerja akan sangat membantu, karena mengubah database, baik menambahkan atau mengurangi bukanlah perkara yang sederhana. Menghapus satu database bisa mempengaruhi database lain yang terhubung.
- Membantu untuk memperlihatkan gambaran secara luas tentang basisdata yang kita kerjakan.
- Memperlihatkan bagaimana setiap tabel pada database saling terhubung dan kolom dari setiap tabelnya
- Menjadi *blueprint* saat kita bekerja dalam sebuah tim sehingga jika ada anggota baru yang bergabung dapat langsung mempelajari desain dari basisdata kita

1.2.8. Notasi dalam ERD

Beberapa hal yang menjadi alasan kenapa menggambar ERD itu perlu diantaranya adalah:

Entitas

Entitas adalah sebuah objek yang merepresentasikan suatu baris data pada tabel. satu entitas akan memiliki karakteristik tersendiri berdasarkan atribut-atribut yang dimiliki. Contohnya pada sebuah database toko online terdapat entitas *pelanggan*, *profil*, *barang*, *transaksi*, dan *kategori*. Pada ERD, sebuah entitas ditunjukan dengan persegi tumpul.



Gambar 2. Contoh entitas

Atribut

Atribut adalah kolom-kolom data yang terdapat pada sebuah entitas dan berfungsi untuk mendeskripsikan karakteristik entitas tersebut. Di dalam sebuah entitas biasanya terdapat atribut kunci (*primary key*) yang merupakan pembeda antara satu entitas dengan entitas lainnya. Contohnya pada entitas pelanggan terdapat atribut *id_pelanggan*, *nama*, dan *email*.

Setiap atribut dapat kita definisikan tipe data nya. Misalnya adalah Entitas Kelas mungkin memiliki atribut guru, murid, dan pelajaran.

Pada saat mengubah ERD konseptual menjadi ERD sebenarnya ada baiknya kita mengecek atribut apa saja yang diperbolehkan pada suatu RDBMS. Misalnya, Oracle menggunakan varchar dimana sqlite menggunakan Text.

Contoh dari atribut ditunjukan pada gambar dibawah.

| Customer | | |
|----------|-------------|--------------|
| | ID | integer(10) |
| | First_Name | varchar(255) |
| | Last_Name | varchar(255) |
| | Address | varchar(255) |
| | Telephone | integer(10) |
| | Gender | char(1) |
| | Active | char(1) |
| | Email | varchar(50) |
| | Create_Date | date |
| | Last_Update | date |

Gambar 3. Contoh atribut entitas

Kunci/Key

Atribut Kunci merupakan salah satu cara untuk mengkategorikan atribut khusus pada suatu entitas. Atribut kunci ini biasanya digunakan sebagai dasar untuk menghubungkan (membuat relasi) antara satu entitas dengan entitas yang lain.



Gambar 4. Contoh Primary Key

Kardinalitas

Kardinalitas menggambarkan tentang asosiasi atau hubungan antara dua tabel atau lebih. Relasi kardinalitas yang paling umum digambarkan ada tiga, yaitu hubungan satu - satu, hubungan satu - banyak, atau hubungan banyak - banyak. Pada ERD biasanya relasi digambarkan dengan garis. Setiap satu jenis relasi memiliki jenis garis yang berbeda dengan jenis relasi lainnya.

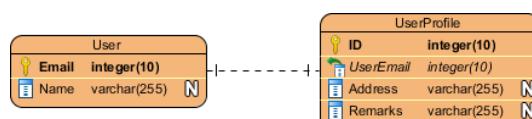
Hubungan Satu - Satu

Relasi *one to one* berarti satu entitas memiliki hubungan dengan satu entitas lainnya. Relasi ini digambarkan dengan garis seperti berikut



Gambar 5. Contoh symbol relasi

Misalnya pada kejadian dimana satu orang hanya bisa memiliki satu NIK.



Gambar 6. Contoh relasi one to one antara table user dan UserProfile

Hubungan Satu - Banyak

Relasi *One To Many* yaitu ketika satu entitas memiliki hubungan dengan banyak entitas lainnya. Contohnya dalam hal ini adalah di dalam satu kategori terdapat banyak barang yang

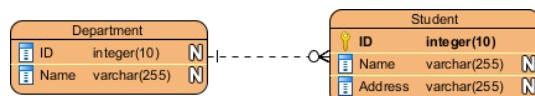
terkait dengan kategori tersebut. Sebaliknya, untuk satu barang dapat kita klasifikasikan ke satutu kategori saja.

Relasi ini biasanya digambarkan dengan simbol garis seperti berikut



Gambar 7. simbol garis one to many

Misalnya pada kejadian dimana satu fakultas dapat memiliki banyak murid akan tetapi satu murid hanya bisa terdaftar pada satu fakultas.



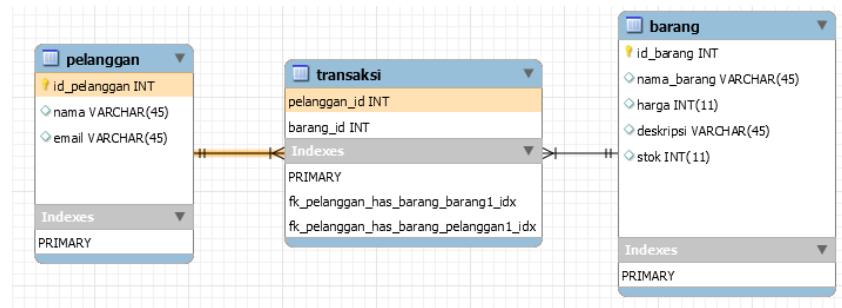
Gambar 8. Contoh relasi one to many dari table departemen ke student

Hubungan Banyak - Banyak

Relasi *Many To Many* adalah ketika banyak entitas terhubung dengan banyak entitas lainnya. Contohnya adalah seorang pelanggan dapat membeli banyak barang yang berbeda jenis. Sebaliknya satu jenis barang dapat dibeli oleh banyak pelanggan yang berbeda.

Untuk merepresentasikan hubungan antar dua entitas yang memiliki relasi ini maka dibutuhkan satu entitas tambahan yang biasa disebut dengan pivot. Dalam kaitannya dengan contoh antara entitas pelanggan dan barang maka entitas penghubungnya yaitu transaksi.

Misalnya pada kejadian dimana satu kelas dapat diikuti oleh banyak mahasiswa dan satu mahasiswa dapat mengikuti banyak kelas,

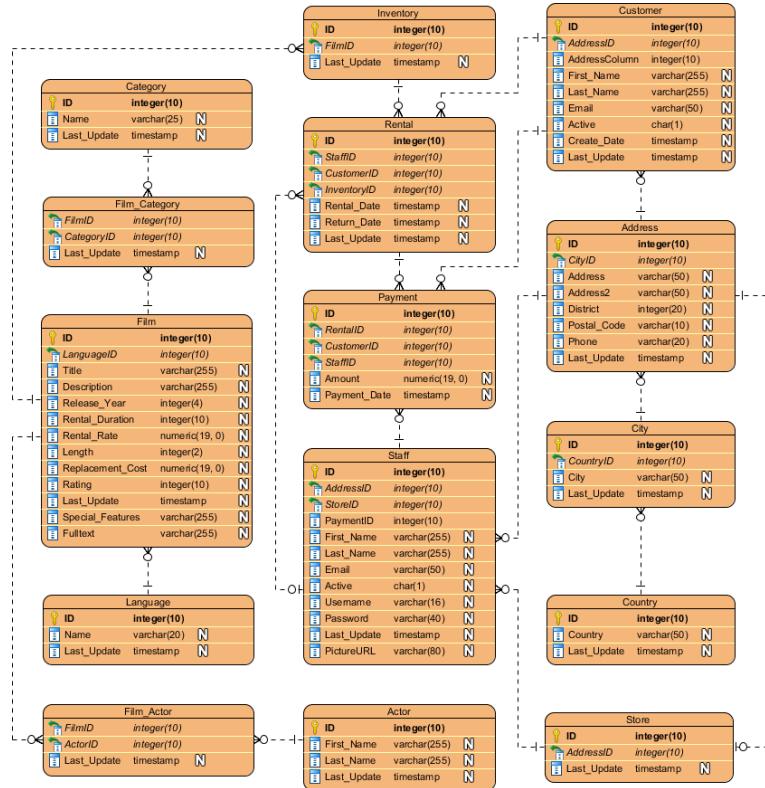


Gambar 9. Contoh relasi many to many

Pada entitas transaksi terdapat dua *foreign key* yaitu pelanggan_id dan barang_id yang terhubung masing-masing ke *primary key* pada entitas pelanggan dan entitas barang.

Contoh ERD

Sistem Penyewaan Film



Gambar 10. Contoh Erd

sumber : <https://cdn-images.visual-paradigm.com/guide/data-modeling/what-is-erd/13-erd-example-movie-rental-system.png>

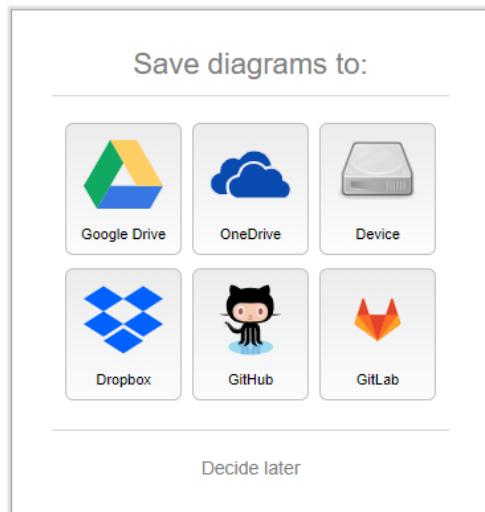
1.3. Software

1. Diagram
2. XAMPP

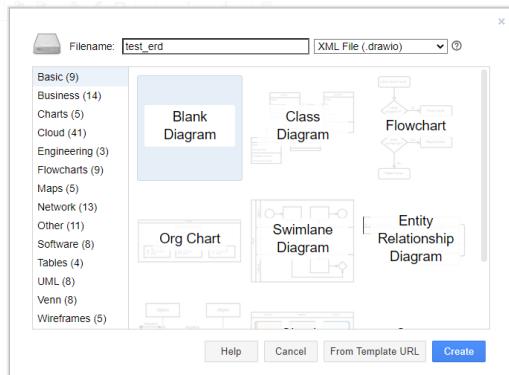
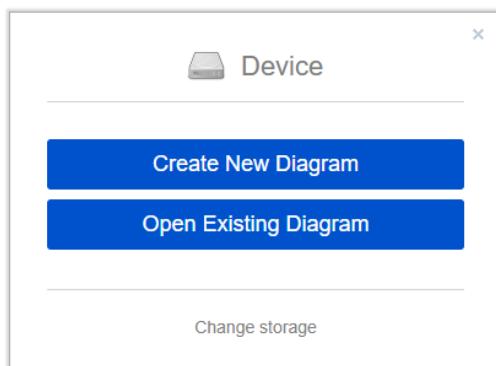
1.4. Tahapan Kerja

1.4.1. Membuat Erd

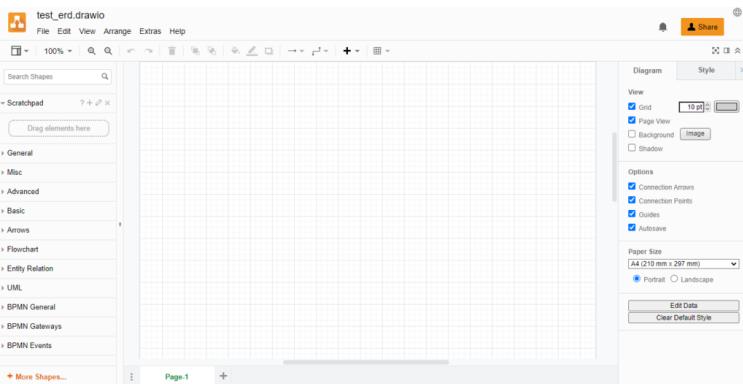
1. Buka website <https://app.diagrams.net/>. Saat membuka alamat tersebut kita akan diminta untuk memilih dimana kita akan menyimpan file xml kita. Untuk kesempatan kali ini kita akan menggunakan penyimpanan lokal kita.



pilih menu *create new diagram* dan pilih blank diagram.



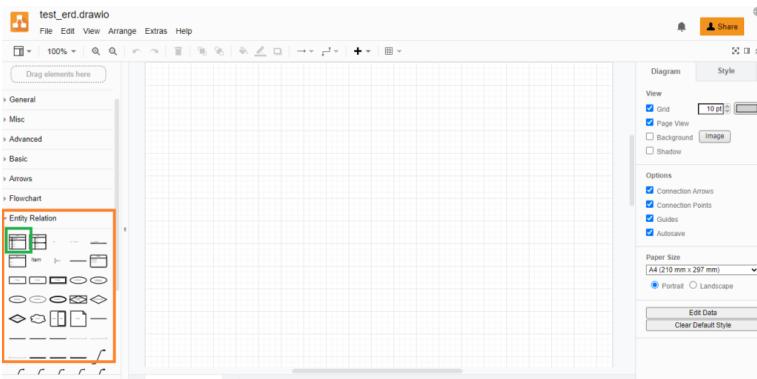
Maka akan muncul jendela seperti pada gambar di bawah



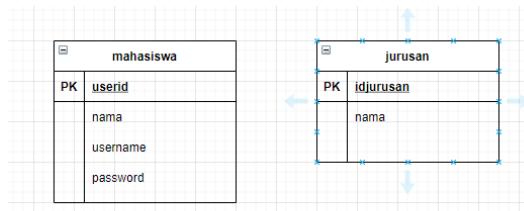
Kita akan mencoba untuk membuat ERD sederhana dengan dua tabel

- Mahasiswa - berisi id, nama, email, password
- Jurusan - berisi id, nama

Kita akan menggunakan sidebar kiri untuk memilih objek yang akan kita gunakan, kita pilih pada menu Entity Relation dan pilih yang ditandai dengan kotak hijau.

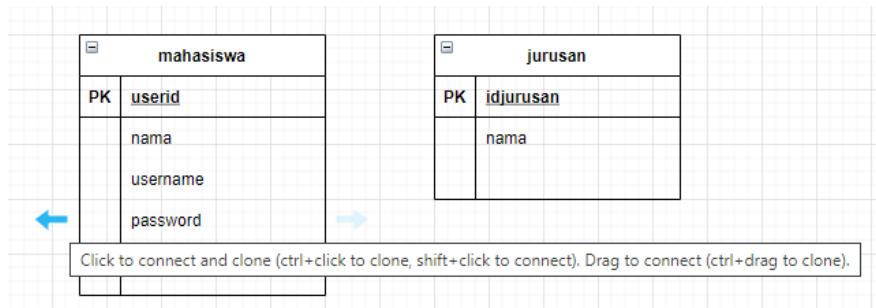


2. Setelah itu kita tambahkan dua buah tabel dan ubah nama tabel dan nama kolom kita.

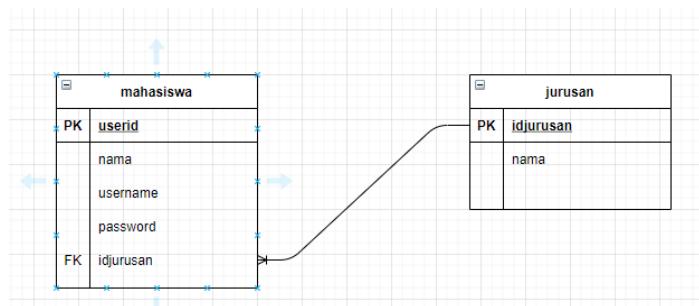


Sekarang kita akan membuat sebuah hubungan atau relasi antara dua buah tabel tersebut, relasi yang akan kita bangun berdasarkan pada hubungan antara PK (Primary Key) dan FK (Foreign Key). Karena kita tahu seorang mahasiswa akan terdaftar dalam satu jurusan maka kita akan membuat kolom FK baru pada tabel mahasiswa yang menghubungkan dengan PK pada tabel jurusan.

3. Untuk menambahkan kolom pada tabel kita dapat mengarahkan kursor kita pada kolom yang ingin ditambahkan kolom baru di bawahnya, arahkan ke kanan/kiri kolom tersebut hingga muncul panah dan tekan panah tersebut.



4. Ubah nama kolomnya menjadi nama yang kita inginkan, disarankan nama kolom FK sama dengan nama kolom PK nya. Lalu sekarang kita akan membuat garis penghubungnya. Pilih pada sidebar kiri dan tambahkan garis one to many di menu *Entity Relation*. Hubungkan Many pada mahasiswa dan One pada jurusan. Karena kita tahu seorang mahasiswa hanya dapat memiliki satu jurusan sedangkan satu jurusan dapat memiliki banyak mahasiswa.

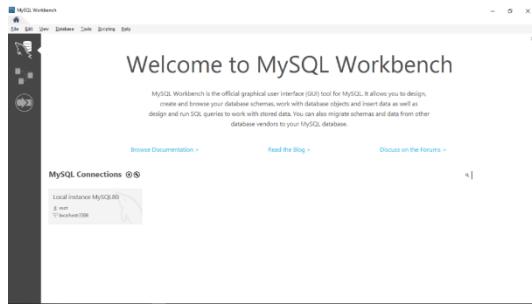


Akhirnya kita telah membuat sebuah ER-Diagram sederhana dengan memanfaatkan platform online app.diagram. Setelah selesai kita tinggal melakukan *Save* untuk menyimpan file xml kita pada tempat yang sudah kita tentukan di awal.

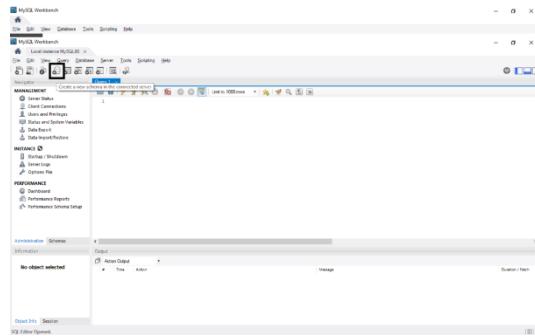
5. Kerjakan Latihan 1 pada modul ini.

1.4.2. Membuat database

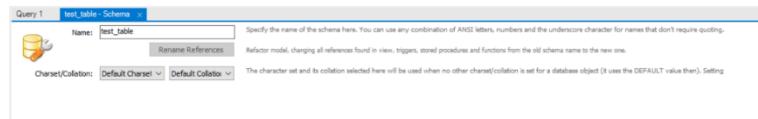
1. Lakukan instalasi MySQL Workbench. Software bisa di download di <https://www.mysql.com/products/workbench/>, bisa juga gunakan softwaredbeaver.
2. Untuk membuat koneksi dengan mesin basis data, kita dapat memilih mesin yang telah kita buat saat instalasi atau membuat baru dengan menekan tombol “plus”.



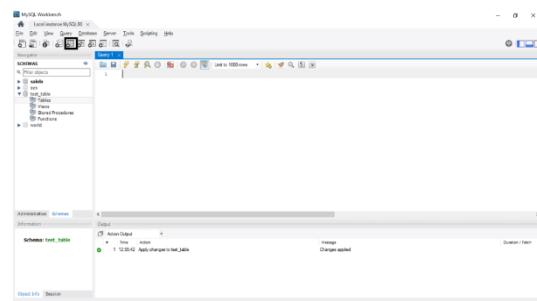
3. Untuk membuat database baru kita menggunakan menu toolbar “*create a new schema in the connected server*”



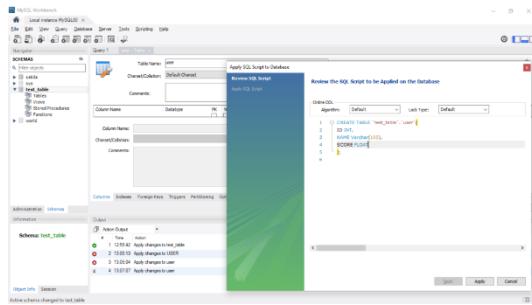
4. Akses schema pada sidebar schema di sebelah kiri.



5. Untuk membuat sebuah tabel pada schema tersebut kita dapat menggunakan menu “*create a new table in the active schema in connected server*”.



6. Tuliskan nama tabel kalian, pastikan schema yang dimaksud benar. Lalu klik apply. Kolom bisa kalian tambahkan pada jendela “*Apply SQL Script to Database*”.



7. Buat database sesuai dengan desain ERD rental film yang sudah kita buat sebelumnya.
Untuk membuat basisdata kita dapat menggunakan query

```
CREATE DATABASE [NAMA BASISDATA];
```

8. Kita akan mencoba untuk membuat basisdata sebuah rental film dengan nama RENTALFILM.

```
CREATE DATABASE RENTALFILM;
```

9. Untuk melihat basisdata yang sudah kita buat maka kita bisa menggunakan query.

```
SHOW DATABASES;
```

10. Untuk menggunakan basisdata yang sudah ada kita bisa menggunakan query,

```
USE rentalfilm;
```

11. Setelah berhasil membuat basisdata, sekarang saatnya kita membuat tabel.

```
CREATE TABLE customers (
    ID int AUTO_INCREMENT PRIMARY KEY,
    NAMAID varchar(255) NOT NULL,
    NAMALENGKAP varchar(255) NOT NULL,
    EMAIL varchar(255) NOT NULL,
    UNIQUE (NAMAID, EMAIL)
);
```

12. Untuk menampilkan tabel kita, kita bisa menggunakan query

```
SHOW TABLES;
```

```
mysql> SHOW TABLES;
+-----+
| Tables_in_rentalfilm |
+-----+
| customers           |
+-----+
1 row in set (0.02 sec)

mysql>
```

13. Untuk Mengetahui detail dari tabel kita, kita bisa menggunakan query

```
DESCRIBE customers;
```

```
mysql> DESCRIBE customers;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| ID    | int    | NO   | PRI | NULL    | auto_increment |
| NAMAID | varchar(255) | NO   | MUL | NULL    |                |
| NAMALENGKAP | varchar(255) | NO   |     | NULL    |                |
| EMAIL | varchar(255) | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql>
```

14. Untuk selanjutnya, kerjakan Latihan no.2 & no.3

1.5. Tugas dan Latihan

1. Buatlah ERD Diagram sesuai dengan contoh ERD yang ada pada Dasar Teori modul ini.
2. Lengkapi table rentalfilm sesuai dengan Contoh ERD.
3. Lakukan latihan menghapus dan merubah atribut tabel

MODUL 2

DATA MANIPULATION LANGUAGE (DML)

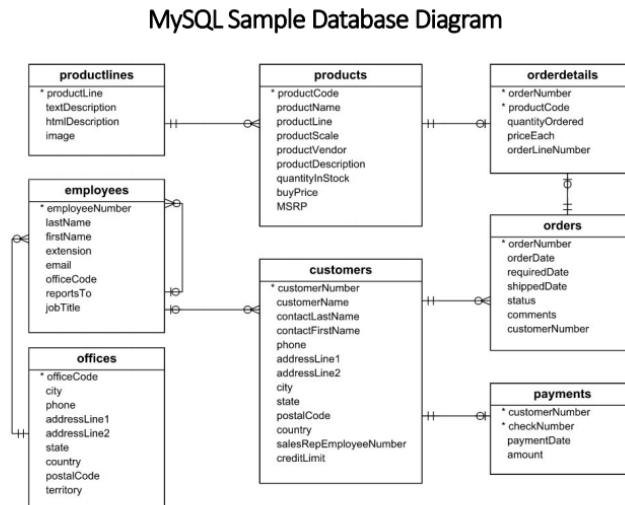
2.1. Tujuan Praktikum

1. Mengenal Query DML
2. Mampu menggunakan query DML

2.2. Dasar Teori

2.2.1. Manipulasi Data MySQL

Pada Manipulasi data kali ini kita akan belajar 4 teknik utama, yaitu memasukkan data, mengubah data, menghapus data, dan melihat data yang kita miliki. Berikut ERD database yang akan dimanipulasi.



Gambar 11. Sample database diagram

Select – Where

Digunakan ketika ingin mengambil data dari sebuah tabel berdasarkan kriteria yang ditentukan. Misalnya digunakan ketika ingin mengambil data dari tabel “customers” yang memiliki tempat tinggal di kota Bogor.

Syntax yang digunakan ketika ingin menggunakan query select where adalah :

```
SELECT * FROM table_name WHERE field_name = value
Contoh : SELECT * FROM customers WHERE city = "Bogor"
```

Select – Join

Seperti yang sudah kita ketahui sebelumnya bahwa tabel-tabel di dalam sql seringkali memiliki relasi satu dan lainnya. Ini membuat hanya mengetahui nilai satu tabel saja tidaklah lengkap untuk mengerti sistem secara keseluruhan.

Kita ambil contoh pada schema *classicmodels*. Dengan membaca tabel user customers dan orders kita tidak dapat menarik kesimpulan secara cepat dari data tersebut. Untuk hal seperti inilah dimana kita menggunakan PK dan FK untuk melakukan Join.

Seperti pada materi tentang struktur data set terdahulu, Join dilakukan untuk menyatukan dua atau lebih tabel dengan korespondensi tertentu, pada kasus sql adalah FK dan PK.

Terdapat 4 macam Join yang dikenal pada mysql,

- Inner Join,
- Left Join,
- Right Join, dan
- Cross Join
- Diagram

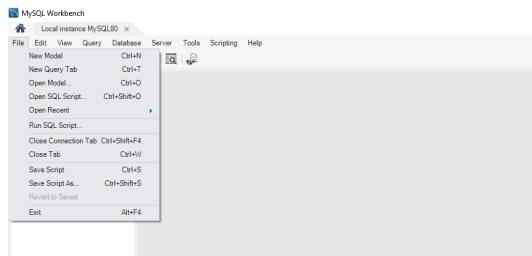
2.3. *Software*

1. Diagram
2. XAMPP
3. MySQL Workbench

2.4. Tahapan Kerja

2.4.1 Persiapkan database

1. Download database dari link berikut <https://www.mysqltutorial.org/wp-content/uploads/2018/03/mysqlsampledatabase.zip>
2. ekstrak sample-database yang telah kita unduh, silahkan pilih tempat ekstrak yang paling mudah untuk dicari. Setelah itu kita buka menu file pada mysql workbench dan pilih *open sql script*



Jika ingin menggunakan Phpmmyadmin dari XAMPP maka buka Command Prompt kemudian arahkan ke folder mysql/bin di tempat installasi XAMPP.

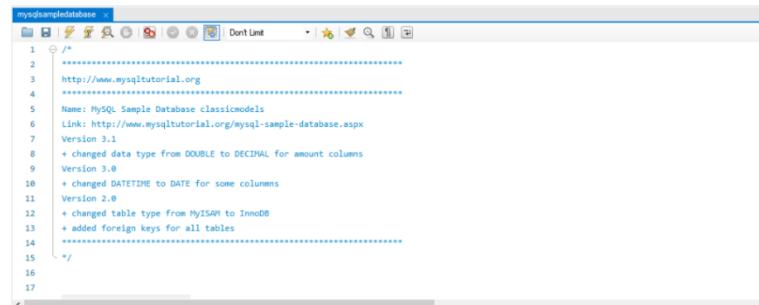
```
D:\>cd xampp\mysql\bin
D:\xampp\mysql\bin>
```

Kemudian ketik “mysql -u root” lalu masukan perintah “source <lokasi file sql script yang sudah didownload>”.

```
D:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 16
Server version: 10.4.21-MariaDB mariadb.org binary distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]> source D:\mysqlsampledatabase.sql
```

Jika sudah selesai maka bisa langsung ke tahap nomor 5.

3. Pilih file sql yang telah berhasil kita ekstrak. Nanti akan muncul jendela seperti pada gambar di bawah.



4. Jalankan seperti pada query-query sebelumnya, maka kalian akan mendapatkan basis data baru pada sidebar, yaitu classicmodels.
5. Struktur query yang digunakan untuk melihat data yang kita miliki adalah,

```
SELECT select_list
FROM table_name
;
```

6. Misalkan kita ingin melihat data pada tabel customers pada basis data classicmodels. kita siapkan dulu jendela query kita seperti saat mengerjakan query sebelumnya. Lalu kita ketik query kita.

```
SELECT *
FROM customers
;
```

7. hasilnya,

| customerNumber | customerName | contactLastName | contactFirstName | phone | addressLine1 | addressLine2 | city | state | postalCode | country |
|----------------|------------------------------|-----------------|------------------|-------------------|------------------------------|--------------|---------------|----------|------------|-----------|
| 103 | Atelier graphique | Schmitt | Carine | 40.32.2555 | 54, rue Royale | | Nantes | NULL | 44000 | France |
| 112 | Signal Gift Stores | King | Jean | 7025551838 | 8489 Strong St. | | Las Vegas | NV | 83030 | USA |
| 114 | Australian Collectors, Co. | Ferguson | Peter | 03 9520 4555 | 636 St Kilda Road | Level 3 | Melbourne | Victoria | 3004 | Australia |
| 119 | La Rochelle Gifts | Labrune | Janine | 40.67.8555 | 67, rue des Cinquante Otages | | Nantes | NULL | 44000 | France |
| 121 | Baane Mini Imports | Bergulfsen | Jonas | 07-98 9555 | Erling Skakkes gate 78 | | Stavarn | NULL | 4110 | Norway |
| 124 | Mini Gifts Distributors Ltd. | Nelson | Susan | 4155551460 | 5677 Strong St. | | San Rafael | CA | 97562 | USA |
| 125 | Havel & Zbyszek Co | Zbyszek | Roland | (20) 642-7555 | ul. Filrowa 68 | | Warszawa | NULL | 01-012 | Poland |
| 128 | Blauer See Auto, Co. | Ketel | Julie | +49 69 66 90 2555 | Lyonerstr. 34 | | Frankfurt | NULL | 60528 | Germany |
| 129 | Mini Wheels Co. | | | 6505555787 | 5557 North Pendale Street | | San Francisco | CA | 94217 | USA |

8. Tanda bintang setelah select melambangkan kita memilih semua kolom pada tabel tersebut. Jika kita ingin memilih hanya dua kolom, misalkan customerNumber dan customerName maka query kita menjadi

```
SELECT customerNumber, customerName
FROM customers
;
```

9. Hasilnya menjadi,

| customerNumber | customerName |
|----------------|------------------------------|
| 103 | Atelier graphique |
| 112 | Signal Gift Stores |
| 114 | Australian Collectors, Co. |
| 119 | La Rochelle Gifts |
| 121 | Baane Mini Imports |
| 124 | Mini Gifts Distributors Ltd. |
| 125 | Havel & Zbyszek Co |
| 128 | Blauer See Auto, Co. |
| 129 | Mini Wheels Co. |
| 131 | Land of Toys Inc. |
| 141 | Euro+ Shopping Channel |

10. Ternyata nama yang ditampilkan sebagai hasil tidak enak dipandang mata, jika kita ingin mengubah nama dari kolom kita bisa menggunakan alias dengan *AS*.

```
SELECT customerNumber as id, customerName as name
FROM customers
;
```

| id | name |
|-----|------------------------------|
| 103 | Atelier graphique |
| 112 | Signal Gift Stores |
| 114 | Australian Collectors, Co. |
| 119 | La Rochelle Gifts |
| 121 | Baane Mini Imports |
| 124 | Mini Gifts Distributors Ltd. |
| 125 | Havel & Zbyszek Co |
| 128 | Blauer See Auto, Co. |
| 129 | Mini Wheels Co. |
| 131 | Land of Toys Inc. |
| 141 | Euro+ Shopping Channel |

11. Jika kita hanya ingin mencari data tertentu dan tidak ingin menampilkan semua data maka kita bisa menggunakan klausa *WHERE*. struktur query menjadi,

```

SELECT
    select_list
FROM
    table_name
WHERE
    search_condition
;

```

12. Misalkan pada query di atas kita ingin mencari untuk customerNumber lebih besar dari 125 dan lebih kecil dari 150. maka query kita menjadi,

```

SELECT customerNumber as id, customerName as name
FROM customers
WHERE
    125 < customerNumber and customerNumber < 150
;

```

| Result Grid | | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|-----|--------------------------|---------|--------------------|
| | id | name | | |
| ▶ | 128 | Blauer See Auto, Co. | | |
| | 129 | Mini Wheels Co. | | |
| | 131 | Land of Toys Inc. | | |
| | 141 | Euro+ Shopping Channel | | |
| | 144 | Volvo Model Replicas, Co | | |
| | 145 | Danish Wholesale Imports | | |
| | 146 | Savely & Henriot, Co. | | |
| | 148 | Dragon Souveniers, Ltd. | | |

13. Pada klausa where di atas kita dapat memasukkan operator. beberapa contoh operator yang bisa kita gunakan adalah

- OR - jika kita memiliki dua statement atau lebih dan ingin mencari dengan salah satu statement benar.
- AND - jika kita memiliki dua statement atau lebih dan ingin mencari jika dan hanya jika semua statement bernilai benar.
- BETWEEN - Jika kita ingin mencari nilai diantara suatu range.

Query di atas sebenarnya bisa disederhanakan dengan menggunakan operator between.

```
WHERE customerNumber BETWEEN 125 AND 150
```

14. Pada kesempatan kali ini kita akan mencoba untuk menggunakan inner join untuk mendapatkan insight dari basisdata yang kita miliki.
15. Query Join secara umum adalah,

```

SELECT column_list
FROM table_1
INNER JOIN table_2 ON join_condition;

```

16. Sekarang kita akan mencoba untuk melakukan studi kasus, kita akan melakukan *data mining* terhadap tabel *order*. pertama lihat terlebih dahulu data pada tabel order.

```
SELECT *
FROM orders
LIMIT 10
```

| | orderNumber | orderDate | requiredDate | shippedDate | status | comments | customerNumber |
|---|-------------|------------|--------------|-------------|---------|--|----------------|
| ▶ | 10100 | 2003-01-06 | 2003-01-13 | 2003-01-10 | Shipped | NULL | 363 |
| | 10101 | 2003-01-09 | 2003-01-18 | 2003-01-11 | Shipped | Check on availability. | 128 |
| | 10102 | 2003-01-10 | 2003-01-18 | 2003-01-14 | Shipped | NULL | 181 |
| | 10103 | 2003-01-29 | 2003-02-07 | 2003-02-02 | Shipped | NULL | 121 |
| | 10104 | 2003-01-31 | 2003-02-09 | 2003-02-01 | Shipped | NULL | 141 |
| | 10105 | 2003-02-11 | 2003-02-21 | 2003-02-12 | Shipped | NULL | 145 |
| | 10106 | 2003-02-17 | 2003-02-24 | 2003-02-21 | Shipped | NULL | 278 |
| | 10107 | 2003-02-24 | 2003-03-03 | 2003-02-26 | Shipped | Difficult to negotiate with customer. We need m... | 131 |
| | 10108 | 2003-03-03 | 2003-03-12 | 2003-03-08 | Shipped | NULL | 385 |
| | 10109 | 2003-03-10 | 2003-03-19 | 2003-03-11 | Shipped | Customer requested that FedEx Ground is used... | 486 |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Limit digunakan untuk membatasi hasil yang diambil.

Seperti yang terlihat, data pada orders kurang bisa menjelaskan apapun. Akan tetapi, ternyata tabel ini tersambung dengan table customers pada kolom CustomerNumber. Maka dari itu kita coba gunakan join pada tabel tersebut.

```
SELECT *
FROM orders a
INNER JOIN customers b ON a.customerNumber = b.customerNumber
LIMIT 10
;
```

| | orderNumber | orderDate | requiredDate | shippedDate | status | comments | customerNumber | customerNumber | customerName | contact.lastName |
|---|-------------|------------|--------------|-------------|---------|--|----------------|----------------|----------------------------|------------------|
| ▶ | 10123 | 2003-05-20 | 2003-05-29 | 2003-05-22 | Shipped | NULL | 103 | 103 | Atelier graphique | Schmitt |
| | 10298 | 2004-09-27 | 2004-10-05 | 2004-10-01 | Shipped | NULL | 103 | 103 | Atelier graphique | Schmitt |
| | 10345 | 2004-10-01 | 2004-10-16 | 2004-10-16 | Shipped | NULL | 103 | 103 | Atelier graphique | Schmitt |
| | 10324 | 2003-09-21 | 2003-09-29 | 2003-09-29 | Shipped | Customer very concerned about the exact color... | 112 | 112 | Signal Gf Stores | King |
| | 10278 | 2004-08-06 | 2004-08-16 | 2004-08-09 | Shipped | NULL | 112 | 112 | Signal Gf Stores | King |
| | 10346 | 2004-11-29 | 2004-12-05 | 2004-11-30 | Shipped | NULL | 112 | 112 | Signal Gf Stores | King |
| | 10120 | 2003-04-29 | 2003-05-08 | 2003-05-01 | Shipped | NULL | 114 | 114 | Australian Collectors, Co. | Ferguson |
| | 10125 | 2003-05-21 | 2003-05-27 | 2003-05-24 | Shipped | NULL | 114 | 114 | Australian Collectors, Co. | Ferguson |
| | 10223 | 2004-02-20 | 2004-02-29 | 2004-02-24 | Shipped | NULL | 114 | 114 | Australian Collectors, Co. | Ferguson |
| | 10342 | 2004-11-24 | 2004-12-01 | 2004-11-29 | Shipped | NULL | 114 | 114 | Australian Collectors, Co. | Ferguson |

17. Kita berhasil menggabungkan kedua tabel. a dan b digunakan sebagai alias untuk kedua tabel. Kita akan mencoba mengambil hanya beberapa kolom.
orders

1. orderDate
2. orderNumber

customers

1. customerName
2. country

18. Jika melihat pada ER-Diagram, kita tahu bahwa orders terhubung dengan orderdetails. orderNumber pada orders menjadi FK pada orderdetails. kita akan coba hubungkan dan ambil kolom productCode dan quantityOrdered. Kita akan mencari untuk setiap orang yang tinggal di negara Spanyol dan America. Query yang akan kita tulis dari hasil penjelasan di atas adalah

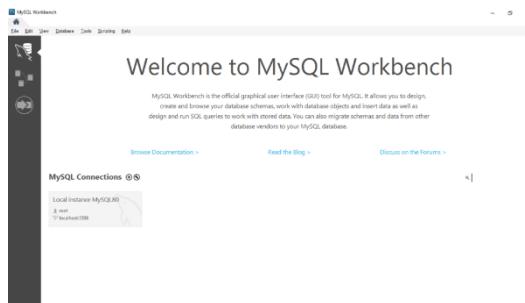
```
select a.orderNumber, a.orderDate, b.customerName, b.country, c.productCode,
c.quantityOrdered
from orders a
inner join customers b on a.customerNumber = b.customerNumber
inner join orderdetails c on a.orderNumber = c.orderNumber
where b.country in ("Spain", "USA")
;
```

| | orderNumber | orderDate | customerName | country | productCode | quantityOrdered |
|---|-------------|------------|--------------------|---------|-------------|-----------------|
| ▶ | 10124 | 2003-05-21 | Signal Gift Stores | USA | S18_1749 | 21 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S18_2248 | 42 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S18_2325 | 42 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S18_4409 | 36 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S18_4933 | 23 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_1046 | 22 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_1937 | 45 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_2022 | 22 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_2766 | 32 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_2887 | 25 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_3191 | 49 |
| | 10124 | 2003-05-21 | Signal Gift Stores | USA | S24_3422 | 42 |

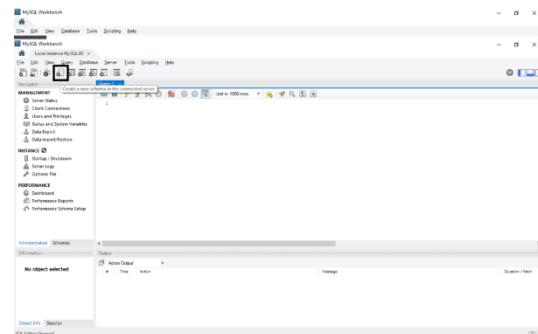
Akhirnya kita bisa memisahkan pembeli yang berasal dari dua negara dan melihat jumlah pembelian beserta nama mereka.

2.4.2. Membuat database

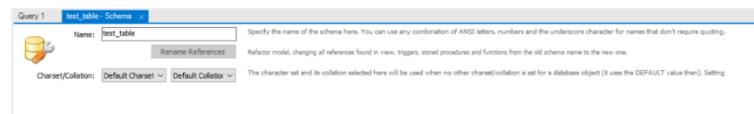
1. Lakukan instalasi MySQL Workbench. Software bisa di download di <https://www.mysql.com/products/workbench/>, bisa juga gunakan softwaredbeaver.
2. Untuk membuat koneksi dengan mesin basis data, kita dapat memilih mesin yang telah kita buat saat instalasi atau membuat baru dengan menekan tombol “plus”.



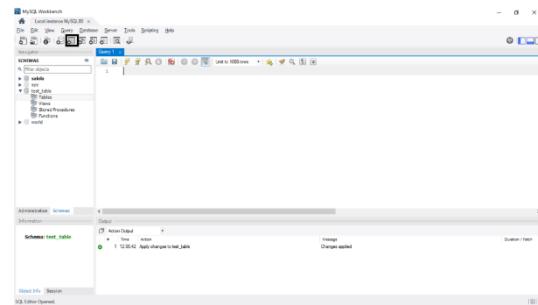
3. Untuk membuat database baru kita menggunakan menu toolbar “create a new schema in the connected server”



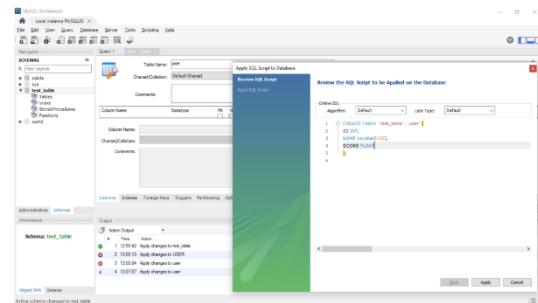
4. Akses schema pada sidebar schema di sebelah kiri.



5. Untuk membuat sebuah tabel pada schema tersebut kita dapat menggunakan menu “create a new table in the active schema in connected server”.



6. Tuliskan nama tabel kalian, pastikan schema yang dimaksud benar. Lalu klik apply. Kolom bisa kalian tambahkan pada jendela “Apply SQL Script to Database”.



7. Buat database sesuai dengan desain ERD rental film yang sudah kita buat sebelumnya. Untuk membuat basisdata kita dapat menggunakan query

```
CREATE DATABASE [NAMA BASISDATA];
```

8. Kita akan mencoba untuk membuat basisdata sebuah rental film dengan nama RENTALFILM.

```
CREATE DATABASE RENTALFILM;
```

9. Untuk melihat basisdata yang sudah kita buat maka kita bisa menggunakan query.

```
SHOW DATABASES;
```

```

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| rentalfilm |
| sakila |
| sys |
| test_table |
| world |
+-----+
8 rows in set (0.05 sec)

mysql> -

```

10. Untuk menggunakan basisdata yang sudah ada kita bisa menggunakan query,

```
USE rentalfilm;
```

11. Setelah berhasil membuat basisdata, sekarang saatnya kita membuat tabel.

```

CREATE TABLE customers (
    ID int AUTO_INCREMENT PRIMARY KEY,
    NAMAID varchar(255) NOT NULL,
    NAMALENGKAP varchar(255) NOT NULL,
    EMAIL varchar(255) NOT NULL,
    UNIQUE (NAMAID, EMAIL)
);

```

12. Untuk menampilkan tabel kita, kita bisa menggunakan query

```
SHOW TABLES;
```

```

mysql> SHOW TABLES;
+-----+
| Tables_in_rentalfilm |
+-----+
| customers |
+-----+
1 row in set (0.02 sec)

mysql>

```

13. Untuk Mengetahui detail dari tabel kita, kita bisa menggunakan query

```
DESCRIBE customers;
```

```

mysql> DESCRIBE customers;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| ID   | int  | NO   | PRI | NULL   | auto_increment |
| NAMAID | varchar(255) | NO   | MUL | NULL   | |
| NAMALENGKAP | varchar(255) | NO   |     | NULL   | |
| EMAIL | varchar(255) | NO   |     | NULL   | |
+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql>

```

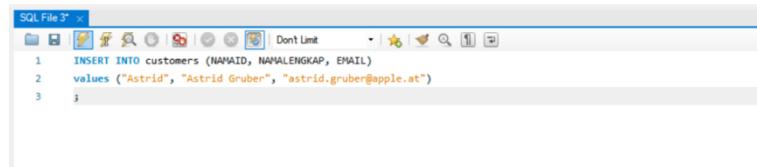
14. Insert data ke dalam basisdata rentalfilm dan tabel customers. Data yang akan kita masukkan adalah

```

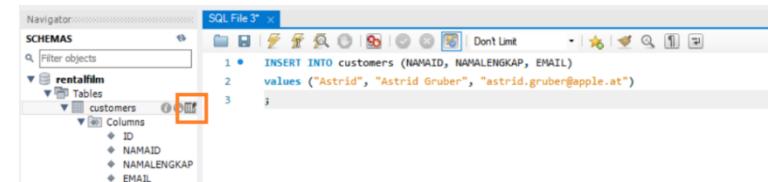
id = Astrid,
fullname = Astrid Gruber,
email = astrid.gruber@apple.at
Maka query kita akan menjadi,
INSERT INTO customers (NAMAID, NAMALENGKAP, EMAIL)
values ("Astrid", "Astrid Gruber", "astrid.gruber@apple.at")
;

```

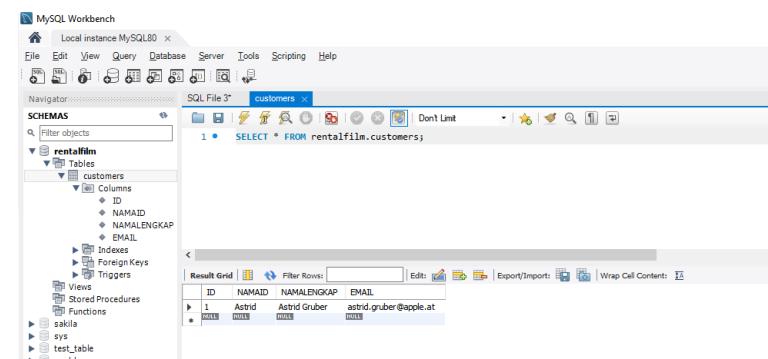
15. Untuk menjalankan kode kita kita gunakan menu ketiga pada sql toolbar, ditandai dengan kotak biru.



16. Untuk Melihat hasil kerja kita kita hanya perlu memilih tabel yang ingin kita lihat dan tekan icon pada kotak jingga.



17. Seperti yang kita lihat pada gambar di bawah ini, data yang kita ingin kan telah masuk kedalam tabel customers.



18. Tambahkan banyak data baru

```

INSERT INTO customers (NAMAID, NAMALENGKAP, EMAIL)
values
("Fernanda", "Fernanda Ramos", "fernadaramos4@uol.com.br"),
("Mark", "Mark Philips", "mphilips12@shaw.ca"),
("Jennifer", "Jennifer Peterson", "jenniferp@rogers.ca")
;

```

| ID | NAMAID | NAMALENGKAP | EMAIL |
|----|----------|-------------------|--------------------------|
| 1 | Astrid | Astrid Gruber | astrid.gruber@apple.at |
| 2 | Fernanda | Fernanda Ramos | fernadaramos4@uol.com.br |
| 3 | Mark | Mark Philips | mphilips12@shaw.ca |
| 4 | Jennifer | Jennifer Peterson | jenniferp@rogers.ca |
| * | NULL | NULL | NULL |

19. Mengubah data. Misalkan kita akan menggunakan data pada tabel customers dan ingin mengganti Nama Jennifer Peterson menjadi Jennifer Lauren dan emailnya menjadi jenniferl@rogers.ca. Untuk mengubah data yang telah ada kita dapat menggunakan query,

```
UPDATE customers
SET
    NAMALENGKAP = "Jennifer Lauren",
    EMAIL = "jenniferl@rogers.ca"
WHERE
    ID = 4
;
```

| ID | NAMAID | NAMALENGKAP | EMAIL |
|----|----------|-----------------|--------------------------|
| 1 | Astrid | Astrid Gruber | astrid.gruber@apple.at |
| 2 | Fernanda | Fernanda Ramos | fernadaramos4@uol.com.br |
| 3 | Mark | Mark Philips | mphilips12@shaw.ca |
| 4 | Jennifer | Jennifer Lauren | jenniferl@rogers.ca |
| * | NULL | NULL | NULL |

20. Menghapus data. Misalkan kita ingin menghapus Data Jennifer. Kita ketahui pada id jennifer adalah 4, maka query kita menjadi

```
DELETE FROM table_name
WHERE condition
;
```

| ID | NAMAID | NAMALENGKAP | EMAIL |
|----|----------|----------------|--------------------------|
| 1 | Astrid | Astrid Gruber | astrid.gruber@apple.at |
| 2 | Fernanda | Fernanda Ramos | fernadaramos4@uol.com.br |
| 3 | Mark | Mark Philips | mphilips12@shaw.ca |
| * | NULL | NULL | NULL |

2.5. Tugas dan Latihan

- Buatlah ERD Diagram sesuai dengan contoh ERD yang ada pada Dasar Teori modul ini.
- Buatlah query untuk menampilkan daftar nama customer yang telah melakukan order dimana barang yang pesan lebih dari 50.

| customerName | productName | quantityOrdered |
|----------------------------|---------------------------------------|-----------------|
| Australian Collectors, Co. | 1948 Porsche 356-A Roadster | 55 |
| Australian Collectors, Co. | 1976 Ford Gran Torino | 55 |
| La Rochelle Gifts | 1970 Dodge Coronet | 55 |
| Euro+ Shopping Channel | 1957 Chevy Pickup | 54 |
| Euro+ Shopping Channel | 1964 Mercedes Tour Bus | 56 |
| Euro+ Shopping Channel | 1992 Ferrari 360 Spider red | 60 |
| Euro+ Shopping Channel | 1962 Volkswagen Microbus | 70 |
| Euro+ Shopping Channel | 1969 Harley Davidson Ultimate Chopper | 66 |
| Euro+ Shopping Channel | 2003 Harley-Davidson Eagle Drag Bike | 56 |
| Euro+ Shopping Channel | 1940 Ford Pickup Truck | 54 |
| Danish Wholesale Imports | 1993 Mazda RX-7 | 61 |
| Danish Wholesale Imports | 1948 Porsche Type 356 Roadster | 65 |
| Diecast Classics Inc. | 1937 Lincoln Berlin | 51 |
| Handji Gifts& Co | 1939 Chevrolet Deluxe Coupe | 61 |
| Gift Depot Inc. | 1962 City of Detroit Streetcar | 51 |
| UK Collectables, Ltd. | 2003 Harley-Davidson Eagle Drag Bike | 66 |
| UK Collectables, Ltd. | 2002 Suzuki XREO | 66 |
| Mini Caravy | 1969 Dodge Charger | 97 |
| Mini Caravy | 1948 Porsche 356-A Roadster | 61 |
| Mini Caravy | 1992 Ferrari 360 Spider red | 55 |
| Mini Caravy | 1956 Porsche 356A Coupe | 76 |
| Enaco Distributors | 1960 BSA Gold Star DBD34 | 55 |
| Enaco Distributors | 1997 BMW F650 ST | 55 |
| Enaco Distributors | America West Airlines B757-200 | 55 |
| Souveniers And Things Co. | 1932 Alfa Romeo 8C2300 Spider Sport | 66 |

3. Buatlah query untuk menampilkan daftar nama customer, nama depan pegawai, nama belakang pegawai dan negara kantor pegawai tersebut.

| customerName | nama depan pegawai | nama belakang pegawai | country |
|------------------------------|--------------------|-----------------------|---------|
| Mini Gifts Distributors Ltd. | Leslie | Jennings | USA |
| Mini Wheels Co. | Leslie | Jennings | USA |
| Technics Stores Inc. | Leslie | Jennings | USA |
| Corporate Gift Ideas Co. | Leslie | Jennings | USA |
| The Sharp Gifts Warehouse | Leslie | Jennings | USA |
| Signal Collectibles Ltd. | Leslie | Jennings | USA |
| Signal Gift Stores | Leslie | Thompson | USA |
| Toys4GrownUps.com | Leslie | Thompson | USA |
| Boards & Toys Co. | Leslie | Thompson | USA |
| Collectable Mini Designs Co. | Leslie | Thompson | USA |
| Men 'R' US Retailers, Ltd. | Leslie | Thompson | USA |
| West Coast Collectables Co | Leslie | Thompson | USA |
| Cambridge Collectables Co | Julie | Firrelli | USA |
| Online Mini Collectables | Julie | Firrelli | USA |
| Mini Creations Ltd. | Julie | Firrelli | USA |
| Classic Gift Ideas, Inc | Julie | Firrelli | USA |
| Collectables For Less Inc. | Julie | Firrelli | USA |
| Diecast Collectables | Julie | Firrelli | USA |
| Diecast Classics Inc. | Steve | Patterson | USA |
| Auto-Moto Classics Inc. | Steve | Patterson | USA |
| Marta's Replicas Co. | Steve | Patterson | USA |
| Gifts4AllAges.com | Steve | Patterson | USA |
| Online Diecast Creations Co. | Steve | Patterson | USA |
| FunGiftIdeas.com | Steve | Patterson | USA |
| Muscle Machine Inc | Foon Yue | Tseng | USA |

MODUL 3

NOSQL & MONGODB

3.1. Tujuan Praktikum

1. Mengetahui perbedaan SQL dan NoSQL
2. Mampu menginstall MongoDB
3. Memahami struktur NoSQL

3.2. Dasar Teori

3.2.1. Mengenal NoSQL

NoSQL memiliki arti Not Only SQL. RDBMS (SQL) pada umumnya menggunakan query sql untuk menyimpan dan melakukan pengambilan data. Akan tetapi, NoSQL dapat menyimpan data dalam berbagai bentuk, baik terstruktur, semi terstruktur, maupun yang tidak terstruktur. Basis data NoSQL termasuk pada Non-Relational Data Management System, tidak adanya relasi langsung antara datanya, tidak memerlukan schema yang tetap, dan lebih mudah untuk melakukan *scaling* pada sistem.

Beberapa perbedaan SQL dan NoSQL diantaranya:

- SQL adalah basis data relasional sedangkan NoSQL tidak relational.
- SQL menggunakan query yang terstruktur dan memiliki schema yang disiapkan terlebih dahulu sedangkan NoSQL tidak memiliki schema tertentu sehingga lebih dinamis dalam bentuk data.
- Pengembangan SQL biasanya secara vertikal, sedangkan NoSQL lebih ke arah horizontal.
- SQL berbasis pada tabel, sedangkan NoSQL bisa memiliki banyak bentuk misalnya pasangan key-value, dokumen, ataupun graph.
- SQL lebih baik untuk bertransaksi dengan banyak baris, sedangkan NoSQL lebih baik untuk transaksi tak berstruktur seperti JSON.

Salah satu alasan kenapa NoSQL menarik ada pada poin 3, yaitu *scalability* atau kemampuan untuk dikembangkan.

Bila dilakukan pemisalan sebagai sebuah bangunan hotel, pengembangan vertikal artinya pengembangan dengan melakukan penambahan tinggi dan besar dari bangunan hotel tersebut. Lebih mudah dari sisi pengurusan dan manajemen akan tetapi membutuhkan biaya dan teknik yang tinggi semakin besar dan tinggi pembangunannya.



Gambar 12. Vertical dan Horizontal Scaling

Pengembangan Horizontal merupakan pengembangan dengan menambah jumlah Hotel. Lebih mudah dari sisi pengembangan akan tetapi terkait sisi manajemen hotel akan lebih sulit semakin banyak hotelnya.

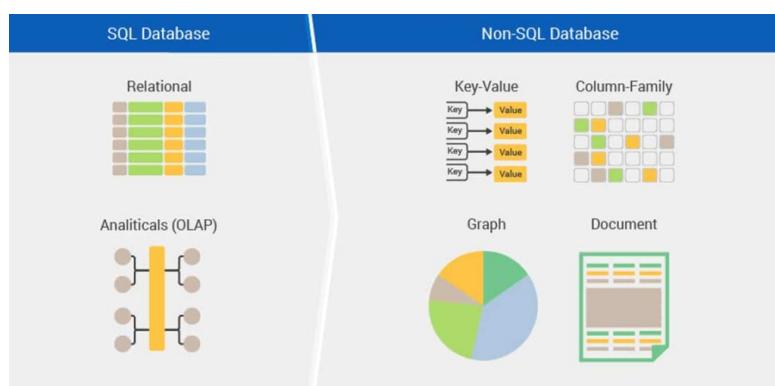
Pengembangan satu server SQL dilakukan dengan menambah kemampuan dasar servernya (dalam hal hotel adalah pondasinya), seperti RAM, CPU, dan SSD. Dan seperti yang kita ketahui, adakan ada batasnya terkait hal tersebut.

Pengembangan NoSQL sesederhana dengan menambah jumlah dari servernya, Karena data yang tidak terstruktur dari NoSQL lebih mudah dipecah daripada data berschema dan berelasi seperti SQL.

3.2.2. Tipe-tipe NoSQL

Secara garis besar, basisdata NoSQL dibagi kedalam 4 kategori utama.

- Pasangan Key - Value
- Berbasis Dokumen
- Berbasis Graph
- Berbasis Tabular



Gambar 13. Tipe SQL dan NoSQL database

3.2.3. MongoDB

MongoDB adalah basisdata NoSQL yang berbasis pada dokumen. MongoDB menyimpan datanya dalam bentuk dokumen yang mirip dengan JSON dan tentu saja tidak perlu schema. Jadi satu dokumen dan dokumen yang lain bisa memiliki struktur yang berbeda.

3.2.3. Terminologi

Struktur data pada mongoDB terdiri collection dan document. Untuk lebih mengerti struktur data ini kita akan membandingkannya dengan MySQL pada gambar di bawah ini.

| RDBMS | MongoDB |
|-------------|--|
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key _id provided by MongoDB itself) |

Gambar 14. Terminologi

Jadi setiap kolom dijelaskan oleh field, lalu dokumen merepresentasikan baris, dan collection merupakan tabel dimana dokumen dikumpulkan.

Misalkan kita coba mengacu pada saat kita belajar mysql maka database kita akan menjadi,

```
database : rentalfilm
collection : customers
field : id, username(namaId), namaLengkap, email
```

collection bisa diibaratkan sebagai tabel dan field bisa diibaratkan sebagai kolom. Untuk membuat collection kita hanya perlu menggunakan query:

```
db.createCollection(<namacollection>, parameter(opsional))
```

- capped - Ketika mencapai ukuran maksimum maka data lama akan dihapus agar data baru bisa masuk
- autoIndexId - membuat index otomatis pada kolom _id
- size - menentukan ukuran maksium dalam byte pada saat parameter capped aktif
- max - menentukan jumlah maksimal dokumen pada saat parameter capped aktif.

Seperti yang dilihat, kita tidak perlu menentukan struktur kolom kita di awal pembuatan seperti mysql, sehingga NoSQL lebih dinamis terkait hal tersebut.

3.2.4. Perintah dasar

Berikut ini adalah perintah-perintah dasar dari MongoDB

Perintah Dasar

| | |
|---------------------------------------|--|
| mongo | masuk ke dalam terminal mongodb |
| show dbs | menampilkan semua basisdata pada server |
| show collections | menampilkan semua collection pada basisdata |
| use <namabasisdata> | memilih basisdata yang ingin digunakan |
| db | menampilkan basisdata yang sedang digunakan |
| db.dropDatabase() | menghapus basisdata yang sedang digunakan |
| db.createCollection(<namacollection>) | membuat collection pada basisdata yang digunakan |

Gambar 15. Perintah dasar MongoDB

3.3. Software

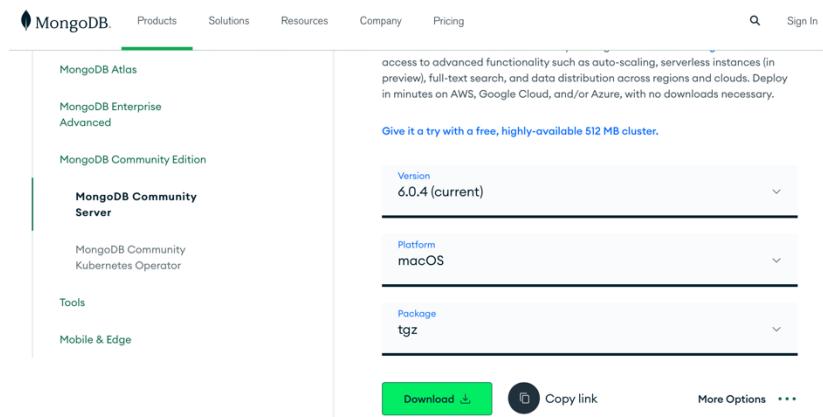
1. MongoDB
2. Compass
3. MongoDB Shell

3.4. Tahapan Kerja

3.4.1. Instalasi MongoDB

1. Download MongoDB dari link berikut, download versi stable terakhir 6.0.1(current)

<https://www.mongodb.com/try/download/community>

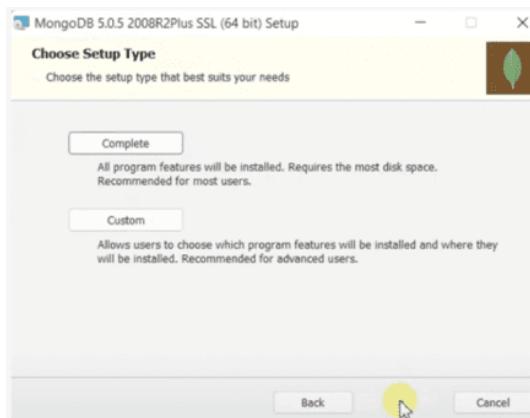


- Setelah selesai download silahkan untuk menjalankan file instalasi.

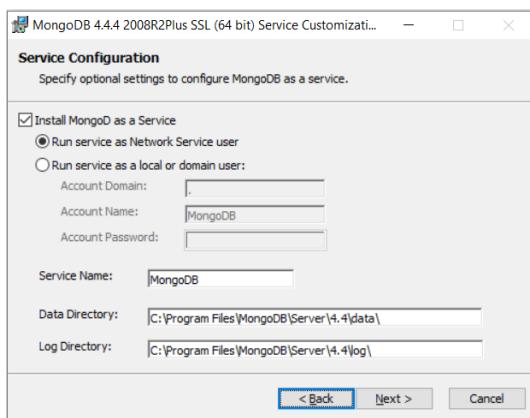


Gambar 16. Instalasi MongoDB

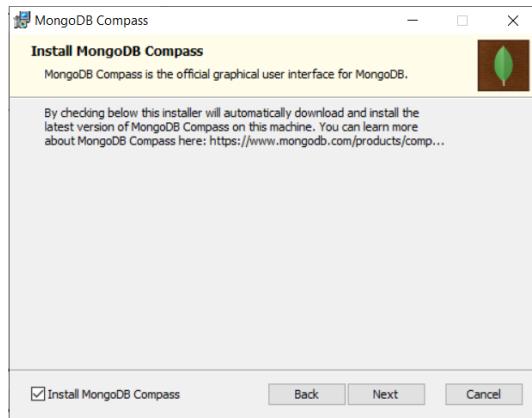
- Lanjutkan instalasi, Pilihlah menu complete untuk saat ini.



- Pada menu konfigurasi silahkan pilih *install mongod as a service* agar kita lebih mudah untuk mengelola server kita. Pilih *run service as Network Service User*.

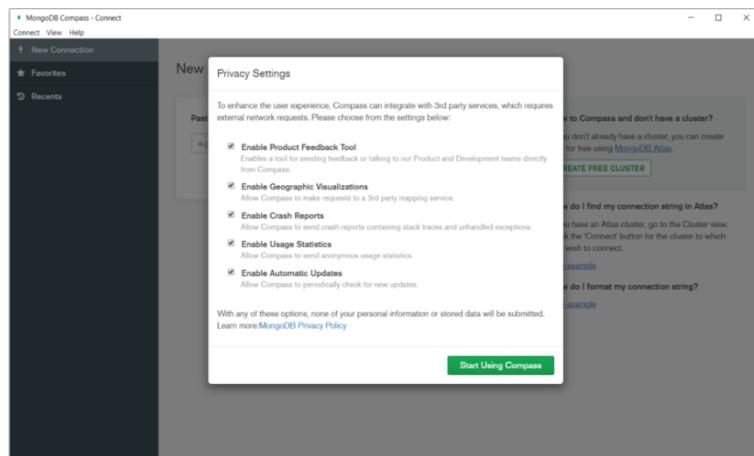


- Pada menu mongoDB Compass, lakukan instalasi. MongoDB Compass merupakan GUI untuk mongoDB, sama seperti MySQL Workbench untuk MySQL.



Gambar 17. Opsi instalasi MongoDB Compass

6. Setelah selesai instalasi mongodb, silahkan tutup jendela instalasi. Untuk mongodb compass akan terlihat tampilan seperti di bawah ini.



Gambar 18. Aplikasi MongoDB

7. Tekan *Start using compass* lalu tutup jendela mongodb compass untuk saat ini.

3.4.2. Set up Environment Variable

1. Download “MongoDB Shell” melalui halaman download di <https://www.mongodb.com/try/download/shell>. Kemudian ekstrak zip file yang sudah didownload ke folder yang diinginkan.

TOOLS

MongoDB Shell Download

MongoDB Shell is the quickest way to connect to (and work with) MongoDB. Easily query data, configure settings, and execute other actions with this modern, extensible command-line interface – replete with syntax highlighting, intelligent autocomplete, contextual help, and error messages.

Note: MongoDB Shell is an open source (Apache 2.0), standalone product developed separately from the MongoDB Server.

[Learn more](#)

Version
1.8.0

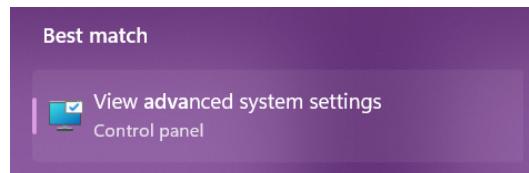
Platform
Windows 64-bit (8.1+)

Package
zip

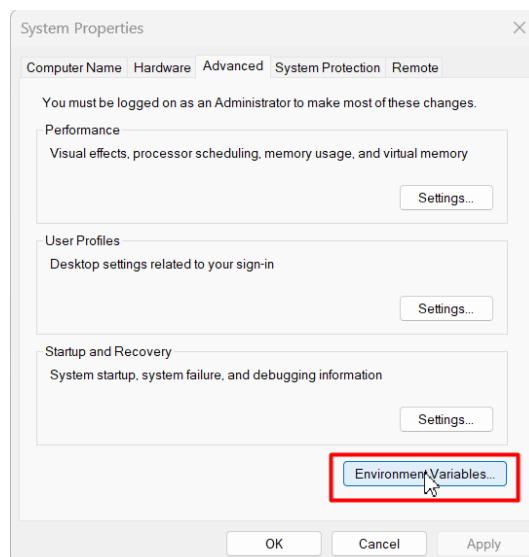
[Download](#) [Copy link](#) More Options 



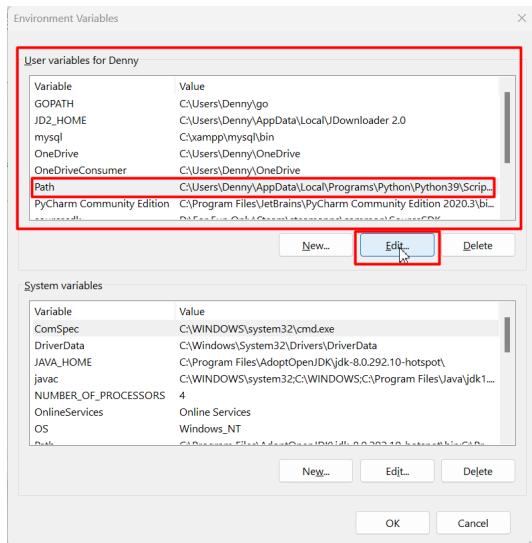
2. Buka search dengan menekan tombol start dan S di keyboard lalu ketik “Advanced System Settings” lalu pilih menu “View Advanced System Setting”.



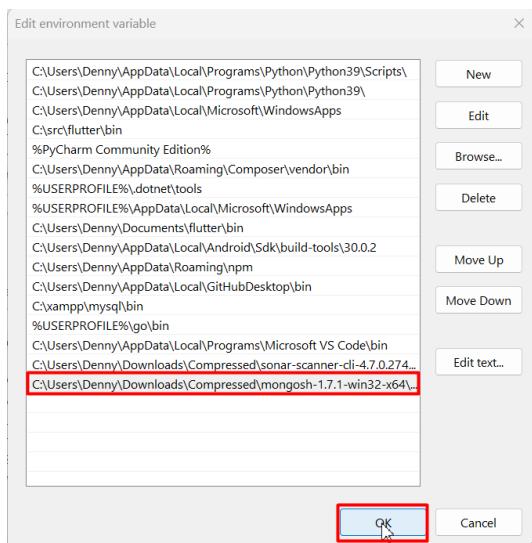
3. Dari jendela baru yang terbuka tekan tombol “Environment Variables”.



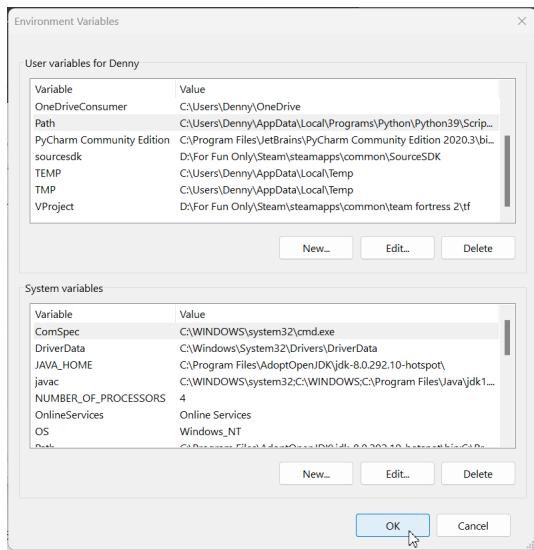
4. Di menu berikutnya pada bagian “User Variables” Cari dan pilih variable “Path” kemudian tekan tombol “Edit”.



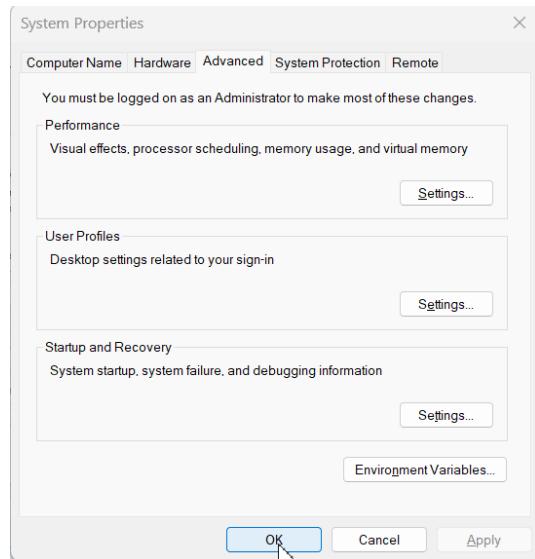
5. Cari dan salin lokasi folder MongoDB Shell yang sudah diekstrak tadi, contohnya “C:\Users\Denny\Downloads\Compressed\mongosh-1.7.1-win32-x64\bin”. Kemudian di menu Edit Environment Variable tadi tambahkan lokasi folder MongoDB bin yang sudah di salin kedalam baris yang kosong dengan mengklik dua kali. Jika sudah tekan tombol OK.



6. Kembali pada menu “Environment Table” Tekan Tombol OK.



7. Lakukan hal yang sama pada “Advanced System Settings”.



3.4.3. Persiapan Basisdata

- Pada tahap persiapan database kita dapat menggunakan cmd untuk bekerja. Untuk menggunakan cmd kita hanya perlu mengetikan pada cmd.

```
mongosh
```

```
C:\Users\Deny>mongosh localhost
Current Mongosh Log ID: c40ed62f99ea9a079faf7db
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
Using MongoDB: 6.0.4
Using Mongosh: 1.7.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

The server generated these startup warnings when booting
2023-03-13T06:47:26.572+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

localhost>
```

2. Untuk menunjuk pada suatu server khusus kita bisa menggunakan alamat server kita setelah mongo. pada contoh ini kita akan menggunakan localhost

```
mongosh Localhost
```

3. Pastikan path menuju mongodb server sudah tersimpan pada windows. Untuk membuat/berpindah basisdata, kita dapat menggunakan query.

```
use <namadatabase>
```

4. Maka mongo akan mengarahkan kita pada basisdata yang kita maksud jika ada, jika belum ada maka akan otomatis untuk dibuat. Kita akan membuat sama seperti pada saat mempelajari mysql.

```
use rentalfilm
```

```
C:\Users\thiop>mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4b3f3756-1d79-4152-9f63-ead0f1ac027") }
MongoDB server version: 4.4.4

The server generated these startup warnings when booting:
2021-04-01T23:43:15.160+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

> use rentalfilm
switched to db rentalfilm
>
```

5. Untuk melihat semua basisdata yang kita miliki pada server mongodb, kita dapat menggunakan query

```
show dbs
```

```

C:\Users\thiop\mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4b3f3756-1d79-4152-9f63-ead0f1ac027") }
MongoDB server version: 4.4.4
...
The server generated these startup warnings when booting:
2021-04-01T23:43:15.160+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
...
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
...
> use rentalfilm
switched to db rentalfilm
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
>

```

Basis data *rentalfilm* belum muncul karena kita belum memasukan data. sehingga belum tersimpan sepenuhnya

6. Setelah berhasil membuat basisdata, sekarang saatnya kita membuat collection. Seperti yang dibahas sebelumnya,
7. Sekarang kita akan membuat collection *customers*

```
db.createCollection("customers")
```

```

C:\Users\thiop\mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4b3f3756-1d79-4152-9f63-ead0f1ac027") }
MongoDB server version: 4.4.4
...
The server generated these startup warnings when booting:
2021-04-01T23:43:15.160+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
...
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
...
> use rentalfilm
switched to db rentalfilm
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
> db.createCollection("customers")
{ "ok" : 1 }
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
rentalfilm 0.000GB
>

```

8. Jika kita cek kembali daftar basis data kita, maka *rentalfilm* sudah terdaftar karena sudah memiliki isi di dalamnya.

```

C:\Users\thiop\mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4b3f3756-1d79-4152-9f63-ead0f1ac027") }
MongoDB server version: 4.4.4
...
The server generated these startup warnings when booting:
2021-04-01T23:43:15.160+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
...
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
...
> use rentalfilm
switched to db rentalfilm
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
> db.createCollection("customers")
{ "ok" : 1 }
> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
rentalfilm 0.000GB
>

```

3.4.4. Manipulasi Data

1. Inser Data. Memasukan data dapat kita lakukan pada mongo shell dengan menggunakan query,

```
db.<namacollection>.insert(<document>)
```

Misalkan kita ingin memasukkan data,

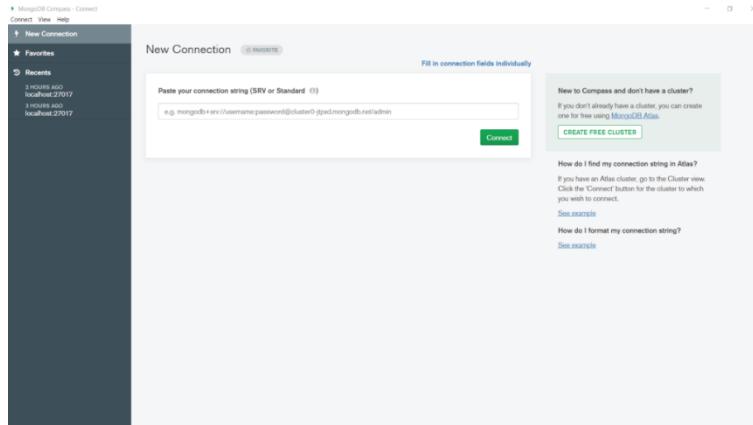
```
id = Astrid,  
fullname = Astrid Tiar,  
email = astrid.tiar@ibik.ac.id
```

maka query kita menjadi,

```
db.customers.insert({  
  id : "Astrid",  
  fullname : "Astrid Tiar ",  
  email : "astrid.tiar@ibik.ac.id"  
})
```

```
> db.customers.insert({id : "Astrid", fullname : "Astrid Gruber", email : "astrid.gruber@apple.at"})  
> WriteResult({ "nInserted" : 1 })
```

2. Untuk melihat datanya kita bisa menggunakan MongoDB Compass. Jadi pertama bukalah MongoDB Compass kita dan pada menu koneksi kita tekan *connect*, kita akan tersambung dengan localhost pada port 27017 (default port mongodb). Jika ingin mengisi secara manual server kita, bisa dengan memilih menu *fill in connection fields manually*.



3. Setelah itu kita pilih Basis data dan collection yang ingin manipulasi datanya.

The screenshot shows the MongoDB Compass interface. At the top, there are two tabs: "Databases" and "Performance". Below them is a table for "Databases" with columns: Database Name, Storage Size, Collections, and Indexes. It lists four databases: admin (20.0KB, 0 collections, 1 index), config (36.0KB, 0 collections, 2 indexes), local (20.0KB, 1 collection, 1 index), and rentalfilm (4.0KB, 1 collection, 1 index). Below this is another table for "Collections" with columns: Collection Name, Documents, Avg. Document Size, Total Document Size, Num. Indexes, and Total Index Size. It lists one collection: customers (0 documents, - avg. size, 0.0 B total size, 1 index, 4.0 KB total index size). The main area shows the "rentalfilm.customers" collection details. It has tabs for "Documents", "Aggregations", "Schema", "Explain Plan", "Indexes", and "Validation". A "FILTER" button is present. Below the tabs, there are buttons for "ADD DATA", "VIEW", and "OPTIONS". A message says "Displaying documents 0 - 0 of 0". A large green "Import Data" button is at the bottom. The overall interface is light gray with green highlights for selected items.

4. Tekan tombol *find* maka datanya akan muncul

This screenshot shows the same MongoDB Compass interface as the previous one, but with a single document listed in the "Documents" tab of the rentalfilm.customers collection. The document details are as follows:

```

_id: ObjectId("6006d27344be35c5chedf4e1ad")
id: "Astrid"
fullname: "Astrid Gruber"
email: "astrid.gruber@apple.at"
  
```

The rest of the interface remains the same with its various tabs and import/export buttons.

5. Kita juga bisa memasukkan data menggunakan tombol *add data* --> *Insert document*. Untuk memasukkan data kita hanya perlu memasukkan data tersebut ke dalam list, mirip seperti saat kita bekerja dengan dictionary.

```

db.customers.insert([
  {
    id : "Fernanda",
    fullname : "Fernanda Ramos",
    email : "fernadaramos4@uol.com.br",
    age : 24
  },
  {
    id : "Mark",
    fullname : "Mark Philips",
    email : "mphilips12@shaw.ca",
    city : "San Francisco"
  },
  {
    id : "Jennifer",
    fullname : "Jennifer Peterson",
    email : "jenniferp@rogers.ca",
    occupation : "teacher"
  }
])

```

6. Berbeda dengan sql yang memiliki schema yang tetap, pada NoSQL kita bisa memasukkan field yang berbeda seperti age, city, dan occupation.

Insert Data

| | |
|---|---|
| insertOne | Memasukan satu dokumen ke dalam collection |
| db.customers.insertOne({id : "Astrid" }) | memasukan data dengan id Astrid |
| insertMany | Memasukan banyak dokumen ke dalam collection |
| db.customers.insertMany([{id : "Astrid" }, {id : "Mark" }]) | Memasukan data dengan id Astrid dan Mark |
| insert | Memasukan satu/banyak dokumen ke dalam collection |
| db.customers.insert([{ id : "Fernanda"}, {id : "Mark"}]) | Memasukan data dengan id Fernanda dan Mark |

7. Update Data. Untuk mengubah data kita bisa menggunakan query.

```
db.<namacollection>.update({<kriteria>}, {$set:{<nilai baru>}})
```

8. Kita ingin mengubah nama dengan id "Jennifer" dari Jennifer Peterson menjadi Jennifer Aniston.

```
db.customers.update(
  {_id: "Jennifer"}, 
  {$set: {fullname: 'Jennifer Aniston'}})
```

```
_id: ObjectId("60662af48b14a546b91b8e18")
id: "Mark"
fullname: "Mark Phillips"
email: "mphilips12@shaw.ca"
city: "San Francisco"
```

```
_id: ObjectId("60662af48b14a546b91b8e19")
id: "Jennifer"
fullname: "Jennifer Aniston"
email: "jennifer@rogers.ca"
occupation: "teacher"
```

Update Data

| | |
|---|--|
| updateOne | Mengubah nilai pertama yang ditemukan oleh filter dengan nilai pada set mengubah data age menjadi 24 pada id Astrid |
| updateMany | Mengubah semua nilai yang ditemukan oleh filter dengan nilai pada set mengubah data age menjadi 24 pada semua id Astrid |
| update | Mengubah nilai pertama yang ditemukan oleh filter dengan nilai pada set. (mongodb < version 5) Memasukkan data dengan id Fernanda dan Mark |
| replaceOne | Menghapus dan mengganti data sesuai filter dengan data kedua (delete lalu update) Menghapus data Astrid lalu menggantinya dengan data Anata |
| db.customers.replaceOne({id: "Astrid"}, {id: "Anata"}) | |

9. Delete Data. Untuk menghapus data dari tabel dapat menggunakan query

```
db.<nama collection>.remove(<query>)
```

misalkan kita ingin menghapus data dengan id = Jennifer

```
db.customers.remove(
  {_id: "Jennifer"} )
```

```
_id: ObjectId("6066273448ee35c9edf4e1ad")
id: "Astrid"
fullname: "Astrid Gruber"
email: "astrid.gruber@apple.at"
```

```
_id: ObjectId("60662af48b14a546b91b8e17")
id: "Fernanda"
fullname: "Fernanda Ramos"
email: "fernadaramos4@uol.com.br"
age: 24
```

```
> _id: ObjectId("60662af48b14a546b91b8e18")
id: "Mark"
fullname: "Mark Phillips"
email: "mphilips12@shaw.ca"
city: "San Francisco"
```

Delete Data

| | |
|--|--|
| deleteOne | menghapus nilai pertama yang ditemukan oleh filter |
| db.customers.deleteOne({ id : "Astrid" }) | menghapus data dengan id Astrid |
| deleteMany | menghapus semua nilai yang ditemukan oleh filter |
| db.customers.deleteMany({ id : "Astrid" }) | menghapus semua data dengan id Astrid |
| remove | menghapus nilai pertama yang ditemukan oleh filter |
| db.customers.remove({ id : "Astrid" }) | menghapus data dengan id Astrid |

3.5. Tugas dan Latihan

1. Buatlah database rentalfilm di mongoDB seperti yang telah dilakukan di pertemuan pertama.
2. Masukkan tiga buah data di setiap collections pada database rentalfilm menggunakan query.
3. Lakukan minimal dua buah query update data dan delete data.

MODUL 4

TIPE DATA DAN RELASI PADA MONGODB

4.1. Tujuan Praktikum

1. Mengenal tipe data pada MongoDB
2. Mengenal relasi pada MongoDB
3. Mengenal Lookup

4.2. Dasar Teori

4.2.1. Tipe Data MongoDB

Document didalam MongoDB dapat dianggap sebagai "JSON" karena secara konseptual mirip dengan object pada JavaScript. Perlu untuk diketahui bahwa JSON memiliki keterbatasan dalam hal dukungan tipe data yang diantaranya hanyalah: null, boolean, numeric, string, array, dan object. Dalam hal ini, MongoDB tersedia dengan dukungan tambahan terhadap tipe data dengan tetap membawa sifat dasar dari JSON itu sendiri.

Mongodb mempunyai beberapa tipe data dalam penyimpanannya. Tipe – tipe datanya sebagai berikut :

1. String

Tipe data string adalah tipe data yang digunakan untuk menyimpan sebuah kata – kata pada database. Contohnya : { nama : “Basis Data Lanjut” }

2. Int

Tipe data int atau integer adalah tipe data yang digunakan untuk menyimpan sebuah angka pada database. Disini untuk integer di bagi ke menjadi 2 yaitu int dan number long, perbedaannya adalah dari segi banyaknya penyimpanan untuk int bisa menampung data kurang lebih "2.000.000.000" sedangkan untuk number long bisa menampung data kurang lebih "9.000.000.000.000.000.000". Contohnya : { harga : 1000000 }. (untuk angka tidak menggunakan kutip 2, yang menggunakan kutip 2 hanya untuk tipe data string. Untuk tipe data ini juga jangan menulis menggunakan titik).

3. Decimal

Tipe data decimal adalah tipe data yang digunakan untuk menyimpan sebuah angka pecahan pada database. Contohnya : { nilai : 50.5 }. (*untuk koma disini menggunakan titik, bukan menggunakan koma*).

4. Boolean

Tipe data boolean adalah tipe data yang hanya mengenal “true” dan “false” atau benar dan salah saja. Contohnya { status_aktif : true } (*untuk penulisan true atau false disini juga tidak memakai kutip 2*)

5. Date

Tipe data date adalah tipe data yang digunakan untuk menyimpan sebuah data tanggal pada database. Contohnya { tanggal_masuk : ISO("2018-11-20") } untuk tanggal yang sudah di tetapkan atau kalau ingin menyimpan tanggal sekarang bisa menggunakan new date() contohnya { hari_ini : new Date() }.

6. Array

Tipe data array adalah tipe data yang digunakan untuk menyimpan beberapa data dalam 1 kolom pada database. Contohnya { hobi : ["sepak bola", "renang"] }. Array bisa digunakan untuk banyak hal salah satunya adalah untuk menyimpan daftar data untuk sebuah field seperti yang telah di contohkan.

7. Object

Tipe data object adalah tipe data yang digunakan untuk menyimpan beberapa data dalam 1 kolom pada database (hampir sama dengan array tetapi kalau array data yang ada di dalamnya kita bisa menyebutnya hobi[0] untuk baris pertama nah sedangkan untuk object ini kita bisa memberikan nama data pada setiap barisnya). Contohnya

```
{  
    status : {  
        status_aktif : true,  
        status_lulus : false  
    }  
}
```

8. ObjectId

Tipe data objectid adalah tipe data yang digunakan untuk menyimpan id pada database. Contohnya

```
{ _id : ObjectId("4beffd2be5896hror92k123a") }.
```

Secara default MongoDB akan membuat ObjectId pada _id pada setiap data baru yang ditambahkan. ObjectId ini berupa 12-byte BSON (binary JSON) hexadesimal.

4 byte pertama pada ObjectId merupakan timestamp kapan waktu ObjectId itu dibuat dan 5 byte setelahnya adalah angka acak.

4.2.2. Relasi MongoDB

Relasi adalah suatu hubungan antar tabel pada database dimana suatu tabel mempunyai data yang sama dengan data yang ada di tabel lainnya. Terdapat 3 macam relasi yaitu :

1. Relasi one to one

Hubungan antar tabel dimana tabel A adalah data master dan tabel B harus mempunyai data yang ada pada tabel A

2. Relasi one to many

Sebuah hubungan antara tabel dimana tabel A memiliki sebuah data yang bisa dipakai pada tabel B data tersebut bisa banyak data atau beberapa data saja.

3. Relasi many to many

Sebuah hubungan antara tabel dimana tabelnya ini ada banyak. Hubungannya itu bisa banyak tabel misalkan tabel A dengan tabel B dan tabel C dengan tabel B seperti itu.

4.2.3. Lookup

Lookup adalah sebuah metode yang digunakan untuk menampilkan data seperti find tetapi di lookup ini kita dapat menampilkan beberapa tabel untuk ditampilkan (lookup ini bisa berjalan kalau tabelnya sudah berrelasi). Secara sederhana Lookup merupakan cara kita untuk melakukan Join antar collections dan perintah ini akan menambahkan array field baru dari setiap data dari collection lain yang ditambahkan berdasarkan kondisinya.

```
{ _id : ObjectId("4beffd2be5896hror92k123a") }.
```

```
db.collection.aggregate({  
    $Lookup : {  
        from : "collection2",  
        localField : "_id",  
        foreignField : "_id",  
        as : "join"  
    }  
})
```

Keterangan :

Aggregate : untuk mengelompokan data

Lookup : untuk mengabungkan data

From : diisi tabel ke 2 yang akan ditampilkan

localField : diisi dengan nama data yang ada di tabel pertama (data ini harus data yang nanti sama dengan data yang ada di tabel ke 2)

foreignField : sama dengan localField tetapi untuk foreignField diisi dengan nama data yang ada di tabel kedua

as : as atau alias dapat diisi bebas karena disini jika nanti data kita mau ditampilkan kita cukup memanggil nama yang sudah di aliaskan saja.

4.2.3. Validasi

Validasi adalah sebuah metode untuk pengecekan suatu data yang dimasukkan, misalkan pada saat kita login pada suatu website kita ketikan asal – asalan maka akan muncul peringatan.

Validasi bisa ditambahkan ketika kita sedang membuat collection contohnya:

```
db.createCollection ("students", {  
    $validator :  
        {$jsonSchema : {  
            bsonType: "object",  
            title: "Student Object Validation",  
            required: [ "alamat", "jurusan", "nama", "tahun" ],  
            properties: {  
                nama: {  
                    bsonType: "string",  
                    description: "'nama' harus dalam rupa string dan wajib diisi"  
                },  
                tahun: {  
                    bsonType: "int",  
                    minimum: 2016,  
                    maximum: 3016,  
                    description: "tahun harus berupa angka dan tidak kurang dari 2016 dan tidak lebih  
dari 3016"  
                },  
                ipk: {  
                    bsonType: [ "double" ],  
                    description: "'ipk' harus dalam bentuk tipe data double apabila ada pada sebuah  
document"  
                }  
            }  
        }  
})
```

Validasi diatas akan memastikan apabila kita akan menambahkan data ke collection students maka jika tidak memiliki field alamat, jurusan, nama serta tahun maka akan muncul error dan data tidak akan masuk dan juga ketika field nama tidak diisi atau tidak dalam bentuk string akan terjadi error juga begitu pula dengan field tahun harus dalam bentuk integer dan ipk harus dalam bentuk double.

4.3. Software

1. MongoDB
2. Compass
3. MongoDB Shell

4.4. Tahapan Kerja

4.4.1. Tipe Data

1. Buka MongoDB shell dengan melakukan command “mongosh” pada cmd.

```
C:\Users\Deny>mongosh
Current Mongosh Log ID: 6421014b53f1b96d75c75b1f
Connecting to:      mongod://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.7.1
Using MongoDB:     6.0.4
Using Mongosh:    1.7.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2023-03-26T18:19:28.638+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring() To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
-----
tests
```

2. Buat Database baru dengan nama bebas menggunakan query use.

```
test> use Modul3
switched to db Modul3
Modul3>
```

3. Buat collection baru.

```
Modul3> db.createCollection("TipeData")
{ ok:1 }
Modul3>
```

4. Masukkan document baru dengan setiap tipe data sesuai dari bagian dasar teori.

```
Modul3> db.TipeData.insertOne(
{
  Nama_Produk : "Kulkas",
  Harga : 100000000,
  Berat_Produk : 8.9,
  Rusak : false,
  Tanggal_Masuk : new Date(),
  Tag : ["Elektronik", "Perabotan"],
  Dimensi : {
    tinggi : 80,
    panjang : 50,
    lebar : 20
  },
  Id_Product : ObjectId()
}
```

Query diatas menggunakan semua data type yang sudah diterangkan pada dasar teori :

- Nama_Produk menggunakan String
- Harga Menggunakan Integer
- Berat_Produk menggunakan Decimal
- Rusak menggunakan Boolean
- Tanggal_Masuk menggunakan Date
- Tag menggunakan Array untuk menyimpan banyak nilai sekaligus
- Dimensi menggunakan Object untuk menyimpan banyak data
- Id_Produk menggunakan Object id untuk membuat id yang unik.

```

_id: ObjectId('64211d9576ba2ce70b5b89ed')
Nama_Produk: "Kulkas"
Harga: 10000000
Berat_Produk: 8.9
Rusak: false
Tanggal_Masuk: 2023-03-27T04:37:41.539+00:00
Tag: Array
  0: "Elektronik"
  1: "Perabotan"
Dimensi: Object
  tinggi: 80
  panjang: 50
  lebar: 20
Id_Produk: ObjectId('64211d9576ba2ce70b5b89ec')

```

4.4.2. Lookup

1. Buka CMD lalu ketik mongosh untuk mengakses mongoDB shell.

```

C:\Users\Deeomy\mongosh
Current Mongosh log ID: 6421014b53f1b9667c75b1f
Using MongoDB: 4.0.4
Using Mongosh: 1.7.1
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
.....
The server generated these startup warnings when booting
2023-03-26T18:19:28.618497+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
.....
Enable Mongosh's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. Mongosh may use this information to make product
improvements and to suggest Mongosh products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring() To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
.....
test>

```

2. Lalu ganti database yang dipilih ke database rentalfilm yang telah dibuat sebelumnya dengan menggunakan query “use”.

```

test> use rentalfilm
switched to db rentalfilm
rentalfilm>

```

3. Disini kita akan melakukan query lookup sederhana dimana kita akan mengambil data “Country” beserta data “City” dengan melakukan join.

```

rentalfilm>db.country.aggregate({
  $lookup:{
    "city"
    localField: "id",
    foreignField: "CountryID",
    as :"join"
  }
});

```

4. Query diatas akan menampilkan data dari collection Country beserta data city yang memiliki relasi berdasarkan field Country ID.

```

[ {
  "_id: 1,
  country: 'Indonesia',
  last_update: '2023-03-15T10:22:52.525+00:00',
  join: [
    {
      _id: 1,
      CountryID: 1,
      country: 'Bogor',
      last_update: '2023-03-15T10:22:52.525+00:00'
    },
    {
      _id: 3,
      CountryID: 1,
      country: 'Jakarta',
      last_update: '2023-03-15T10:22:52.525+00:00'
    },
    {
      _id: 4,
      CountryID: 1,
      country: 'Bandung',
      last_update: '2023-03-15T10:22:52.525+00:00'
    }
  ],
  {
    _id: 2,
    country: 'Canada',
    last_update: '2023-03-15T10:22:52.525+00:00',
    join: [
      {
        _id: 2,
        CountryID: 2,
        country: 'Toronto',
        last_update: '2023-03-15T10:22:52.525+00:00'
      },
      {
        _id: 5,
        CountryID: 2,
        country: 'Montreal',
        last_update: '2023-03-15T10:22:52.525+00:00'
      }
    ]
  }
]

```

4.4.3. Validasi

1. Buka kembali mongo shell dengan perintah mongosh pada cmd.

```

C:\Users\Deny>mongosh
Current Mongosh Log ID: 6421203861aa5a17d78ab75c
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.7.1
Using MongoDB: 6.0.4
Using Mongosh: 1.7.1

For mongosh info see: https://docs.mongodb.com/mongosh-shell/
-----
The server generated these startup warnings when booting
2023-03-26T18:19:28.638+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
-----
test>

```

2. Lalu pilih lagi database rental film.

```
rentalfilm>db.country.aggregate({  
  $lookup:{  
    "city"  
    localField: "id",  
    foreignField: "CountryID",  
    as : "join"  
  }  
});
```

```
test> use rentalfilm  
switched to db rentalfilm  
rentalfilm>
```

3. Buat collection baru dengan nama continent beserta validator

```
rentalfilm>db.createCollection("continent", {  
  validator:{  
    $jsonSchema: {  
      bsonType: "object",  
      title: "continent validator",  
      required: ["continent", "last_update"],  
      properties: {  
        continent: {  
          bsonType: "string",  
          description: "'continent' harus dalam rupa string dan wajib  
diisi"  
        },  
      }  
    }  
  }  
  { pk: 1 }  
Rentalfilm>
```

4. Lakukan testing validator dengan memasukan data yang tidak valid.

```
rentalfilm>db.continent.insertOne({cont : "Asia", last_update : ISODate()})
```

```
rentalfilm> db.continent.insertOne({cont : "Asia", last_update : ISODate()})  
Uncaught:  
MongoServerError: Document failed validation  
Additional information: {  
  failingDocumentId: ObjectId("64210dfc76ba2ce70b5b89e6"),  
  details: {  
    operatorName: '$jsonSchema',  
    title: 'continent validator',  
    schemaRulesNotSatisfied: [  
      {  
        operatorName: 'required',  
        specifiedAs: { required: [ 'continent', 'last_update' ] },  
        missingProperties: [ 'continent' ]  
      }  
    ]  
  }  
}
```

5. Lakukan testing validator dengan memasukan data yang valid

```
rentalfilm> db.continent.insertOne({continent : "Asia", last_update :  
ISODate()})  
{  
  Acknowledged: true,  
  insertedID: ObjectId("64210e2476ba2ce705b89e7")
```

4.5. Tugas dan Latihan

1. Buatlah serta jalankan query untuk membuat lebih dari 1 document baru yang menggunakan semua jenis tipe data.
2. Buatlah collection baru serta tambahkan validator dengan minimal 2 field.
3. Buat dan jalankan query untuk menambahkan data yang valid kedalam collection yang sudah ada validatornya.
4. Buat dan jalankan query untuk menambahkan data yang tidak valid kedalam collection yang sudah ada validatornya.
5. Buat dan jalankan dua buah query lookup pada database rentalfilm yang telah dibuat selain pada collection country dan city.

MODUL 5

DATA MODEL DAN QUERY

5.1. Tujuan Praktikum

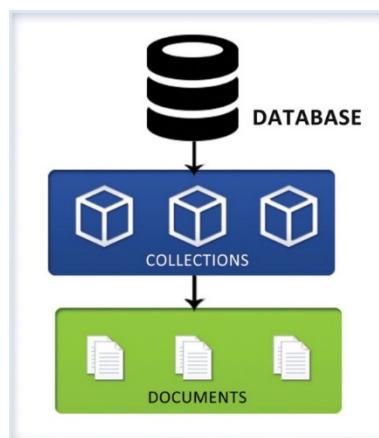
1. Mengenal data model pada mongodb
2. Mahasiswa Menguasai query dasar dan query kompleks

5.2. Dasar Teori

5.2.1. Data Model MongoDB

Pada bab sebelumnya, kita telah mempelajari bahwa MongoDB adalah sistem basis data berbasis dokumen di mana dokumen dapat memiliki skema yang fleksibel. Ini berarti bahwa dokumen dalam koleksi dapat memiliki rangkaian tipe data yang berbeda (atau sama). Ini memberi lebih banyak fleksibilitas ketika berhadapan dengan data.

MongoDB dapat memiliki banyak database. Setiap database adalah satu set koleksi. Koleksi mirip dengan konsep tabel di SQL; Namun, mereka tanpa skema. Setiap koleksi dapat memiliki beberapa dokumen. Pikirkan dokumen sebagai baris dalam SQL. Gambar 4.1 menggambarkan model database MongoDB.



Gambar 19. model database MongoDB

Dalam sistem RDBMS, karena struktur tabel dan tipe data untuk setiap kolom adalah tetap, Anda hanya dapat menambahkan data dari tipe data tertentu dalam kolom. Di MongoDB, koleksi adalah kumpulan dokumen tempat data disimpan sebagai pasangan nilai kunci. Mari kita pahami dengan contoh bagaimana data disimpan dalam dokumen. Dokumen berikut berisi nama dan nomor telepon pengguna:

```
{"Nama": "ABC", "Telepon": ["1111111", "222222"] }
```

Skema dinamis berarti bahwa dokumen dalam koleksi yang sama dapat memiliki kumpulan bidang atau struktur yang sama atau berbeda, dan bahkan bidang umum dapat menyimpan berbagai jenis nilai di seluruh dokumen. Tidak ada kekakuan dalam cara penyimpanan data dalam dokumen koleksi.

Mari kita lihat contoh koleksi Wilayah:

```
{ "R_ID" : "REG001", "Nama" : "Indonesia" }  
{ "R_ID" :1234, "Nama" : "Jakarta" , "Negara" : "Indonesia" }
```

Dalam kode ini, Anda memiliki dua dokumen di koleksi Wilayah. Meskipun kedua dokumen tersebut merupakan bagian dari satu koleksi, mereka memiliki struktur yang berbeda: koleksi kedua memiliki atribut informasi tambahan, yaitu **Negara**. Faktanya, jika Anda melihat atribut "R_ID", ini menyimpan nilai tipe data STRING di dokumen pertama, sedangkan dokumen kedua adalah angka. Jadi dokumen koleksi dapat memiliki skema yang sama sekali berbeda. Semua tergantung pada aplikasi mau menyimpan dokumen dalam koleksi tertentu bersama-sama atau memiliki banyak koleksi.

5.2.2. JSON dan BSON

MongoDB adalah database berbasis dokumen. MongoDB menggunakan Binary JSON untuk menyimpan datanya. Notasi adalah standar yang digunakan untuk pertukaran data di Web modern saat ini (bersama dengan XML). Formatnya dapat dibaca oleh manusia dan mesin. Ini bukan hanya cara yang bagus untuk bertukar data tetapi juga cara yang bagus untuk menyimpan data.

Semua tipe data dasar (seperti string, angka, nilai Boolean, dan array) didukung oleh JSON. Kode berikut menunjukkan seperti apa dokumen JSON:

```
// "Document collections" - "HTMLPage" document  
{  
    _id: 1,  
    title: "Hello",  
    type: "HTMLpage",  
    text: "<html>Hi..Welcome to my world</html>"  
}  
...  
// Document collection also bisa menyimpan dokumen "Gambar"  
{  
    _id: 3,  
    title: "Family Photo",  
    type: "JPEG",  
    sizeInMB: 10,.....  
}
```

Skema ini tidak hanya memungkinkan untuk menyimpan data terkait dengan struktur berbeda secara bersamaan dalam **Collection** yang sama, tetapi juga menyederhanakan query. Collection yang sama dapat digunakan untuk melakukan query pada atribut umum seperti mengambil semua konten yang diunggah pada tanggal dan waktu tertentu serta query pada atribut tertentu seperti menemukan gambar dengan ukuran lebih besar dari X MB. Jadi pemrograman berorientasi objek adalah salah satu contoh kasus penggunaan database dengan skema polimorfik.

5.2.3. Konsep Method, Filter, Operator

Untuk penulisankn rumus pada mongodb dibutuhkan ketiga script ini yaitu Method, Filter, dan Operator. Method adalah metode yang di pakai sebagai aksi yang akan digunakan misalkan `find()`, `insertOne()`, `deleteOne()`, dll. Untuk filter adalah sebuah fungsi untuk menyaring sebuah data agar datanya tersebut bisa di kelompokan dan tidak tampil semua pada saat kita mau menampilkannya contoh `find({ umur : 20 })`. Sedangkan untuk operator adalah sebuah fungsi yang digunakan agar menyaringan data pada filter bisa lebih spesifik lagi misalkan seperti contoh pada filter disitu hanya menyaring data umur yang 20 tahun saja sedangkan ketika kita memkasi operator disini bisa menyaring bedasarkan umur yang lebih dari 20 atau di bawah 20 tahun dan dll. Contohnya :

```
db.mahasiswa.find({ umur : { $gt : 20 } })
```

- █ = Method
- █ = Filter
- █ = Operator

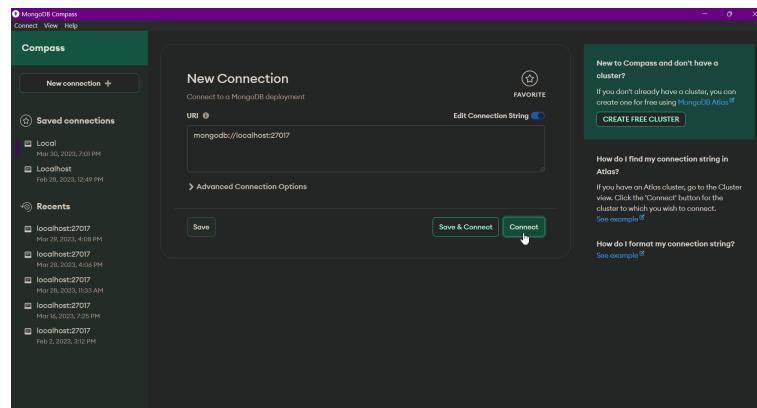
5.4. Software

1. MongoDB
2. Compass
3. MongoDB Shell

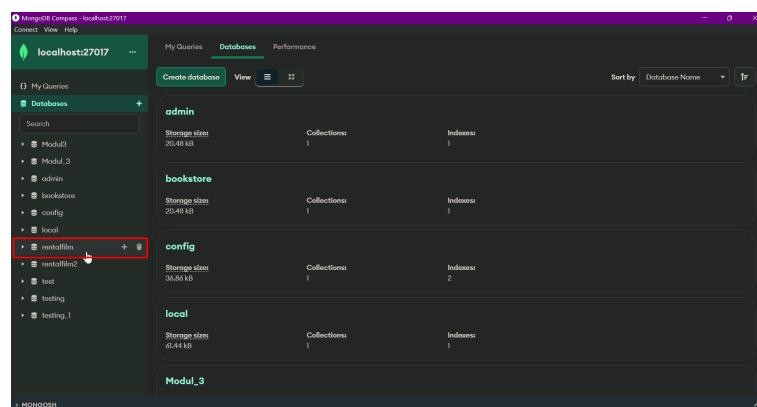
5.5. Tahapan Kerja

5.4.1. Menjalankan MongoDB di Compass dan Command Prompt

1. Buka Aplikasi MongoDB Compass kemudian klik “Connect”



2. Untuk melanjutkan ketahapan kerja selanjutnya pastikan database “Rentalfilm” sudah ada di MongoDB.



3. Buka MongoShell dengan melakukan perintah “mongosh” di dalam command prompt.

```
C:\Users\Deny>mongosh
```

4. Lalu ganti ke database rental film dengan menggunakan perintah “use”.

```
test> use rentalfilm
switched to db rentalfilm
rentalfilm>
```

5.4.2. Query Dasar

1. Untuk mencari data dan menampilkan data tersebut kita dapat menggunakan query:

```
db.<nama collection>.find()
```

Untuk melihat semua isi customers maka kita ketikkan

```
db.customers.find()
```

Ketik query tersebut pada cmd

```
test> use rentalfilm
switched to db rentalfilm
rentalfilm> db.customers.find()
```

```

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
...
> use rentalfilm
switched to db rentalfilm
> db.customers.find()
{
  "_id" : ObjectId("6066273d48ee35c9edf4e1ad"),
  "id" : "Astrid",
  "fullname" : "Astrid Gruber",
  "email" : "astrid.gruber@apple.at"
},
{
  "_id" : ObjectId("60662af48b14a546b91b8e17"),
  "id" : "Fernanda",
  "fullname" : "Fernanda Ramos",
  "email" : "fernadaramos4@uol.com.br",
  "age" : 24
},
{
  "_id" : ObjectId("60662af48b14a546b91b8e18"),
  "id" : "Mark",
  "fullname" : "Mark Philips",
  "email" : "mphilips12@shaw.ca",
  "city" : "San Francisco"
}
>

```

- Untuk mencari nilai tertentu kita bisa mengisikan pasangan key dan value di dalam kurung. Misalkan kita ingin mencari data dengan usia 24

```
db.customers.find( {age:24} ).pretty()
```

```

> use rentalfilm
switched to db rentalfilm
> db.customers.find()
{
  "_id" : ObjectId("6066273d48ee35c9edf4e1ad"),
  "id" : "Astrid",
  "fullname" : "Astrid Gruber",
  "email" : "astrid.gruber@apple.at"
},
{
  "_id" : ObjectId("60662af48b14a546b91b8e17"),
  "id" : "Fernanda",
  "fullname" : "Fernanda Ramos",
  "email" : "fernadaramos4@uol.com.br",
  "age" : 24
},
{
  "_id" : ObjectId("60662af48b14a546b91b8e18"),
  "id" : "Mark",
  "fullname" : "Mark Philips",
  "email" : "mphilips12@shaw.ca",
  "city" : "San Francisco"
}
> db.customers.find( {age:24} )
{
  "_id" : ObjectId("60662af48b14a546b91b8e17"),
  "id" : "Fernanda",
  "fullname" : "Fernanda Ramos",
  "email" : "fernadaramos4@uol.com.br",
  "age" : 24
}
> db.customers.find( {age:24} ).pretty()

{
  "_id" : ObjectId("60662af48b14a546b91b8e17"),
  "id" : "Fernanda",
  "fullname" : "Fernanda Ramos",
  "email" : "fernadaramos4@uol.com.br",
  "age" : 24
}
>

```

method pretty() membuat tampilan dari data kita menjadi lebih rapi layaknya penulisan dictionary pada python. Seperti disebutkan di awal, tidak ada konsep join pada MongoDB setidaknya secara langsung. Akan tetapi, query-query yang kita gunakan lebih sederhana dan mudah dimengerti layaknya saat kita bermain dengan object pada python.

5.4.3. Query Kompleks

- \$in & \$nin

\$in adalah sebuah operator yang berfungsi hampir menyerupai sama dengan (=) tetapi kalau dalam operator ini kita bisa menampilkan data lebih dari 1 data. Perintah dasarnya adalah seperti ini :

```
db.tabel.find({ data : { $in : [ nama1,nama2] } }).pretty()
```

Contohnya pada command prompt :

```
Rentalfilm> db.Customer.find({Active : {$in : [ "Y", "N" ]}})
```

```

rentalfilm> db.Customer.find({Active : {$in : ["Y", "N"]}})
[
  {
    _id: 1,
    AddressId: 1,
    AddressColumn: 2,
    FirstName: 'Denny',
    LastName: 'Partala',
    Email: 'email@gmail.com',
    Active: 'Y',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 2,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Otoso',
    LastName: 'Bearu',
    Email: 'email2@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 3,
    AddressId: 3,
    AddressColumn: 3,
    FirstName: 'Bjorn',
    LastName: 'Mato',
    Email: 'email3@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  }
]

```

Sedangkan untuk \$nin adalah kebalikan dari \$in yang berfungsi untuk menampilkan data yang tidak ditampilkan pada saat di \$in di jalankan. Perintah dasarnya adalah seperti ini:

```
db.tabel.find({ data : { $in : [ nama1,nama2] } }).pretty()
```

Contohnya pada command prompt :

```

rentalfilm> db.Customer.find({Active : {$nin : ["Y", "N"]}}).pretty()
[
  {
    _id: 4,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Mitch',
    LastName: 'Benjamin',
    Email: 'email4@gmail.com',
    Active: 'T',
    CreateDate: '2023-03-15T10:22:52.525+00:00',
    LastUpdate: '2023-03-15T10:22:52.525+00:00'
  }
]

```

2. \$and, \$or, \$not

And, or , not adalah sebuah operator yang digunakan untuk menyaring data dengan syarat tertentu agar datanya bisa ditampilkan. \$and adalah sebuah operator yang digunakan untuk menyaring data dengan 2 atau lebih kriteria dimana semua kriterianya itu harus benar untuk menampilkan datanya. \$or adalah sebuah operator yang digunakan untuk menyaring data dengan 2 atau lebih kriteria dimana jika salah satu kriteria terpenuhi maka akan menampilkan datanya. \$not adalah sebuah operator yang digunakan untuk menyaring data dengan menampilkan kebalikan dari hasil yang seharusnya. Ketentuan dari \$and dan \$or adalah seperti ini :

| Kriteria 1 | Kriteria 2 | \$and | \$or |
|------------|------------|-------|-------|
| Benar | Benar | Benar | Benar |
| Salah | Salah | Salah | Salah |
| Salah | Benar | Salah | Benar |
| Benar | Salah | Salah | Benar |

Perintah dasar \$and seperti ini:

```
db.tabel.find( { $and : [ { data1 : isi1 },{ data2 : isi2 } ] } ).pretty()
```

```
rentalfilm> db.Customer.find( { $and : [ { AddressId : 2, Active : "T"}] } )
[ {
  _id: 4,
  AddressId: 2,
  AddressColumn: 2,
  FirstName: 'Mitch',
  LastName: 'Benjamin',
  Email: 'email4@gmail.com',
  Active: 'T',
  CreateDate: '2023-03-15T10:22:52.525+00:00',
  LastUpdate: '2023-03-15T10:22:52.525+00:00'
}]
```

Perintah dasar \$or seperti ini:

```
db.tabel.find( { $or : [ { data1 : isi1 },{ data2 : isi2 } ] } ).pretty()
```

```
rentalfilm> db.Customer.find({$or : [{Active : "Y"}, {AddressId : 3}]})
[ {
  _id: 1,
  AddressId: 1,
  AddressColumn: 2,
  FirstName: 'Denny',
  LastName: 'Partala',
  Email: 'email1@gmail.com',
  Active: 'Y',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
{
  _id: 3,
  AddressId: 3,
  AddressColumn: 3,
  FirstName: 'Bjorn',
  LastName: 'Moto',
  Email: 'email3@gmail.com',
  Active: 'N',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
}]
```

Perintah dasar \$not seperti ini:

```
db.tabel.find( {data : { $not : { $eq : isi } } } ).pretty()
```

```
rentalfilm> db.Customer.find({AddressId: {$not : {$eq : 1 }}})
```

```

rentalfilm> db.Customer.find( {AddressId: { $not : {$eq : 1}}}) 
[ 
  {
    _id: 2,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Otoso',
    LastName: 'Bearu',
    Email: 'email2@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 3,
    AddressId: 3,
    AddressColumn: 3,
    FirstName: 'Bjorn',
    LastName: 'Mato',
    Email: 'email3@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 4,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Mitch',
    LastName: 'Benjamin',
    Email: 'email4@gmail.com',
    Active: 'T',
    CreateDate: '2023-03-15T10:22:52.525+00:00',
    LastUpdate: '2023-03-15T10:22:52.525+00:00'
  }
]

```

Keterangan :

- \$eq adalah equal atau sama dengan(=)
- \$ne adalah not equal atau tidak sama dengan(!=)
- \$eq atau \$ne rumus yang harus digunakan dalam menggunakan \$not kalau tidak di gunakan salah satu rumus ini maka akan error.

3. \$exists, \$type, \$regex

Operator exists adalah operator yang digunakan untuk pengecekan data apakah data di suatu tabel ada atau tidak. Contohnya pada suatu tabel barang terdapat data gelas dan piring. Pada data gelas terdapat sebuah data diskon tetapi pada data piring tidak ada data diskon nah pada kasus ini kita dapat memakai \$exists untuk mengecek keberadaan suatu datanya. Perintah dasarnya adalah sebagai berikut :

```
db.tabel.find( {data : { $exists : true } } ).pretty()
```

```
rentalfilm> db.Customer.find({Email: {$exists : true }})
```

```

rentalfilm> db.Customer.find( {Email : { $exists: true}})
[ {
    _id: 1,
    AddressId: 1,
    AddressColumn: 2,
    FirstName: 'Denny',
    LastName: 'Partala',
    Email: 'email@gmail.com',
    Active: 'Y',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
{
    _id: 2,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Otoso',
    LastName: 'Bearu',
    Email: 'email2@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
{
    _id: 3,
    AddressId: 3,
    AddressColumn: 3,
    FirstName: 'Bjorn',
    LastName: 'Mato',
    Email: 'email3@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
{
    _id: 4,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Mitch',
    LastName: 'Benjamin',
}
]

```

Contoh diatas merupakan perintah untuk mencari semua data yang dimana field “Email” ada.

\$type

Operator type adalah operator yang digunakan untuk menyaring atau mensortir data berdasarkan suatu tipe data pada sebuah data tertentu. Contohnya pada tabel barang terdapat price yang bertipe data number lalu kita akan mencarinya berdasarkan type data number maka akan muncul data barang apabila kita mencarinya berdasarkan type data string maka hasilnya tidak akan muncul. Perintah dasarnya adalah sebagai berikut:

```
db.tabel.find( {data : { $type : "tipe-datanya" } } ).pretty()
```

```
rentalfilm> db.Customer.find({FirstName: {$type : "string" }})
```

```

rentalfilm> db.Customer.find( {FirstName: {$type : "string"} } )
[
  {
    _id: 1,
    AddressId: 1,
    AddressColumn: 2,
    FirstName: 'Denny',
    LastName: 'Partala',
    Email: 'email@gmail.com',
    Active: 'Y',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 2,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Otoso',
    LastName: 'Bearu',
    Email: 'email2@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 3,
    AddressId: 3,
    AddressColumn: 3,
    FirstName: 'Bjorn',
    LastName: 'Mato',
    Email: 'email3@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 4,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Mitch',
    LastName: 'Benjamin',
    Email: 'email4@gmail.com',
  }
]

```

\$regex

Operator regex adalah operator yang digunakan untuk mencari sebuah data berdasarkan kata atau huruf tertentu. Misalkan kita mempunyai data mata kuliah dengan nama “Basis Data Lanjut” lalu kita mau mencari data yang Namanya Basis Data maka kita bisa menggunakan operator ini. Contoh penulisan operator regex :

```

db.tabel.find( {data : { $regex : /isi-yang-mau-dicari/ , $options : '<options>' } } ).pretty()

```

```

rentalfilm> db.users.find( {name : {$regex : /^B/, $options : 'm'} } )
[
  {
    _id: 5,
    name: 'Bosco',
    status: 1,
    dob: ISODate("2023-04-04T04:09:26.587Z"),
    gender: 'M',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z"),
    nama: 'Dendol'
  }
]

```

Keterangan :

- \$options bisa di isi dengan berbagai jenis value contohnya : ‘m’ ketika ingin menggunakan simbol anchor seperti ^, &.
- isi-yang-mau-dicari bisa diisi dengan nama atau huruf yang dicari misalkan mau mencari basisdatalanjut kita bisa menulis /basisdatalanjut/.

Selain // kita juga dapat menggunakan :

/^ isi/ = untuk pencarian nama sesuai huruf depanya misalkan pada pencarian di atas kita menulisnya /**Basis**/

/isi\$/ = untuk pencarian nama sesuai huruf belakang misalnya /**Lanjut&**/ untuk pencarian “Basis Data”

4. . \$expr

Operator expr adalah sebuah operator yang befungsi untuk membandingkan suatu data tertentu. Misalkan kita mempunyai data harga dan data modal pada sebuah tabel, Lalu kita cek apakah harga melebihi dari modal yang diberikan kalau iya maka datanya akan muncul tetapi jika tidak maka dia tidak akan muncul. Perintah dasarnya adalah seperti ini:

```
db.tabel.find( {$expr : { $gt : ["$harga","$modal"] } } ).pretty()
```

```
rentalfilm> db.Film.find({$expr : {$gt : ["$Rental_Duration", "$Replacement_Cost"]}})  
[  
  {  
    _id: 1,  
    LanguageID: 1,  
    Title: 'Country Bear',  
    Description: '-',  
    Release_Year: 2023,  
    Rental_Duration: 1608,  
    Rental_Rate: 10.2,  
    Length: 90,  
    Replacement_Cost: 1200,  
    Rating: 9,  
    Last_Update: '2023-03-15T10:22:52.525+00:00',  
    Special_Features: 'Bears',  
    Fulltext: 'Bear Bear Bear'  
  }  
]
```

Keterangan :

- \$gt adalah operator disini kita dapat menggunakan operator lain sesuai kebutuhan.
- \$harga dan \$modal adalah sebuah nama di kolom tabel dan disini dapat kita panggil dengan menggunakan dolar (\$)

5. \$Next

Operator next adalah sebuah operator yang digunakan untuk menampilkan data di mulai data awal terlebih dahulu sampai data terakhir. Perintah dasar dari next adalah seperti ini:

```
db.tabel.find().next()
```

```
rentalfilm> db.users.find().next()  
{  
  _id: 1,  
  name: 'Denny',  
  status: 1,  
  dob: ISODate("1970-01-01T00:00:00.000Z"),  
  gender: 'M',  
  created_at: ISODate("2023-03-15T10:22:52.525Z"),  
  updated_at: ISODate("2023-03-15T10:22:52.525Z")  
}
```

Jika dijalankan perintah di atas pada command prompt keluar data yang pertama tetapi jika kita jalankan lagi maka datanya tersebut akan memunculkan data pertama lagi.

Untuk memunculkan data berikutnya kita memerlukan satu fungsi lagi yaitu menggunakan “const”, Const adalah sebuah penampung data yang apabila datanya sudah diisi maka tidak bisa dirubah kembali rumus pembuatan const adalah :

```
const nama-variabel = isi
```

```
rentalfilm> const FindNext = db.users.find()  
rentalfilm> FindNext
```

Keterangan:

- Untuk nama-variabel bebas kita mau memberikan nama apa asal kita ingat nanti saat pemanggilannya
- Isi ini bisa kita isikan apa saja bebas asal sesuai dengan kebutuhan contoh kalau pada kasus next kita bisa mengisikan db.tabel.find().

Kalau sudah dibuat sebuah variabelnya sekarang coba ketika “nama- variabel.next()” pada command prompt maka akan keluar data yang pertama, jika kita menuliskannya lagi maka akan muncul data seterusnya sampai data terakhir dan apabila sudah sampai data terakhir ingin menuliskannya lagi maka data tersebut tidak tampil.

6. Sort

Operator sort adalah sebuah operator yang digunakan untuk memilah data contoh kita data mengurutkan data dari A sampai Z atau sebaliknya, bisa juga untuk mengurutkan angka dari yang terkecil terlebih dahulu atau yang terbesar terlebih dahulu. Rumus dari sort adalah seperti ini :

```
db.tabel.find().sort({data:-1}).pretty()
```

```
rentalfilm> db.users.find().sort({_id : -1})
```

```

rentalfilm> db.users.find().sort({_id : -1})
[
  {
    _id: 5,
    name: 'Bosco',
    status: 1,
    dob: ISODate("1970-01-01T00:00:00.000Z"),
    gender: 'M',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 4,
    name: 'Ursidae',
    status: 1,
    dob: ISODate("2000-01-12T00:00:00.000Z"),
    gender: 'F',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 3,
    name: 'Fredrica',
    status: 1,
    dob: ISODate("1999-10-11T00:00:00.000Z"),
    gender: 'F',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 2,
    name: 'Wen',
    status: 1,
    dob: ISODate("2001-08-16T00:00:00.000Z"),
    gender: 'M',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 1,
    name: 'Denny',
    status: 1,
    dob: ISODate("1970-01-01T00:00:00.000Z"),
  }
]

```

Keterangan :

- Data adalah sebuah data yang akan di sortir
- -1 adalah pengurutan data berdasarkan descending atau menurun (selain -1 terdapat juga 1 yaitu ascending atau menaik)

7. Limit

Operator limit adalah sebuah operator yang berfungsi untuk memfilter data yang kita ingin tampilkan dengan jumlah yang telah di tentukan sebelumnya. Rumus dari sort adalah seperti ini :

```
db.tabel.find().limit(5).pretty()
```

```

rentalfilm> db.users.find().limit(2).pretty()
[
  {
    _id: 1,
    name: 'Denny',
    status: 1,
    dob: ISODate("1970-01-01T00:00:00.000Z"),
    gender: 'M',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 2,
    name: 'Wen',
    status: 1,
    dob: ISODate("2001-08-16T00:00:00.000Z"),
    gender: 'M',
    created_at: ISODate("2023-03-15T10:22:52.525Z"),
    updated_at: ISODate("2023-03-15T10:22:52.525Z")
  }
]

```

Keterangan :

Jumlah data yang ingin ditampilkan diisi di dalam kurung pada limit misalkan contoh di atas adalah 5.

8. \$Unset

Operator unset adalah sebuah operator yang berfungsi untuk menghapus data pada tabel. Misalkan kita salah memasukan sebuah data contoh pada suatu tabel barang data pertama berisi nama-barang lalu di data yang kedua berisi nama-barang, tipe. Untuk menghapus tipe di data yang ke dua agar data bisa sama dengan data yang pertama. Disini kita bisa menggunakan unset, untuk unset rumusnya adalah seperti ini :

```
db.tabel.updateMany({kriteria}, {$unset : {kriteria2} } )
```

```
rentalfilm> db.users.updateMany({_id : 1}, {$unset : {gender: ""}})  
{  
    acknowledged: true,  
    insertedId: null,  
    matchedCount: 1,  
    modifiedCount: 1,  
    upsertedCount: 0  
}
```

Keterangan :

- Kriteria pertama adalah kriteria yang dipakai untuk update data.
- Kriteria2 adalah kriteria yang dipakai untuk menghapus data tersebut.

9. \$Rename

Operator rename adalah sebuah operator yang berfungsi untuk mengubah nama data pada tabel. Misalkan kita salah memasukan sebuah data contoh pada suatu tabel barang data pertama berisi nama-barang lalu di data yang kedua berisi nma-barang (salah pengetikan) lalu kita bisa merubah nma-barang yang salah pengetika menjadi nama-barang menggunakan operator rename. Untuk perintah dasarnya adalah seperti ini:

```
db.tabel.updateMany({kriteria}, {$rename : {data: "data-baru"} } )
```

```
rentalfilm> db.users.updateMany({_id : 1}, {$rename : {name : "nama"}})  
{  
    acknowledged: true,  
    insertedId: null,  
    matchedCount: 1,  
    modifiedCount: 1,  
    upsertedCount: 0  
}
```

Keterangan :

- Data adalah sebuah nama data yang ada pada suatu tabel
- Data-baru adalah sebuah nama baru untuk merubah nama data awal

10. \$Upsert

Operator upsert adalah sebuah operator yang digunakan untuk menambahkan suatu data jika data yang terupdate tidak di temukan, misalkan pada update disitu ada sebuah kriteria jika kriterianya itu tidak sesuai maka data tersebut akan disimpan secara otomatis. Untuk rumus upsert adalah seperti ini :

```
db.tabel.updateMany({kriteria}, {$set : {data : "isi"} }, { upsert : true } )
```

```
rentalfilm> db.users.updateOne({_id : 10}, {nama : "Dendol", dob: ISODate()}, {upsert: true})
```

```
rentalfilm> db.users.updateOne({_id : 10}, {$set : {nama : "Dendol", dob: ISODate()}}, {upsert: true})
{
  acknowledged: true,
  insertedId: 10,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
```

11. \$ElemMatch

Operator elemMatch adalah sebuah operator yang digunakan untuk mencari data pada suatu data array. Misalkan kita mempunyai data array 82,85,88 dan data kedua 75, 88, 89 lalu kita ingin mencari data yang lebih dari 80 dan kurang dari 85 kita disini bisa menggunakan operator elemMatch. Untuk perintah elemMatch seperti ini:

```
db.tabel.find( { data : { $elemMatch : { $gt : 80, $lt : 85 } } }).pretty()
```

```
db.Elem.find( { results : { $elemMatch : { $gte : 80, $lt : 85 } } })
```

```
testing> db.Elem.find(
...   { results: { $elemMatch: { $gte: 80, $lt: 85 } } }
... )
[ { _id: 1, results: [ 82, 85, 88 ] } ]
testing> ■
```

Keterangan :

Diatas adalah rumus untuk mencari data array yang datanya lebih dari 80 dan kurang dari 85 pada data sebelumnya kan 82,85,88 dan data kedua 75, 88, 89 untuk 75 dan 89 tidak masuk di kriteria tetapi 90 masuk ke dalam kriteria sehingga data array ini bisa dapat ditampilkan.

12. \$Push

Operator push adalah sebuah operator yang digunakan untuk menambahkan data pada sebuah data array. Misalkan kita mempunyai data array 70,80,90 lalu kita ingin

menambahkannya menjadi 70,80,90,100 maka kita bisa menggunakan operator push.

Untuk perintah push 1 data array seperti ini :

```
db.tabel.updateOne({kriteria}, {$push : {data : isi} } )
```

```
db.Elem.updateOne({_id : 1}, {$push : {results : 100} } )
```

```
testing> db.Elem.updateOne({_id : 1}, {$push : {results: 100}})  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}  
testing>
```

Untuk menambahkan lebih dari 1 data array, seperti ini:

```
db.tabel.updateOne({kriteria}, {$push : {data : { $each : [isi,isi,isi]}}} )
```

```
db.Elem.updateOne({_id : 1}, {$push : {results : { $each : [ 160, 1600]}}} )
```

```
testing> db.Elem.updateOne({_id : 1}, {$push : {results: { $each : [ 160, 1600]}}} )  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}  
testing>
```

13. \$Pull

Operator pull adalah sebuah operator yang digunakan untuk menghapus data pada sebuah data array. Misalkan kita mempunyai data array 70,80,90 lalu kita ingin menghapus salah satunya menjadi 70,80 maka kita bisa menggunakan operator pull. Untuk perintah dasar pull seperti ini :

```
db.tabel.updateOne({kriteria}, {$pull : {data : isi-yang-dihapus} } )
```

```
db.Elem.updateOne({_id : 1}, {$pull : {results : 1600} } )
```

```
testing> db.Elem.updateOne({_id : 1 }, {$pull : { results : 1600}})  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}  
testing>
```

14. \$AddToSet

Operator addToSet adalah sebuah operator yang digunakan untuk menambahkan data pada sebuah data array tetapi jika datanya sudah ada pada array tersebut maka datanya tidak akan ditambahkan. Untuk perintah dasar addToSet seperti ini :

```
db.tabel.updateOne({kriteria}, {$addToSet : {data : isi} } )
```

```
db.Elem.updateOne({_id : 1}, {$addToSet : {results : 160} } )
```

```
testing> db.Elem.updateOne({_id : 1}, {$addToSet : { results : 160}})  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 0,  
  upsertedCount: 0  
}  
testing> _
```

5.5. Tugas dan Latihan

1. Buat dan jalankan perintah untuk mencari data dari collection Customer menggunakan minimal 2 buah variasi dari perintah \$in dan \$nin
2. Buatlah dan Jalankan perintah untuk mencari data dari collection Film untuk menampilkan data Title yang diakhiri dengan huruf G.
3. Buatlah dan jalankan perintah untuk menampilkan 2 buah data dari collection Customer.
4. Buatlah dan jalankan perintah untuk menampilkan semua data di film berdasarkan urutan alfabet.
5. Buatlah collection baru dengan nama ArrTest masukan beberapa data yang memiliki setidaknya 1 array kemudian buat dan jalankan perintah Push, Pull dan addToSet secara bergiliran.

MODUL 6

PROJECTION DAN AGGREGATION

6.1. Tujuan Praktikum

1. Mahasiswa mampu menggunakan query projection
2. Mahasiswa mampu menggunakan query Aggregation

6.2. Dasar Teori

6.2.1. Update dan Save Document MongoDB

Metode update() dan save() MongoDB digunakan untuk memperbarui dokumen menjadi koleksi. Metode update() memperbarui nilai dalam dokumen yang ada

Sintaks dasar metode update() adalah sebagai berikut:

```
>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

6.2.2. Remove Document MongoDB

Metode remove() MongoDB digunakan untuk menghapus dokumen dari koleksi. metode remove() menerima dua parameter. Salah satunya adalah kriteria penghapusan dan yang kedua adalah bendera justOne.

- kriteria penghapusan: (Opsional) kriteria penghapusan menurut dokumen akan dihapus.
- justOne: (Opsional) jika disetel ke true atau 1, maka hapus hanya satu dokumen.

Sintaks dasar metode remove() adalah sebagai berikut:

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Jika ada beberapa record dan Anda hanya ingin menghapus record pertama, maka atur hanya satu parameter dalam metode remove().

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Pada mongoDB 6.0 perintah Remove diganti menjadi Delete. Dan variasinya adalah deleteOne() untuk menghapus satu dokumen saja dan deleteMany() untuk menghapus banyak dokumen sekaligus.

6.2.3. Limit Record MongoDB

Untuk membatasi record di MongoDB, Anda perlu menggunakan metode limit(). Metode ini menerima satu argumen tipe angka, yang merupakan jumlah dokumen yang ingin Anda tampilkan.

Sintaks dasar metode limit() adalah sebagai berikut:

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Selain metode limit(), ada satu lagi metode skip() yang juga menerima argumen tipe angka dan digunakan untuk melewaskan jumlah dokumen. Sintaks dasar metode skip() adalah sebagai berikut:

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

6.2.4. Indexing MongoDB

Indeks mendukung resolusi query yang efisien. Tanpa indeks, MongoDB harus memindai setiap dokumen koleksi untuk memilih dokumen yang cocok dengan pernyataan query. Pemindaian ini sangat tidak efisien dan membuat MongoDB harus memproses data dalam jumlah besar.

Indeks adalah struktur data khusus, yang menyimpan sebagian kecil kumpulan data dalam bentuk yang mudah dilintasi. Indeks menyimpan nilai bidang tertentu atau kumpulan bidang, diurutkan berdasarkan nilai bidang sebagaimana ditentukan dalam indeks. Untuk membuat indeks, Anda perlu menggunakan metode createIndex() dari MongoDB. Sintaks dasar metode **createIndex()** adalah sebagai berikut:

```
>db.COLLECTION_NAME.createIndex({KEY:1})
```

Kuncinya adalah nama file yang ingin Anda buat indeksnya, nilai 1 untuk urutan menaik dan nilai -1 untuk membuat indeks dalam urutan menurun. Contoh:

```
>db.mycol.createIndex({"title":1})
```

Dalam metode **createIndex()** Anda dapat mengirim banyak parameter, untuk membuat indeks pada banyak field. Contoh:

```
>db.mycol.createIndex({"title":1,"description":-1})
```

Metode **createIndex()** juga memiliki sejumlah opsi (yang bersifat opsional). Berikut ini adalah daftarnya:

| Parameter | Type | Description |
|--------------------|---------------|---|
| Unique | Boolean | Membuat indeks unik sehingga koleksi tidak akan menerima penyisipan dokumen di mana kunci atau kunci indeks cocok dengan nilai yang ada di indeks. Tentukan true untuk membuat indeks unik. Nilai defaultnya salah. |
| Name | String | Nama indeks. Jika tidak ditentukan, MongoDB menghasilkan nama indeks dengan menggabungkan nama bidang yang diindeks dan menguratkannya. |
| Sparse | Boolean | Jika bernilai true , indeks hanya mereferensikan dokumen dengan bidang tertentu. Indeks ini menggunakan lebih sedikit ruang tetapi berperilaku berbeda dalam beberapa situasi (terutama pengurutan). Nilai defaultnya false . |
| expireAfterSeconds | Integer | Menentukan nilai, dalam hitungan detik, sebagai TTL untuk mengontrol berapa lama MongoDB menyimpan dokumen dalam kumpulan ini. |
| V | Index Version | Nomor versi indeks. Versi indeks default bergantung pada versi MongoDB yang berjalan saat membuat indeks. |
| Weights | Document | Bobotnya adalah angka mulai dari 1 hingga 99.999 dan menunjukkan pentingnya field yang terikat terhadap field lain yang diindeks dalam bentuk skor. |
| default_language | String | Untuk indeks teks, bahasa yang digunakan untuk menentukan kata yang digunakan untuk |

| | | |
|-------------------|--------|---|
| | | menghentikan proses dan memberikan aturan untuk stemmer dan tokenizer. Nilai default adalah english . |
| language_override | String | Untuk indeks teks, tentukan nama field dalam dokumen yang berisi kata yang akan diganti bahasanya. Nilai default adalah language . |

6.2.5. Aggregation MongoDB

Operasi agregasi memproses data record dan mengembalikan hasil yang dihitung. Operasi agregasi mengelompokkan nilai dari beberapa dokumen secara bersamaan, dan dapat melakukan berbagai operasi pada data yang dikelompokkan untuk mengembalikan satu hasil. Dalam SQL, perintah count(*) dan group by setara dengan agregasi mongodb. Sintaks dasar metode **aggregate()** adalah sebagai berikut:

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

Berikut adalah daftar ekspresi agregasi yang tersedia:

| Expression | Description |
|------------|---|
| \$sum | Menjumlahkan nilai yang ditentukan dari semua dokumen dalam koleksi. |
| \$avg | Menghitung rata-rata semua nilai yang diberikan dari semua dokumen dalam koleksi. |
| \$min | Mendapatkan nilai minimum yang sesuai dari semua dokumen dalam koleksi. |
| \$max | Mendapatkan nilai maksimum yang sesuai dari semua dokumen dalam koleksi. |
| \$push | Menyisipkan nilai ke array dalam dokumen yang dihasilkan. |
| \$addToSet | Menyisipkan nilai ke array dalam dokumen yang dihasilkan tetapi tidak membuat duplikat. |

| | |
|---------|---|
| \$first | Mendapat dokumen pertama dari dokumen sumber sesuai dengan pengelompokannya. Biasanya ini hanya digunakan bersama dengan beberapa tahap "\$sort" yang diterapkan sebelumnya. |
| \$last | Mendapat dokumen terakhir dari dokumen sumber sesuai dengan pengelompokannya. Biasanya ini hanya digunakan bersama dengan beberapa tahap "\$sort" yang diterapkan sebelumnya. |

Pipeline Concept. Dalam perintah UNIX, Pipeline berarti kemungkinan untuk menjalankan operasi pada beberapa input dan menggunakan output sebagai input untuk perintah berikutnya dan seterusnya. MongoDB juga mendukung konsep yang sama dalam kerangka agregasi. Ada satu set tahapan yang mungkin dan masing-masing diambil sebagai satu set dokumen sebagai input dan menghasilkan satu set dokumen yang dihasilkan (atau dokumen JSON yang dihasilkan akhir di akhir pipeline). pipeline ini kemudian dapat digunakan untuk tahap berikutnya dan seterusnya.

| Expression | Description |
|------------|--|
| \$project | Digunakan untuk memilih beberapa bidang tertentu dari koleksi. |
| \$match | Ini adalah operasi penyaringan dan dengan demikian dapat mengurangi jumlah dokumen yang diberikan sebagai input ke tahap berikutnya. |
| \$group | Ini melakukan agregasi aktual seperti yang dibahas di atas. |
| \$sort | Mengurutkan dokumen. |
| \$skip | Dengan ini, dimungkinkan untuk melewati daftar dokumen untuk sejumlah dokumen tertentu. |
| \$limit | Ini membatasi jumlah dokumen untuk dilihat, dengan nomor yang diberikan mulai dari posisi saat ini. |
| \$unwind | Ini digunakan untuk melepas dokumen yang menggunakan array. Saat menggunakan larik, data yang sudah digabungkan sebelumnya dengan menjalankan operasi ini akan dibatalkan, sehingga dokumen akan kembali ke dokumen individual lagi. |

| | |
|--|--|
| | Maka dari itu dengan perintah ini kita akan menambah jumlah dokumen untuk tahap selanjutnya. |
|--|--|

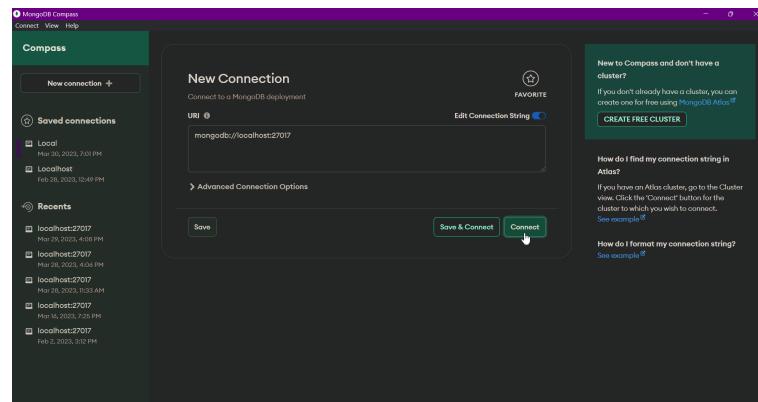
6.4. Software

1. MongoDB
2. Compass
3. MongoDB Shell

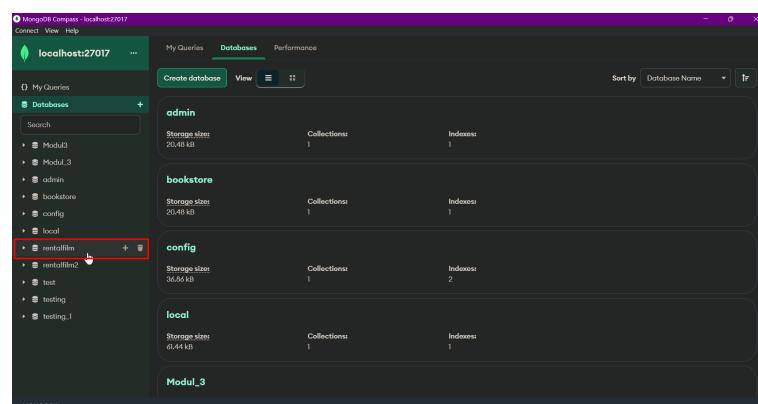
6.5. Tahapan Kerja

6.4.1. Menjalankan MongoDB di Compass dan Command Prompt

1. Buka Aplikasi MongoDB Compass kemudian klik “Connect”



2. Untuk melanjutkan ketahapan kerja selanjutnya pastikan database “Rentalfilm” sudah ada di MongoDB.



3. Buka MongoShell dengan melakukan perintah “mongosh” di dalam command prompt.

| |
|-----------------------|
| C:\Users\Deny>mongosh |
|-----------------------|

4. Lalu ganti ke database rental film dengan menggunakan perintah “use”.

| |
|---|
| <pre>test> use rentalfilm switched to db rentalfilm rentalfilm></pre> |
|---|

6.4.2. Update dan save document

1. Update

```
rentalfilm> db.Film.update({_id: 1}, {$set : {Release_Year : 2003}});
```

```
rentalfilm> db.Film.update({_id: 1}, {$set : {Release_Year : 2003}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
rentalfilm> |
```

6.4.3. Remove Document

1. Remove

```
rentalfilm> db.Film.remove({_id: 1})
```

```
rentalfilm> db.Film.remove({_id : 1})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 1 }
```

2. Remove One

```
rentalfilm> db.Film.deleteOne({Rating: 9})
```

```
rentalfilm> db.Film.deleteOne({Rating: 9})
{ acknowledged: true, deletedCount: 1 }
```

3. Remove All

```
rentalfilm> db.Film.deleteMany({Length: 90})
```

```
rentalfilm> db.Film.deleteMany({Length: 90})
{ acknowledged: true, deletedCount: 2 }
```

6.4.4. Limit Document

1. Limit

```
rentalfilm> db.Customer.find().limit(2)
```

```
rentalfilm> db.Customer.find().limit(2)
[
  {
    _id: 1,
    AddressId: 1,
    AddressColumn: 2,
    FirstName: 'Denny',
    LastName: 'Partala',
    Email: 'email@gmail.com',
    Active: 'Y',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  },
  {
    _id: 2,
    AddressId: 2,
    AddressColumn: 2,
    FirstName: 'Otoso',
    LastName: 'Bearu',
    Email: 'email2@gmail.com',
    Active: 'N',
    CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
    LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
  }
]
```

2. Skip

```
rentalfilm> db.Customer.find().limit(2).skip(2)
```

```
rentalfilm> db.Customer.find().limit(2).skip(2)
[ {
  _id: 3,
  AddressId: 3,
  AddressColumn: 3,
  FirstName: 'Bjorn',
  LastName: 'Mato',
  Email: 'email3@gmail.com',
  Active: 'N',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
{
  _id: 4,
  AddressId: 2,
  AddressColumn: 2,
  FirstName: 'Mitch',
  LastName: 'Benjamin',
  Email: 'email4@gmail.com',
  Active: 'T',
  CreateDate: '2023-03-15T10:22:52.525+00:00',
  LastUpdate: '2023-03-15T10:22:52.525+00:00'
}
```

6.4.5. Index

1. ensureIndex()

```
rentalfilm> db.country.createIndex({"country" : 1})
```

```
rentalfilm> db.country.createIndex({"country" : 1})
country_1
```

2. unique

```
rentalfilm> db.country.createIndex({"country" : 1}, {unique : true})
```

```
rentalfilm> db.country.createIndex({"country" : 1}, {unique : true})
country_1
```

3. sparse

```
rentalfilm> db.Film.createIndex({"LanguageID" : 1}, {sparse : true})
```

```
rentalfilm> db.Film.createIndex({"LanguageID" : 1}, {sparse: true})
LanguageID_1
```

4. expireAfterSeconds

```
rentalfilm> db.city.createIndex({"last_update" : 1}, {expireAfterSeconds : 60})
```

```
rentalfilm> db.city.createIndex({"last_update" : 1 }, {expireAfterSeconds: 60})
last_update_1
```

5. v

```
rentalfilm> db.Customer.createIndex({"AddressId" : 1}, {v : 1})
```

```
rentalfilm> db.Customer.createIndex({"AddressId" : 1}, {v : 1})
AddressId_1
```

6. weights

```
rentalfilm> db.Film.createIndex({Title : "text"}, {weights: {Title : 5}})

rentalfilm> db.Film.createIndex({Title : "text"}, {weights: {Title : 5}})
Title_text
rentalfilm> |
```

7. default_language

```
rentalfilm> db.Customer.createIndex({Email : "text"}, {default_language:
"english"})

rentalfilm> db.Customer.createIndex({Email : "text"}, {default_language: "english"})
Email_text
rentalfilm>
```

8. language_override

```
rentalfilm> db.Customer.createIndex({FirstName : "text"}, {language_override:
"spanish"})

rentalfilm> db.Customer.createIndex({FirstName : "text"}, {language_override: "spanish"})
FirstName_text
rentalfilm>
```

6.4.6. Aggregation

1. \$sum

```
rentalfilm> db.Film.aggregate([ {$group: { _id: null, Total_Replacement_Cost:
{ $sum: '$Replacement_Cost' } } }]);

rentalfilm> db.Film.aggregate([ { $group: { _id: null, Total_Replacement_Cost: { $sum: '$Replacement_Cost' } } } ]);
[ { _id: null, Total_Replacement_Cost: 6000 }
rentalfilm> |
```

2. \$avg

```
rentalfilm> db.Film.aggregate([ {$group: { _id: null, Total_Replacement_Cost:
{ $avg: '$Replacement_Cost' } } }]);

rentalfilm> db.Film.aggregate([ { $group: { _id: null, Total_Replacement_Cost: { $avg: '$Replacement_Cost' } } } ]);
[ { _id: null, Total_Replacement_Cost: 1160 }
rentalfilm> |
```

3. \$min

```
rentalfilm> db.Film.aggregate([ {$group: { _id: null, Total_Replacement_Cost:
{ $min: '$Replacement_Cost' } } }]);

rentalfilm> db.Film.aggregate([ { $group: { _id: null, Total_Replacement_Cost: { $min: '$Replacement_Cost' } } } ]);
[ { _id: null, Total_Replacement_Cost: 1000 }
rentalfilm> |
```

4. \$max

```
rentalfilm> db.Film.aggregate([ {$group: { _id: null, Total_Replacement_Cost:
{ $max: '$Replacement_Cost' } } }]);

rentalfilm> db.Film.aggregate([ { $group: { _id: null, Total_Replacement_Cost: { $max: '$Replacement_Cost' } } } ]);
[ { _id: null, Total_Replacement_Cost: 1200 }
rentalfilm> |
```

5. \$push

```
rentalfilm> db.Elem.updateOne({_id : 1}, {$push : {results: 100}})
```

```

testing> db.Elem.updateOne({_id : 1}, {$push : {results: 100}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
testing>

```

6. *\$addToSet*

```
rentalfilm> db.Elem.updateOne({_id : 1}, {$addToSet : {results: 160}})
```

```

testing> db.Elem.updateOne({_id : 1 }, {$addToSet : { results : 160}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
testing>

```

7. *\$first*

```

rentalfilm> db.Customer.aggregate([
  {
    $group: {
      _id: null,
      first: {
        $first: '$$ROOT'
      }
    }
  },
  [
    {
      _id: null,
      first: {
        _id: 1,
        AddressId: 1,
        AddressColumn: 2,
        FirstName: 'Denny',
        LastName: 'Partala',
        Email: 'email@gmail.com',
        Active: 'Y',
        CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
        LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
      }
    }
  ]
])

```

'\$\$ROOT' disini mereferensikan ke dokumen yang ada didalam collection itu sendiri.

8. *\$last*

```
rentalfilm> db.Customer.aggregate([{$group: {_id: null, first:{ $last: '$$ROOT' } } } ]);
```

```

rentalfilm> db.Customer.aggregate([ { $group: { _id: null, first: { $last: '$$ROOT' } } }]);
[
  {
    _id: null,
    first: {
      _id: 4,
      AddressId: 2,
      AddressColumn: 2,
      FirstName: 'Mitch',
      LastName: 'Benjamin',
      Email: 'email4@gmail.com',
      Active: 'Y',
      CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
      LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
    }
  }
]
rentalfilm>

```

6.4.7. Pipeline

1. \$project

Digunakan untuk meneruskan document dengan field yang diminta ke tahapan pipeline selanjutnya.

```
rentalfilm> db.Film.aggregate([
  {
    $project: {
      "Last_Update" : 0
    }
  }
]
[
  {
    _id: 2,
    Title: 'Country Bear',
    Description: '-',
    Release_Year: 2023,
    Rental_Duration: 1608,
    Rental_Rate: 10.2,
    Length: 90,
    Replacement_Cost: 1200,
    Rating: 9,
    Special_Features: 'Bears',
    Fulltext: 'Bear Bear Bear 2'
  },
]
```

```
rentalfilm> db.Film.aggregate([
  ...
  {
    ...
    $project: {
      ...
      "Last_Update" : 0
      ...
    }
  ...
  ]
]
[
  {
    _id: 2,
    Title: 'Country Bear',
    Description: '_',
    Release_Year: 2023,
    Rental_Duration: 1608,
    Rental_Rate: 10.2,
    Length: 90,
    Replacement_Cost: 1200,
    Rating: 9,
    Special_Features: 'Bears',
    Fulltext: 'Bear Bear Bear 2'
  },
  {
    ...
  }
]
```

Contoh diatas menunjukan penggunaan \$project untuk tidak menampilkan field yang tidak kita perlukan.

2. \$match

Melakukan filter dokumen sesuai kondisi yang ditentukan dan diteruskan ke tahap pipeline selanjutnya.

```
rentalfilm> db.Film.aggregate([{$group: {"Last_Update" : 0 } }, { $match: {
  Title: "Country Bear" } }]);
```

```

rentalfilm> db.Film.aggregate([ { $project: { "Last_Update": 0 } }, { $match: { Title: "Country Bear" } }]);
[ {
    _id: 2,
    Title: 'Country Bear',
    Description: '-',
    Release_Year: 2023,
    Rental_Duration: 1608,
    Rental_Rate: 10.2,
    Length: 99,
    Replacement_Cost: 1200,
    Rating: 9,
    Special_Features: 'Bears',
    Fulltext: 'Bear Bear Bear 2'
},
{
    _id: 1,
    LanguageID: 1,
    Title: 'Country Bear',
    Description: '-',
    Release_Year: 2023,
    Rental_Duration: 1608,
    Rental_Rate: 10.2,
    Length: 99,
    Replacement_Cost: 1200,
    Rating: 9,
    Special_Features: 'Bears',
    Fulltext: 'Bear Bear Bear 2'
}
]
rentalfilm>

```

Contohnya diatas dilakukan \$project pada tahap pertama dimana field last_update dihilangkan kemudian diteruskan ketahap kedua dimana \$match akan mencari dokumen yang memiliki field ‘Title’ dengan value ‘Country Bear’.

3. \$group

```

rentalfilm> db.Customer.aggregate([
    {
        $group: {
            _id: '$Active',
        }
    }
]);
[ { _id: 'Y' }, { _id: 'T' }, { _id: 'N' }]

```

\$Group digunakan untuk mengelompokan field dengan value unik sesuai yang telah ditentukan. \$group juga bisa digunakan dengan operator yang lain seperti \$last, \$avg dll.

4. \$sort

```

rentalfilm> db.Customer.aggregate([
    {
        $sort: {
            LastName: 1
        }
    }
]);

```

```

rentalfilm> db.Customer.aggregate([
...   {
...     $sort: {
...       LastName: 1
...     }
...   },
... ]
[ {
  _id: 2,
  AddressId: 2,
  AddressColumn: 2,
  FirstName: 'Otoso',
  LastName: 'Bearu',
  Email: 'email2@gmail.com',
  Active: 'N',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:50.525Z")
},
{
  _id: 4,
  AddressId: 2,
  AddressColumn: 2,
  FirstName: 'Mitch',
  LastName: 'Benjamin',
  Email: 'email4@gmail.com',
  Active: 'T',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
]

```

\$sort digunakan untuk melakukan pengurutan data pada field yang diberikan, dan disini bisa digunakan juga weight yang telah ditentukan pada index apabila sudah dibuat.

5. *\$limit*

```

rentalfilm> db.Customer.aggregate([
  {
    $Limit: 2
  }
])

```

```

rentalfilm> db.Customer.aggregate([
...   {
...     $limit: 2
...   }
... ])
[ {
  _id: 1,
  AddressId: 1,
  AddressColumn: 2,
  FirstName: 'Denny',
  LastName: 'Partala',
  Email: 'email@gmail.com',
  Active: 'Y',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:52.525Z")
},
{
  _id: 2,
  AddressId: 2,
  AddressColumn: 2,
  FirstName: 'Otoso',
  LastName: 'Bearu',
  Email: 'email2@gmail.com',
  Active: 'N',
  CreateDate: ISODate("2023-03-15T10:22:52.525Z"),
  LastUpdate: ISODate("2023-03-15T10:22:50.525Z")
}
]
rentalfilm> |

```

Digunakan untuk membatasi berapa banyak dokumen yang akan diteruskan ketahapan pipeline berikutnya.

6. *\$unwind*

```

rentalfilm> db.TipeDataValidasi.aggregate([
  {
    $unwind: {
      Path: 'Tag',
    }
  }
]);

```

```

Modul3> db.TipeDataValidasi.aggregate([
...   {
...     $unwind: {
...       path: '$Tag',
...     }
...   }
... ]);
[ {
  {
    _id: ObjectId("64227bf92ec0c7264609f7c8"),
    Nama_Produk: 'Mesin Cuci',
    Harga: 10000000,
    Berat_Produk: 10.9,
    Rusak: false,
    Tanggal_Masuk: ISODate("2023-03-28T05:32:41.054Z"),
    Tag: 'Elektronik',
    Dimensi: { tinggi: 60, panjang: 50, lebar: 20 },
    Id_Produk: ObjectId("64227bf92ec0c7264609f7c7")
  },
  {
    _id: ObjectId("64227bf92ec0c7264609f7c8"),
    Nama_Produk: 'Mesin Cuci',
    Harga: 10000000,
    Berat_Produk: 10.9,
    Rusak: false,
    Tanggal_Masuk: ISODate("2023-03-28T05:32:41.054Z"),
    Tag: 'Perabotan',
    Dimensi: { tinggi: 60, panjang: 50, lebar: 20 },
    Id_Produk: ObjectId("64227bf92ec0c7264609f7c7")
  }
]
Modul3>

```

Digunakan ketika ingin dekonstruksi array menjadi field nya masing masing.

6.5. Tugas dan Latihan

1. Buatlah dan jalankan query untuk membuat setiap jenis tipe index pada database rentalfilm yang berbeda dengan contoh pada tahapan kerja.
2. Buatlah dan jalankan setiap query aggregate pada database rentalfilm yang berbeda dengan contoh yang telah diberikan.
3. Buatlah dan jalankan setiap query pipeline pada database rentalfilm yang berbeda dengan contoh yang telah diberikan.
4. Buatlah dan jalankan minimal 3 buah gabungan query pipeline pada database rentalfilm.

MODUL 7

MAPREDUCE & TRANSACTION

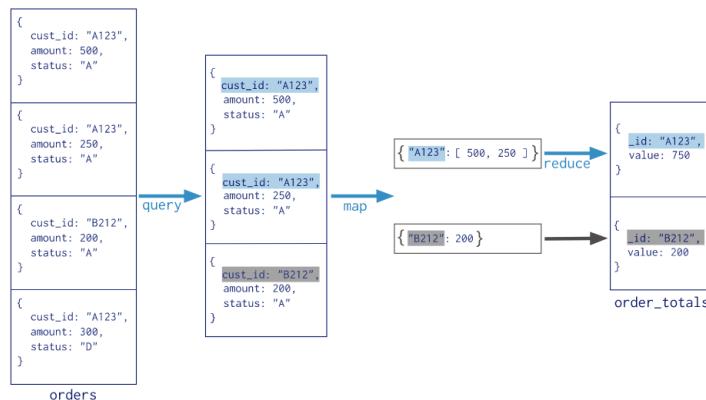
7.1. Tujuan Praktikum

1. Mahasiswa mampu melakukan setup cluster mongodb
2. Mahasiswa mampu melakukan replikasi db
3. Mahasiswa mampu implementasi transaksi
4. Mahasiswa mengenal tools mongodb

7.2. Dasar Teori

7.2.1. Map Reduce

Merupakan data proses paradigma data pada MongoDB yang bertujuan untuk menyatukan atau memadatkan data bervolume banyak menjadi hasil agregat yang berguna.



Gambar 20. Gambar Map Reducing

Contohnya pada gambar diatas, dari data pada koleksi order dilakukan perintah map reducing dimana data tersebut di satukan berdasarkan data “cust_id” sehingga menjadi “order_totals”.

Map Reduce dibagi menjadi 2 function. Function pertama adalah Map dimana semua data akan dikelompokan berdasarkan key-value dan kemudian Reduce digunakan untuk melakukan operasi pada data yang tadi sudah di Mapping.

Pada versi 6.0 map-reduce sudah tidak ada, fungsi ini digantikan pada aggregate pipeline yaitu gabungin \$group dan \$out.

```
db.orders.aggregate([  
  { $group: { _id: "$cust_id", value: { $sum: "$price" } } },  
  { $out: "order_totals" }  
])
```

7.2.2. Transactions

Transaksi memainkan peran penting dalam menjaga konsistensi dan integritas data dalam database. Transaksi mewakili satu unit kerja yang terdiri dari beberapa operasi yang dijalankan dalam satu urutan. Pada bagian ini, kita akan membahas konsep transaksi di MongoDB, penggunaannya, dan bagaimana transaksi membantu menyelesaikan berbagai operasi.

7.2.3. Gambaran umum transaksi

MongoDB mendukung transaksi multi-dokumen, sehingga Anda dapat melakukan beberapa operasi baca dan tulis di beberapa dokumen dalam satu transaksi atomik. Sebuah transaksi dapat melibatkan beberapa operasi, misalnya:

- Membuat dokumen baru
- Memperbarui dokumen yang sudah ada
- Menghapus dokumen
- Membaca dokumen

Tujuan mendasar dari sebuah transaksi adalah untuk menjalankan semua atau tidak sama sekali dari operasi-operasinya. Ini berarti, jika ada operasi dalam transaksi yang gagal, seluruh transaksi akan dibatalkan, dan basis data akan kembali ke kondisi awal, sehingga memastikan konsistensi data.

Transaksi di MongoDB sangat penting untuk mencapai properti ACID berikut ini:

- **Atomicity:** Memastikan bahwa semua operasi dalam transaksi dieksekusi, atau tidak ada yang dieksekusi.
- **Consistency:** Menjamin bahwa, setelah menyelesaikan transaksi, basis data tetap dalam keadaan konsisten.
- **Isolation:** Menjamin bahwa operasi dalam transaksi terisolasi dari transaksi lain yang sedang dieksekusi secara bersamaan.
- **Durability:** Menjamin bahwa setelah transaksi berhasil diselesaikan, efeknya akan disimpan secara permanen di dalam database.

7.2.4. Penggunaan transaksi

Untuk memulai transaksi di MongoDB, Anda harus mendapatkan sesi dan kemudian memulai transaksi menggunakan metode `startTransaction()`. Setelah melakukan operasi yang diperlukan, Anda dapat meng-commit transaksi untuk menerapkan perubahan pada basis data, atau membatalkannya untuk membuang perubahan.

Berikut ini contoh ilustrasi penggunaan transaksi:

```
// Create New client and connect to server
client = MongoClient(uri, server_api=ServerApi('1'))
wc_majority = WriteConcern("majority", wtimeout =1000)

// Define callback for database operations
def callback(session):
    local = session.client.Mahasiswa.DataDiri

    local.insert_one({"NPM":"192310004","nama" : "denny dolok 2", "alamat" :
"bogor"}, session = session)

// start session on this session, executes callback once and then commit
transaction
with client.start_session() as session:
    session.with_transaction(
        callback,
        read_concern=ReadConcern("local"),
        write_concern=wc_majority,
        read_preference=ReadPreference.PRIMARY,
    )
```

7.2.4. Keterbatasan transaksi

Meskipun transaksi memberikan manfaat yang sangat besar dalam hal konsistensi dan integritas data, sangat penting untuk mengetahui beberapa keterbatasannya:

- Transaksi hanya tersedia di MongoDB versi 4.0 ke atas.
- Transaksi dapat menyebabkan overhead kinerja, terutama untuk beban kerja yang banyak menulis.
- Di dalam cluster MongoDB, transaksi hanya mendukung durasi maksimum 60 detik.

Singkatnya, transaksi adalah fitur yang kuat dari MongoDB, memastikan integritas data, dan konsistensi dalam database. Dengan memahami penggunaan dan implikasinya, Anda dapat menggunakannya secara efektif dalam aplikasi Anda sesuai dengan kebutuhan spesifik Anda.

7.3. Software

1. MongoDB
2. Atlas

7.4. Tahapan Kerja

7.4.1. MapReduce

1. Buka Command Prompt dan ketik `mongosh` untuk masuk kedalam terminal monogdb.

```
C:\Users\Denny>mongosh
Current Mongosh Log ID: 64631947e380b9d765d0482a
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.7.1
Using MongoDB: 6.0.4
Using Mongosh: 1.7.1
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
-----
The server generated these startup warnings when booting
2023-05-16T06:57:32.139+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

test> |
```

2. Ganti ke database rentalfilm dengan menggunakan perintah use.

```
test> use rentalfilm
switched to db rentalfilm
rentalfilm>
```

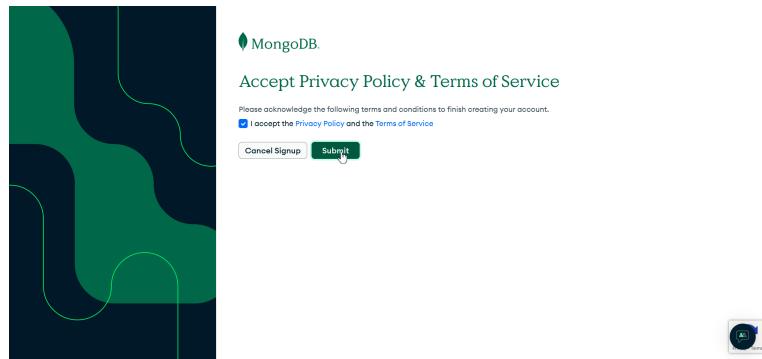
3. Pada versi 6.0 map reduce sudah tidak bisa digunakan sehingga diganti dengan aggregate gabungan antara \$group dan \$out, dengan contoh sebagai berikut.

```
rentalfilm> db.Film.aggregate([
  { $group: {
    _id: "$Length"
    total_rating: {
      $avg : "$Rating"
    }
  }},
  { $out: 'Rating-Based-On-Length'
  }
])
```

Pada aggregate diatas pertama dilakukan pengelompokan berdasarkan field Length dari collection Film, yang kemudian dilakukan perhitungan rata-rata Rating berdasarkan kelompok data tersebut. Dan terakhir akan membuat collection baru dengan '\$out' dari data yang telah diproses tadi.

7.4.2. Setup akun MongoDb atlas dan membuat cluster

1. Buka halaman registrasi MongoDB Atlas dengan membuka link <https://account.mongodb.com/account/register>.
2. Buat akun baru dengan menggunakan metode apa saja.
3. Apabila diminta untuk menyetujui Policy dan Terms of Service, centang checkboxnya lalu tekan Submit.



4. Kemudian jika muncul form singkat tentang tujuan menggunakan MongoDB isi seperti ini.

What is your goal today?
Your answer will help us guide you to successfully getting started with MongoDB Atlas.

- Build a new application
- Learn MongoDB
- Explore what I can build
- Migrate an existing application

What type of application are you building?

I'm just exploring

What is your preferred language?
We'll use this to customize code samples and content we share with you. You can always change this later.

Python

5. Pada halaman utama dashboard MongoDB atlas tekan tombol ‘Build a Database’.

Project 0 Data Services App Services Charts

192310004'S ORO - 2023-06-16 > PROJECT 0

Database Deployments

Create a database

Choose your cloud provider, region, and specs.

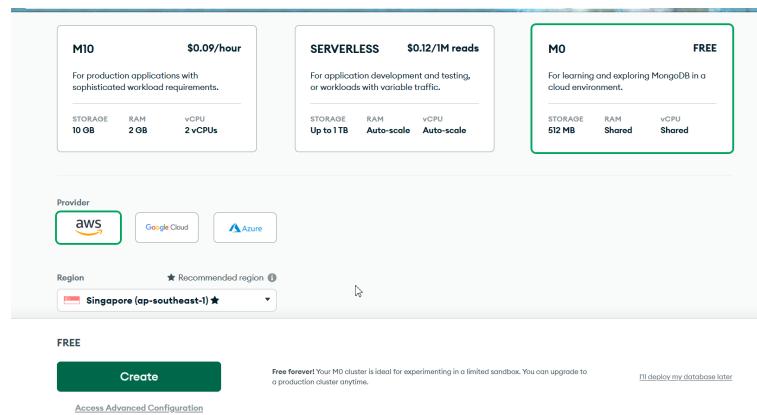
Build a database

Once your database is up and running, live migrate an existing MongoDB database into Atlas with our [Live Migration Service](#).

New On Atlas

<https://cloud.mongodb.com/v2/646325a1b55bf159cc607e74/clusters/starterTemplates>

6. Pilih tipe yang free dan ganti region menjadi Singapore. Lalu tekan tombol Create.



7. Dihalaman Selanjutnya ketik username dan password yang akan digunakan ketika ingin mengakses database mongoDB, lalu tekan tombol “Create User”

Create a database user using a username and password. Users will be given the *read and write* to any database **privilege** by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Username
Dennyl6

Password ⓘ
Bearuang6 Autogenerate Secure Password Copy

Create User

8. Scroll kebawah dan tekan tombol “Add My Current IP Address”. Ini akan memasukan IP yang saat ini digunakan kedalam whitelist sehingga hanya lewat IP yang ditambahkan saja MongoDB bisa diakses.

2 Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.

My Local Environment
Use this to add network IP addresses to the IP Access List. This can be modified at any time.

Cloud Environment
Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

Add entries to your IP Access List

Only an IP address you add to your Access List will be able to connect to your project's clusters.

| IP Address | Description |
|----------------------------------|-------------------|
| Enter IP Address | Enter description |
| Add My Current IP Address | |
| Add Entry | |

9. Jika sudah tekan tombol Finish and Close.
10. MongoDB atlas sudah dibuat dengan satu buah Cluster dengan 3 buah node replica set.

7.4.3. Mengenal Python

1. Install Python ke komputer anda dengan mengunduh terlebih dahulu di <https://www.python.org/downloads/>.
2. Download Python Installer dan mulai proses installasi.

7.4.4. Koneksi Python dengan mongodb

1. Buka Command Prompt dan Jalankan perintah “py -m pip install pymongo”

```
D:\MONGO>py -m pip install pymongo
Collecting pymongo
  Downloading pymongo-4.3.3-cp311-cp311-win_amd64.whl (382 kB)
    382.5/382.5 kB 1.6 MB/s eta 0:00:00
Collecting dnspython<3.0.0,>=1.16.0
  Using cached dnspython-2.3.0-py3-none-any.whl (283 kB)
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.3.0 pymongo-4.3.3

[notice] A new release of pip available: 22.3.1 &gt; 23.1.2
[notice] To update, run: C:\Users\Deny\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
```

2. Buka Visual Studio Code dan buat file baru dengan nama MongoDB.py.
3. Ketik code berikut

```
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
uri =
"mongodb+srv://<Username>:<Password>@cluster0.mi4imwt.mongodb.net/?retryWrites
=true&w=majority"
# Create a new client and connect to the server
client = MongoClient(uri, server_api=ServerApi('1'))
# Send a ping to confirm a successful connection

try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
except Exception as e:
    print(e)
```

Ganti <Username> dan <Password> sesuai dengan yang telah dimasukan ketika membuat database di MongoDB Atlas.

4. Jalankan script tersebut dengan melakukan perintah “py MongoDB.py” pada terminal di Visual Studio Code

```
D:\MONGO>py MongoDB.py
```

```
D:\MONGO>py MongoDB.py
Pinged your deployment. You successfully connected to MongoDB!
```

Python sudah berhasil terhubung dengan Database mongoDB yang dibuat jika pesan diatas muncul.

7.4.5. Transaction mongodb

1. Transaction pada MongoDB menggunakan Pymongo bisa dilakukan dengan menggunakan script Berikut.

```

from pymongo.mongo_client import MongoClient
from pymongo.read_concern import ReadConcern
from pymongo.read_preferences import ReadPreference
from pymongo.write_concern import WriteConcern
from pymongo.server_api import ServerApi
uri =
"mongodb+srv://<Username>:<Password>@cluster0.mi4imwt.mongodb.net/?retryWrites=true&w=majority"
# Create a new client and connect to the server
client = MongoClient(uri, server_api=ServerApi('1'))
wc_majority = WriteConcern("majority", wtimeout = 1000)

def callback(session):
    local = session.client.Mahasiswa.DataDiri

    local.insert_one({"NPM": "192310004", "nama" : "denny dolok 2", "alamat" :
    "bogor"}, session = session)

    with client.start_session() as session:
        session.with_transaction(
            callback,
            read_concern=ReadConcern("local"),
            write_concern=wc_majority,
            read_preference=ReadPreference.PRIMARY,
        )
)

```

Pada def callback(session) dimasukkan semua operasi yang akan dieksekusi oleh MongoDB yang kemudian akan di commit pada saat session.with_transactions.

2. Apabila proses transaksi berhasil maka data akan masuk kedalam database MongoDB di Atlas.

| _id | | NPM | nama | alamat |
|--|--|-------------------------------|------------------------------------|------------------------------|
| <code>_id: ObjectId('6463661661a87d087712d441')</code> | | <code>NPM: "192310004"</code> | <code>nama: "Denny Dolok 2"</code> | <code>alamat: "bogor"</code> |

7.5. Tugas dan Latihan

1. Buat dan jalankan proses transaksi untuk memasukan data diri mahasiswa minimal 5 buah data kedalam database MongoDB Atlas menggunakan pymongo.
2. Buat dan jalankan query aggregate untuk melakukan map reduce pada sebuah collection.

MODUL 8

SECURITY MONGODB

8.1. Tujuan Praktikum

1. Mahasiswa mampu memahami security pada MongoDB
2. Mahasiswa mampu mengimplementasikan Authentication pada MongoDB
3. Mahasiswa mampu mengimplementasikan Authorization pada MongoDB

8.2. Dasar Teori

8.2.1. Security MongoDB

MongoDB menyediakan berbagai fitur, seperti autentikasi, kontrol akses, enkripsi, untuk mengamankan penerapan MongoDB. Beberapa fitur keamanan utama meliputi:

1. Authentication
2. Authorization
3. TLS/SSL

8.2.2. Authentication

Authentication adalah proses memverifikasi identitas klien. Ketika kontrol akses (otorisasi) diaktifkan, MongoDB mengharuskan semua klien untuk mengautentikasi diri mereka sendiri untuk menentukan akses mereka.

Meskipun otentikasi dan otorisasi berhubungan erat, otentikasi berbeda dengan otorisasi:

1. Otentikasi memverifikasi identitas pengguna.
2. Otorisasi menentukan akses pengguna terverifikasi ke sumber daya dan operasi.

MongoDB punya beberapa mekanisme Authentication, diantaranya:

1. SCRAM Authentication

Mekanisme Autentikasi Salted Challenge Response Authentication Mechanism (SCRAM) adalah mekanisme autentikasi default untuk MongoDB.

2. x.509 Certificate Authentication

MongoDB mendukung autentikasi sertifikat x.509 untuk autentikasi klien dan autentikasi internal anggota set replika dan cluster yang dipecah. Autentikasi sertifikat x.509 memerlukan koneksi TLS/SSL yang aman.

Untuk menggunakan MongoDB dengan x.509, diwajibkan untuk menggunakan sertifikat valid yang dibuat dan ditandatangani oleh otoritas sertifikat. Sertifikat x.509 klien harus memenuhi persyaratan sertifikat klien.

3. Kerberos Authentication

MongoDB Enterprise mendukung Autentikasi Kerberos. Kerberos adalah protokol otentikasi standar industri untuk sistem klien/server besar yang menyediakan otentikasi menggunakan token berumur pendek yang disebut tiket.

Untuk menggunakan MongoDB dengan Kerberos, harus memiliki penerapan Kerberos yang dikonfigurasi dengan benar, prinsipal layanan Kerberos yang dikonfigurasi untuk MongoDB, dan prinsipal pengguna Kerberos yang ditambahkan ke MongoDB.

4. LDAP Proxy Authentication

MongoDB Enterprise dan MongoDB Atlas mendukung autentikasi *proxy* LDAP Proxy Authentication melalui layanan *Lightweight Directory Access Protocol* (LDAP).

5. Internal / Membership Authentication

Selain memverifikasi identitas klien, MongoDB dapat meminta anggota set replika dan kluster terpecah untuk mengautentikasi keanggotaan mereka pada set replika atau kluster terpecah masing-masing. Lihat Otentikasi Internal/Keanggotaan untuk informasi lebih lanjut.

8.2.3. Authorization

Authorization pada MongoDB adalah mekanisme yang digunakan untuk mengontrol akses pengguna ke database dan koleksi data tertentu dalam lingkungan MongoDB. Dengan menggunakan authorization, administrator database dapat menentukan hak akses dan peran untuk setiap pengguna atau peran yang terkait dengan basis data.

MongoDB menyediakan beberapa metode untuk melakukan authorization:

1. *Built-in Role-based Access Control (RBAC)*: MongoDB memiliki beberapa peran bawaan yang dapat diberikan kepada pengguna, seperti read, readWrite, dbAdmin, dbOwner, dll. Setiap peran memiliki hak akses yang ditentukan terhadap basis data dan koleksi yang berbeda.

| Roles | Hak |
|-------|----------------------------|
| Read | Membaca data pada database |

| | |
|----------------------|---|
| Write | Memiliki semua hak dari Role Read dan memiliki hak untuk menambah dan mengubah data. |
| dbAdmin | Memiliki hak untuk melakukan perintah administrative seperti indexing, perintah yang berhubungan dengan schema. |
| userAdmin | Memiliki hak untuk membuat dan memodifikasi roles dan user pada database yang ditentukan. |
| dbOwner | Memiliki hak gabungin dari Role ReadWrite, userAdmin, dbAdmin. |
| readAnyDatabase | Memiliki hak yang sama dengan Role Read namun berlaku pada semua database keculai local dan config. |
| writeAnyDatabase | Memiliki hak yang sama dengan Role Write namun berlaku pada semua database keculai local dan config. |
| userAdminAnyDatabase | Memiliki hak yang sama dengan Role userAdmin namun berlaku pada semua database keculai local dan config. |
| dbAdminAnyDatabase | Memiliki hak yang sama dengan Role dbAdmin namun berlaku pada semua database keculai local dan config |
| Root | Memiliki hak akses dari semua Role. |

2. *Custom Roles*: Selain peran bawaan, administrator database juga dapat membuat peran kustom dengan hak akses yang disesuaikan sesuai dengan kebutuhan aplikasi.
3. *Authentication Mechanisms*: Sebelum melakukan authorization, MongoDB memerlukan proses autentikasi pengguna. MongoDB mendukung beberapa mekanisme autentikasi, seperti username/password, Kerberos, LDAP, dan X.509 certificate authentication.
4. *Access Control Lists (ACLs)*: Administrator database dapat mengatur kontrol akses lebih lanjut menggunakan daftar kendali akses (ACLs) yang memungkinkan pengaturan akses yang lebih rinci pada level koleksi dan dokumen individu.

Dengan menggunakan authorization, administrator database dapat membatasi akses pengguna hanya ke data yang relevan, menjaga keamanan dan integritas data, serta melindungi basis data dari akses yang tidak sah. Penting untuk mengatur authorization dengan hati-hati dan memberikan hak akses yang sesuai kepada pengguna sesuai dengan tanggung jawab dan kebutuhan aplikasi.

8.2.4. *TLS/SSL (Transport Encryption)*

MongoDB mendukung TLS/SSL (Transport Layer Security/Secure Sockets Layer) untuk mengenkripsi semua lalu lintas jaringan MongoDB. TLS/SSL memastikan bahwa lalu lintas jaringan MongoDB hanya dapat dibaca oleh klien yang dituju. MongoDB menonaktifkan dukungan untuk enkripsi TLS 1.0 pada sistem yang mendukung TLS 1.1+. Untuk detail lebih lanjut, lihat Nonaktifkan TLS 1.0.

MongoDB menggunakan pustaka native TLS/SSL OS libraries:

| Platform | TLS/SSL Library |
|-----------|---------------------------|
| Windows | Secure Channel (Schannel) |
| Linux/BSD | OpenSSL |
| macOS | Secure Transport |

Gambar 21. Pustaka native TLS/SSL OS

TLS/SSL Ciphers. Enkripsi TLS/SSL MongoDB hanya mengizinkan penggunaan cipher TLS/SSL yang kuat dengan panjang kunci minimal 128-bit untuk semua koneksi.

Certificates. Untuk menggunakan TLS/SSL dengan MongoDB, diwajibkan untuk memiliki sertifikat TLS/SSL sebagai file PEM, yang merupakan wadah sertifikat yang digabungkan. MongoDB dapat menggunakan sertifikat TLS/SSL yang valid yang diterbitkan

oleh otoritas sertifikat atau sertifikat yang ditandatangani sendiri. Untuk penggunaan produksi, penerapan MongoDB harus menggunakan sertifikat valid yang dibuat dan ditandatangani oleh otoritas sertifikat yang sama. Dapat juga membuat dan mengelola otoritas sertifikat independen, atau menggunakan sertifikat yang dibuat oleh vendor TLS/SSL pihak ketiga. Menggunakan sertifikat yang ditandatangani oleh otoritas sertifikat terpercaya memungkinkan driver MongoDB memverifikasi identitas server.

8.3. Software

1. MongoDB
2. Mongo shell

8.4. Tahapan Kerja

8.4.1. Authentication MongoDB

1. Memulai MongoDB tanpa autentikasi
2. Hubungkan ke server menggunakan perintah mongosh
3. Membuat administrator pengguna

```
> use admin
```

Tahap pertama adalah untuk membuat user dengan role **root**, yang memberikan hak istimewa semua role menjadi satu. Contoh berikut ini akan membuat pengguna mongo dengan kata sandi "**mongo**":

```
db.createUser(  
  {  
    user: "mongo",  
    pwd: "mongo",  
    roles: [  
      "root",  
    ]  
  }  
)
```

Kemudian putuskan sambungan dari shell mongo (**Ctrl+D**).

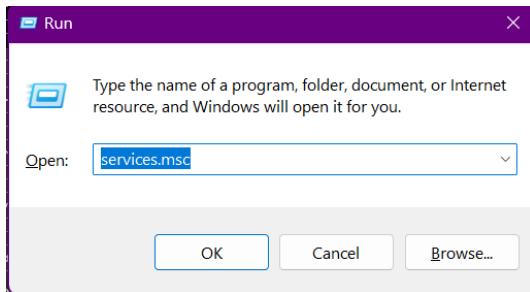
4. Mengaktifkan autentikasi dalam berkas konfigurasi mongod.

Buka “C:\Program Files\MongoDB\Server\6.0\bin\mongod.cfg” dengan kode editor dan tambahkan baris berikut:

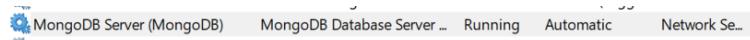
```
Security:  
  Authorization: "enabled"
```

```
security:  
  authorization: "enabled"
```

Setelah selesai buka Microsoft service dengan menekan tombol Start + R lalu ketik “services.msc”.



Kemudian cari service dengan nama “MongoDB Server”.



Klik kanan pada service tersebut lalu pilih “restart”.

Mulai sekarang, semua klien yang terhubung ke server ini harus mengautentikasi diri mereka sendiri sebagai pengguna yang valid, dan mereka hanya akan dapat melakukan tindakan sebagaimana ditentukan oleh peran yang ditetapkan.

5. Login sebagai root bisa dilakukan dengan menggunakan perintah mongosh namun menambahkan beberapa argument menjadi : “mongosh --username root --password root.”

```
C:\Users\Denny->mongosh --username mongo --password mongo
```

6. Apabila kita masuk ke terminal mongosh tanpa username dan password kita tidak akan bisa menjalankan perintah untuk berinteraksi dengan database dikarenakan kita tidak memiliki Authentication.

```
Test> db.test.insertOne({})
```

```
test> db.test.insertOne({})
MongoServerError: command insert requires authentication
```

8.4.2. Authorization MongoDB

Authorization merupakan proses yang dilakukan setelah Authentication berhasil, proses ini dilakukan untuk memeriksa Apakah user memiliki hak akses untuk melakukan sebuah aksi. Hak akses ini disimpan dalam bentuk “Role”.

1. Buat pengguna lain yang memiliki role selain root.

Login menggunakan akun root lalu jalankan perintah berikut ini menambahkan pengguna contoh ke basis data rentalfilm yang memiliki peran read dalam basis data rentalfilm:

```

test> use rentalfilm
switched to db rentalfilm
rentalfilm > db.createUser(
    {
        user: "contoh",
        pwd: "contoh",
        roles: [
            { role: "read", db: "rentalfilm" }
        ]
    }
);

```

```

test> use rentalfilm
switched to db rentalfilm
rentalfilm> db.createUser(
...     {
...         user: "contoh",
...         pwd: "contoh",
...         roles: [
...             { role: "read", db: "rentalfilm" }
...         ]
...     }
... );
{ ok: 1 }

```

Perintah diatas akan membuat user baru pada basis data rentalfilm yang memiliki role “read”, artinya user tersebut bisa melihat database namun tidak bisa melakukan perintah lain seperti insert, update, dll.

2. Login Sebagai user selain root memiliki perintah yang sedikit berbeda contohnya sebagai berikut:

```
C:\Users\Denny->mogosh --username contoh --password contoh - authenticationDatabase rentalfilm
```

3. Ketika menggunakan user yang hanya memiliki role “read” dan menjalankan perintah insert maka akan mendapatkan error.

```
rentalfilm> db.Films.insertOne({})
```

```

rentalfilm> db.Films.insertOne({})
MongoServerError: not authorized on rentalfilm to execute command { insert: "Films", documents: [ { _id: ObjectId('647443f1842248ae093c94b8') } ], ordered: true, lsid: { id: UUID("cbda84f7-8fe3-4017-b7e8-be237e9e5c2e") }, $db: "rentalfilm" }
rentalfilm>

```

4. Buat pengguna baru dengan role “readWrite”.

```

rentalfilm > db.createUser(
    {
        user: "conto2h2",
        pwd: "contoh",
        roles: [
            { role: "readWrite", db: "rentalfilm" }
        ]
    }
);

```

```

rentalfilm> db.createUser(
...   {
...     user: "contoh2",
...     pwd: "contoh2",
...     roles: [
...       { role: "readWrite", db: "rentalfilm" }
...     ]
...   }
... );
{ ok: 1 }

```

Perintah diatas akan membuat pengguna dengan username contoh2 dan password contoh2 yang memiliki role readWrite yaitu membaca dan menulis pada database rentalfilm.

5. Login menggunakan user contoh2 dan jalankan perintah insertOne.

```
C:\Users\Denny->mogosh --username contoh2 --password contoh2 --
authenticationDatabase rentalfilm
```

```
Rentalfilm> db.test.insertOne({});
```

```
C:\Users\Denny>mongosh --username contoh2 --password contoh2 --authenticationDatabase rentalfilm
```

```

rentalfilm> db.test.insertOne({});
{
  acknowledged: true,
  insertedId: ObjectId("6474475e66952f81c5bf5e63")
}

```

Apabila user memiliki authorization yang mengizinkan user tersebut untuk melakukan perintah insert maka, query akan sukses dijalankan.

6. Di MongoDB mendukung pembuatan custom role yang bisa memiliki hak yang berbeda. Contohnya ketika ingin membatasi pengguna hanya bisa mengakses 1 buah collection dalam sebuah database kita bisa menggunakan “Privileges”. Privileges merupakan cara untuk membatasi akses dalam level Collections.

Untuk membuat role baru, wajib memiliki user dengan role “userAdmin”, “dbAdmin” atau “root”. Perintah yang digunakan untuk membuat role baru adalah :

```

rentalfilm> db.createRole({
  role: "readOnly", //nama role
  privileges: [], //privileges
  roles: [           //daftar array role untuk mengambil privileges dari
    role yang sudah ada
      { role: "read", db: "rentalfilm" }
    ]
});

```

```

rentalfilm> db.createRole({
...   role: "readOnly", //nama role
...   privileges: [], //privileges
...   roles: [           //daftar array role untuk mengambil privileges dari role yang sudah ada
...     {
...       role: "read",
...       db: "rentalfilm"
...     }
...   ]
... });

```

Query diatas akan membuat role baru dengan nama readOnly yang hanya memiliki hak untuk membaca data di database rentalfilm.

7. Kita bisa melihat daftar roles yang ditambahkan dengan menggunakan perintah db.getRoles().

```
rentalfilm> db.getRoles();
```

```
rentalfilm> db.getRoles();
{
  roles: [
    {
      _id: 'rentalfilm.readOnly',
      role: 'readOnly',
      db: 'rentalfilm',
      roles: [ { role: 'read', db: 'rentalfilm' } ],
      isBuiltIn: false,
      inheritedRoles: [ { role: 'read', db: 'rentalfilm' } ]
    }
  ],
  ok: 1
}
```

8. Kita bisa menggubah salah satu user yang sudah ada dan mengganti role nya dengan role yang baru saja dibuat dengan menggunakan perintah updateUser.

```
rentalfilm> db.updateUser("contoh2",
  {
    Roles: [
      { role: "readOnly", db: "rentalfilm" }
    ]
  }
);
```

Ketika user contoh2 melakukan perintah insert maka akan muncul error:

```
rentalfilm> db.city.insertOne({});
```

```
rentalfilm> db.test.insertOne({});
MongoServerError: not authorized on rentalfilm to execute command { insert: "test", documents: [ { _id: Object
Id('64744ab6076f14cff7d64b52a') } ], ordered: true, lsid: { id: UUID("7b32425c-4b79-4c53-804f-ab5655e890ea") },
$db: "rentalfilm" }
```

9. Untuk membatasi hak akses collection kita bisa menambahkan di bagian Privileges ketika membuat role atau melakukan update pada role yang sudah dibuat dengan menggunakan perintah updateRole.

```
rentalfilm> db.updateRole("readOnly", {
  privileges: [
    {
      resource: {
        db: "rentalfilm", //nama database
        collection: "city" // nama collection yang diberi akses
      },
    ],
    roles: [
      { role: "read",
        db: "rentalfilm"
      }
    ]
  });
});
```

```
rentalfilm> db.updateRole("readOnly", {  
...   privileges: [  
...     {  
...       resource: {  
...         db: "rentalfilm", //nama database  
...         collection: "city" // nama collection yang diberi akses  
...       },  
...       actions: [ "insert" ] //query yang bisa dijalankan  
...     }  
...   ],  
...   roles: [  
...     {  
...       role: "read",  
...       db: "rentalfilm"  
...     }  
...   ]  
...});  
{ ok: 1 }
```

10. Login kembali menggunakan user contoh2 dan jalankan perintah insert data.

```
rentalfilm> db.city.insertOne({});
```

```
rentalfilm> db.city.insertOne({});  
{  
  acknowledged: true,  
  insertedId: ObjectId("64744cb609e50ed5bd12c6d0")  
}
```

Karena role readOnly sudah ditambahkan privileges untuk melakukan insert maka query tersebut berhasil berjalan.

8.5. Tugas dan Latihan

1. Buat user baru dengan role readWrite dan jalankan query untuk menambahkan lalu melihat data pada database rentalfilm.
2. Hapus user baru yang sudah dibuat.
3. Buat role baru yang hanya memiliki hak untuk membaca semua collection dan menghapus data dari sebuah collection.
4. Buat user baru dengan role yang baru dibuat dari nomor 3 dan jalankan perintah find, insert, delete dan update.

MODUL 9

MENGENAL GRAPH DATABASE

9.1. Tujuan Praktikum

1. Mahasiswa memahami konsep database Graph
2. Mahasiswa mampu menginstall Neo4j
3. Mahasiswa mampu mengimplementasikan query dasar pada database Graph

9.2. Dasar Teori

9.2.1. Graph Database

Graf adalah representasi bergambar dari sekumpulan objek di mana beberapa pasangan objek dihubungkan dengan tautan. Graf terdiri dari dua elemen - node (simpul) dan hubungan (sisi).

Basis data graf adalah basis data yang digunakan untuk memodelkan data dalam bentuk graf. Di sini, node dari sebuah graf menggambarkan entitas sedangkan relasi menggambarkan hubungan dari node-node tersebut.

Saat ini, sebagian besar data ada dalam bentuk hubungan antara objek yang berbeda dan lebih sering, hubungan antara data lebih berharga daripada data itu sendiri. Basis data relasional menyimpan data yang sangat terstruktur yang memiliki beberapa catatan yang menyimpan jenis data yang sama sehingga dapat digunakan untuk menyimpan data terstruktur dan, mereka tidak menyimpan hubungan antara data. Tidak seperti basis data lainnya, basis data graph menyimpan hubungan dan koneksi sebagai entitas kelas satu.

Model data untuk basis data graph lebih sederhana dibandingkan dengan basis data lainnya dan, mereka dapat digunakan dengan sistem OLTP. Mereka menyediakan fitur-fitur seperti integritas transaksional dan ketersediaan operasional.

9.2.2. RDBMS Vs Graph Database

Berikut ini adalah tabel yang membandingkan database Relasional dan database Graph:

| No | RDMBS | Graph Database |
|----|------------------|----------------------|
| 1 | Tabel | Graph |
| 2 | Baris | Nodes(simpul) |
| 3 | Columns dan Data | Properties dan nilai |

| | | |
|---|-------------|--------------|
| 4 | Constraints | Relationship |
| 5 | Joins | Traversal |

9.2.3. *Neo4j*

Neo4j adalah sebuah sistem manajemen basis data graph yang populer. Dikembangkan oleh Neo4j, Inc., Neo4j menyediakan struktur penyimpanan dan pemrosesan data yang didasarkan pada model graph.

Dalam Neo4j, data diorganisasi dalam bentuk graph yang terdiri dari simpul (node) yang mewakili entitas dan relasi (edge) yang mewakili hubungan antara entitas tersebut. Setiap simpul dapat memiliki atribut yang menyimpan informasi tambahan tentang entitas, sedangkan setiap relasi dapat memiliki atribut yang mendefinisikan sifat dan detail dari hubungan tersebut.

Keuntungan menggunakan Neo4j sebagai basis data graph adalah kemampuannya untuk mengelola dan menganalisis data yang memiliki struktur hubungan yang kompleks. Neo4j menyediakan kemampuan pencarian yang kuat, navigasi graph yang efisien, dan algoritma graph yang dapat digunakan untuk menganalisis dan mendapatkan wawasan dari data graph.

Neo4j digunakan secara luas dalam berbagai aplikasi dan industri, termasuk media sosial, e-commerce, keamanan, analisis jaringan, ilmu pengetahuan, dan banyak lagi. Platform ini menyediakan API dan bahasa kueri bernama Cypher yang digunakan untuk mengakses dan memanipulasi data dalam basis data graph Neo4j.

9.2.4. *Kelebihan dari Neo4j*

- **Model data yang fleksibel:** Neo4j menyediakan model data yang fleksibel, sederhana, namun kuat, yang dapat dengan mudah diubah sesuai dengan aplikasi dan industri.
- **Real-time insight:** Neo4j memberikan hasil berdasarkan data real-time.
- **Ketersediaan tinggi:** Neo4j sangat tersedia untuk aplikasi real-time perusahaan besar dengan jaminan transaksional.
- **Data yang saling terhubung dan semi struktur:** Dengan menggunakan Neo4j, Anda dapat dengan mudah merepresentasikan data yang terhubung dan semi terstruktur.
- **Pengambilan yang mudah:** Dengan menggunakan Neo4j, Anda tidak hanya dapat merepresentasikan tetapi juga dengan mudah mengambil (melintasi / menavigasi) data yang terhubung dengan lebih cepat jika dibandingkan dengan database lain.

- **Bahasa kueri Cypher:** Neo4j menyediakan bahasa kueri deklaratif untuk merepresentasikan graph secara visual, menggunakan sintaks ascii-art. Perintah-perintah dari bahasa ini dalam format yang dapat dibaca oleh manusia dan sangat mudah untuk dipelajari.
- **Tanpa penggabungan(joins):** Dengan menggunakan Neo4j, TIDAK memerlukan gabungan yang rumit untuk mengambil data yang terhubung/berhubungan karena sangat mudah untuk mengambil simpul yang berdekatan atau detail hubungan tanpa gabungan atau indeks.

9.2.5. *Fitur dari Neo4j*

Berikut ini adalah sejumlah fitur penting dari Neo4j:

- **Data model (flexible schema)** : Neo4j mendukung aturan ACID (Atomisitas, Konsistensi, Isolasi, dan Daya Tahan) secara penuh.
- **Skalabilitas dan keandalan:** Anda dapat menskalakan basis data dengan meningkatkan jumlah pembacaan/penulisan, dan volume tanpa mempengaruhi kecepatan pemrosesan kueri dan integritas data. Neo4j juga menyediakan dukungan untuk replikasi untuk keamanan dan keandalan data.
- **Bahasa Query Cypher:** Neo4j menyediakan bahasa kueri deklaratif yang kuat yang dikenal sebagai Cypher. Bahasa ini menggunakan seni ASCII untuk menggambarkan graph. Cypher mudah dipelajari dan dapat digunakan untuk membuat dan mengambil relasi antar data tanpa menggunakan query yang rumit seperti Join.
- **Aplikasi web bawaan:** Neo4j menyediakan aplikasi web bawaan Neo4j Browser. Dengan menggunakan ini, Anda dapat membuat dan melakukan kueri terhadap data graph Anda.
- **Driver:** Neo4j dapat bekerja dengan -
 - REST API untuk bekerja dengan bahasa pemrograman seperti Java, Spring, Scala, dll.
 - Java Script untuk bekerja dengan kerangka kerja UI MVC seperti Node JS.
 - Mendukung dua jenis Java API: API Cypher dan API Java asli untuk mengembangkan
- **Aplikasi Java.**
- Selain itu, Anda juga dapat bekerja dengan database lain seperti MongoDB, Cassandra, dll.

- **Pengindeksan:** Neo4j mendukung Indeks dengan menggunakan Apache Lucence.

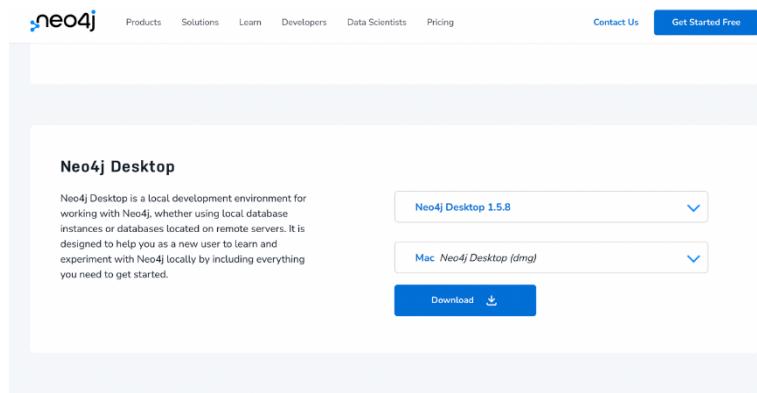
9.3. Software

1. Neo4j

9.4. Tahapan Kerja

9.4.1. Instalasi Neo4j

1. Kunjungi website resmi Neo4j <https://neo4j.com/download/>. (sesuaikan dengan jenis OS kalian)



2. Registrasi untuk bisa mendownload Neo4j Desktop.

Get Started Now

Please fill out this form to begin your download

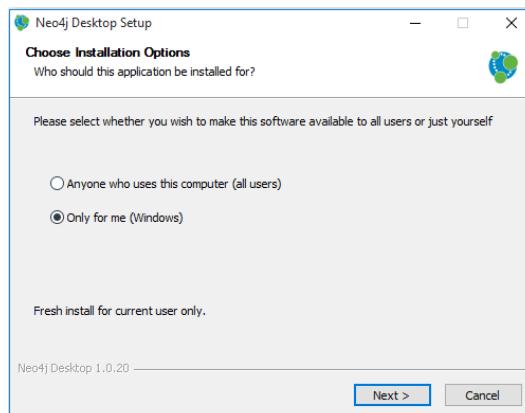
| | |
|---|---------------------------|
| * | aska |
| * | cahyadi |
| * | askaion.cahyadi@gmail.com |
| * | IBI Kesatuan |
| * | 08999506210 |
| * | Indonesia |

By downloading you agree to the [Neo4j License Agreement for Neo4j Desktop Software](#).

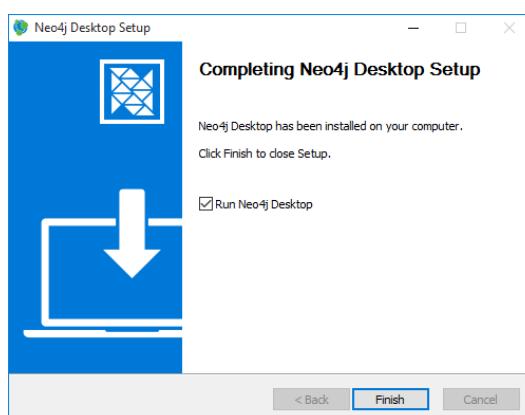
Download Desktop

The information you provide will be used in accordance with the terms of our [privacy policy](#).

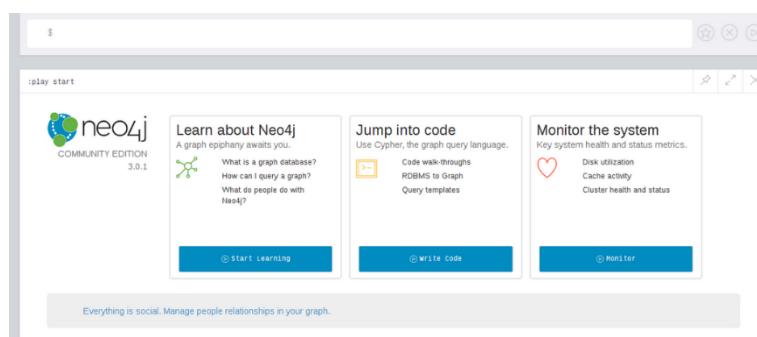
3. Double click file yang telah di download. Lalu install



4. Lalu akan muncul tampilan seperti dibawah yang menandakan telah berhasil install.



5. Setelah aplikasi sudah berhasil terinstall, lalu buka link alamat berikut <http://localhost:7474/> lalu akan muncul tampilan seperti dibawah atau kalian bisa buka aplikasi Noe4j desktop yang sudah di download.



9.4.2. Membuat Graph

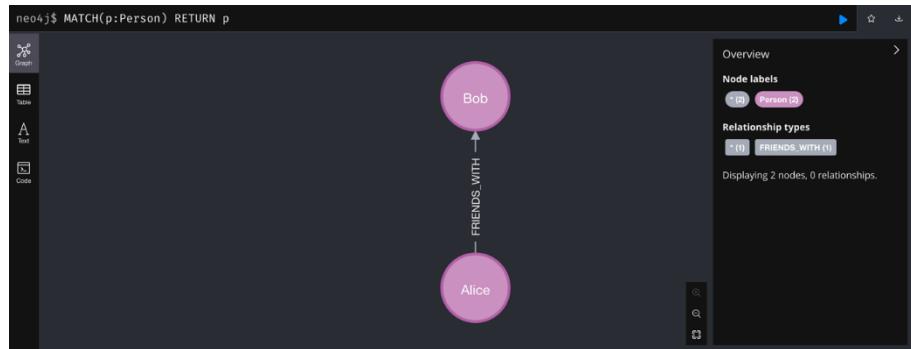
1. Buka Neo4j Browser dan jalankan perintah berikut untuk membuat simpul dan relasi baru dalam graph:

```
CREATE (p1:Person {name: 'Alice', age: 30})
CREATE (p1:Person {name: 'Bob', age: 30})
CREATE (p1)-[:FRIENDS_WITH]-(p2)
```

```
Added 2 labels, created 2 nodes, set 4 properties, created 1 relationship, completed after 32 ms.
```

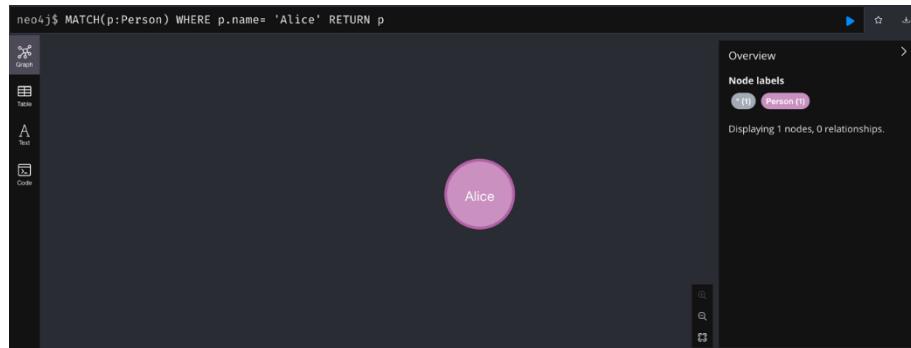
2. Melakukan Pencarian. Gunakan perintah **MATCH** untuk melakukan pencarian dalam graph. Misalnya, untuk mencari semua simpul dengan label "Person":

```
MATCH (p:Person)  
RETURN p
```



3. Melakukan Pencarian dengan Kondisi. Gunakan perintah **WHERE** untuk menambahkan kondisi pada pencarian. Misalnya, untuk mencari simpul dengan nama "Alice":

```
MATCH (p:Person)  
WHERE p.name = 'Alice'  
RETURN p
```



4. Mencari Relasi. Gunakan pola relasi dalam perintah **MATCH** untuk mencari relasi antara simpul. Misalnya, untuk mencari relasi "FRIENDS_WITH":

```
MATCH (p:Person)-[r:FRIENDS_WITH]-(p2:Person)  
RETURN p1, r, p2
```

```

1 MATCH(p1:Person)-[r:FRIENDS_WITH]-(p2:Person)
2 RETURN p1, r, p2
  
```

The screenshot shows the Neo4j Browser interface with the following details:

- Graph View:** Shows a circular node for "Alice" and another for "Bob". A directed edge labeled "FRIENDS_WITH" connects them.
- Overview Panel:** Displays "Node labels: Person" and "Relationship types: FRIENDS_WITH".
- Code Panel:** Shows the executed query: `MATCH(p1:Person)-[r:FRIENDS_WITH]-(p2:Person) RETURN p1, r, p2`.
- Table Panel:** Shows the returned data in a table format:

| | p1 | r | p2 |
|---|---|---|---|
| 1 | { "identity": 0, "labels": ["Person"], "properties": { "name": "Alice", "age": 30 }, "elementId": "4:fc1fbcd-32b4-41d6-b07f-14e6e283ee39:0" } | { "identity": 0, "start": 0, "end": 1, "type": "FRIENDS_WITH", "properties": { "elementId": "5:fc1fbcd-32b4-41d6-b07f-14e6e283ee39:0", "startNodeElementId": "4:fc1fbcd-32b4-41d6-b07f-14e6e283ee39:0", "endNodeElementId": "4:fc1fbcd-32b4-41d6-b07f-14e6e283ee39:1" } } | { "identity": 1, "labels": ["Person"], "properties": { "name": "Bob", "age": 35 }, "elementId": "4:fc1fbcd-32b4-41d6-b07f-14e6e283ee39:1" } |

- Menggunakan Fungsi Agregasi. Gunakan fungsi agregasi seperti **COUNT**, **SUM**, **AVG**, dll., untuk melakukan perhitungan pada hasil kueri. Misalnya, untuk menghitung jumlah simpul dengan label "Person":

```

MATCH (p:Person)
RETURN COUNT(p) as totalPersons
  
```

| totalPersons |
|--------------|
| 2 |

The screenshot shows the Neo4j Browser interface with the following details:

- Table View:** Shows a single row with the value "2" under the column "totalPersons".
- Code Panel:** Shows the executed query: `MATCH (p:Person) RETURN COUNT(p) AS totalPersons`.
- Text Panel:** Shows the returned data: `totalPersons` followed by the value "2".

- Mengurutkan dan Membatasi Hasil. Gunakan perintah **ORDER BY** untuk mengurutkan hasil kueri berdasarkan properti tertentu. Misalnya, untuk mengurutkan simpul berdasarkan umur secara menurun:

```

MATCH (p:Person)
RETURN p
ORDER BY p.age DESC
  
```

```
neo4j$ MATCH (p:Person) RETURN p ORDER BY p.age DESC
```

Graph

Table

Text

Code

MAX COLUMN WIDTH:

7. Menghapus relasi. Untuk menghapus relasi "FRIENDS_WITH" antara dua orang, gunakan query berikut:

```
MATCH (person1:Person)-[rel:FRIENDS_WITH]-(person2:Person)
WHERE person1.name = 'Alice' AND person2.name = 'Bob'
DELETE rel
```

```
neo4j$ MATCH (p1:Person)-[rel:FRIENDS_WITH]-(p2:Person) WHERE p1.name = 'Alice' AND p2.name = 'Bob' DELETE rel
```

Table

Code

Deleted 1 relationship, completed after 17 ms.

8. Menghapus Data. Gunakan perintah **DELETE** untuk menghapus simpul atau relasi dari graph. Misalnya, untuk menghapus simpul dengan nama "Alice":

```
MATCH (p:Person {name:'Alice'})
DELETE p
```

The screenshot shows the Neo4j browser interface. At the top, there is a command bar with the text "neo4j\$ MATCH (p:Person {name: 'Alice'}) DELETE p" and a note "Cmd+click to copy to main editor". Below the command bar is a sidebar with four items: "Table", "Code", "Text", and "Graph". The "Text" item is currently selected. The main area displays the message "Deleted 1 node, completed after 3 ms." twice.

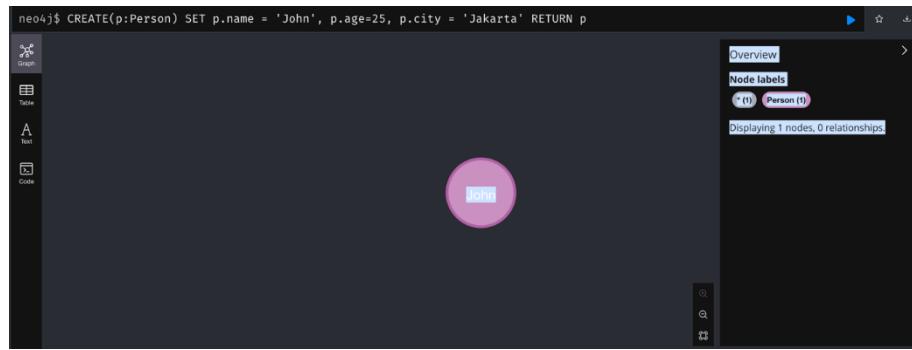
9. Melakukan Pembaruan Data. Gunakan perintah **SET** untuk memperbarui properti pada simpul atau relasi. Misalnya, untuk mengubah umur seseorang menjadi 40:

```
MATCH (p:Person {name:'Alice'})
SET p.age = 40
RETURN p
```

The screenshot shows the Neo4j browser interface. At the top, there is a command bar with the text "neo4j\$ MATCH (p:Person {name:'Alice'}) SET p.age=40 RETURN p". Below the command bar is a sidebar with four items: "Graph", "Table", "Text", and "Code". The "Text" item is currently selected. The main area displays a single node labeled "Alice" with a pink circular background, indicating it has been updated.

10. Query kombinasi CREATE dan SET. Membuat simpul baru dengan label "Person" dan mengatur beberapa properti:

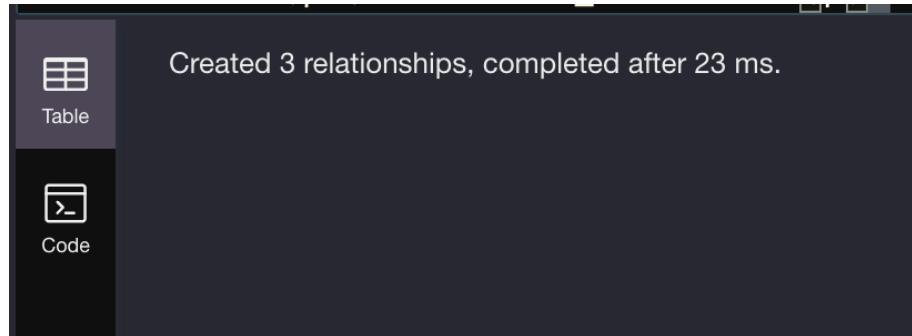
```
CREATE (p:Person)
SET p.name = 'John', p.age = 25, p.city = 'Jakarta'
RETURN p
```



11. Membuat hubungan MUTUAL_FRIEND untuk Person John

```

MATCH (p:Person {name:'John'})
MATCH (p2:Person {name: 'Bob'})
MATCH (p3:Person {name:'Alice'})
CREATE (p)-[:FRIENDS_WITH]-(p2)
CREATE (p)-[:MUTUAL_FRIEND]-(p3)
CREATE (p3)-[:MUTUAL_FRIEND]-(p)
    
```



12. Query kombinasi MATCH, SET, dan DELETE. Mencari simpul dengan label "Person" yang memiliki nama "John", mengubah umurnya menjadi 30, dan menghapus relasi "FRIENDS_WITH" yang terhubung dengan simpul tersebut:

```

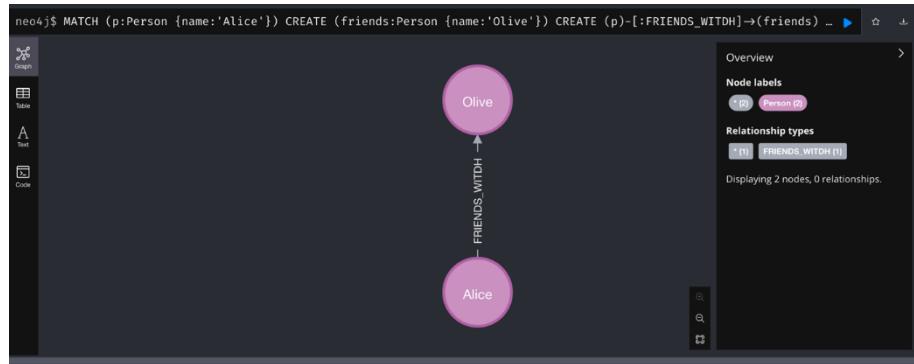
MATCH (p:Person {name: 'John'})
OPTIONAL MATCH (p)-[r:MUTUAL_FRIEND]-()
SET p.age=30
DELETE r
RETURN p
    
```



13. Query kombinasi MATCH, CREATE, dan SET. Mencari simpul dengan label "Person" yang memiliki nama "Alice", membuat simpul teman baru, dan mengatur relasi

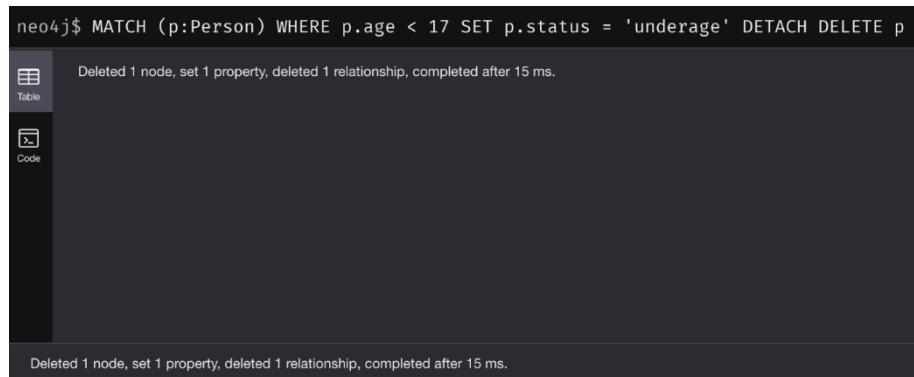
"FRIENDS_WITH" antara mereka:

```
MATCH (p:Person {name: 'Alice'})  
CREATE(friends:Person {name: 'Olive'})  
CREATE (p)-[:FRIENDS_WITH]-(friends)  
RETURN p, friends
```



14. Query kombinasi MATCH, SET, dan DELETE. Mencari simpul dengan label "Person" yang memiliki umur di bawah 20 tahun, mengubah properti "status" mereka menjadi "Underage", dan menghapus simpul tersebut beserta relasinya:

```
MATCH (p:Person)  
WHERE p.age < 20  
SET p.status = 'Underage'  
DETACH DELETE p
```

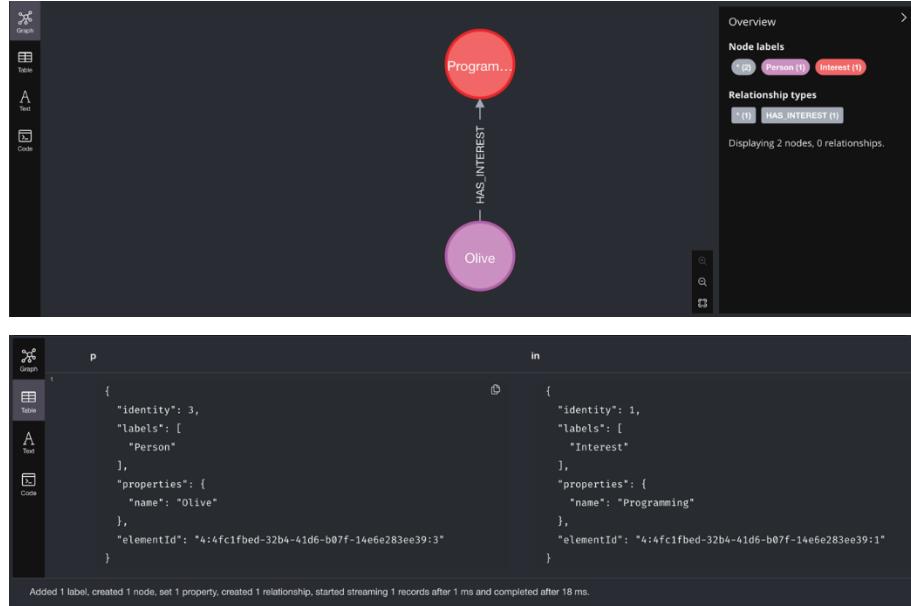


15. Query kombinasi MATCH, CREATE, dan SET. Mencari simpul dengan label "Person" yang memiliki nama "Olive", membuat simpul baru dengan label "Interest", dan mengatur relasi "HAS_INTEREST" antara simpul tersebut:

```

MATCH (p:Person {name: 'Jane'})
CREATE(in:Interest {name: 'Programming'})
CREATE (p)-[:HAS_INTEREST]->(in)
RETURN p, in

```



9.5. Tugas dan Latihan

Buatlah Query berikut:

1. Temukan semua orang yang tinggal di kota 'Jakarta'.
2. Berapa total jumlah teman dari setiap individu?
3. Update umur semua orang menjadi 40 tahun.
4. Hapus semua relasi 'FRIENDS_WITH' yang terhubung dengan individu yang berumur di atas 50 tahun.
5. Buatlah simpul baru dengan label "Hobby" dan properti "name" adalah 'Reading', kemudian hubungkan dengan individu 'Alice' dengan relasi 'HAS_HOBBY'.
6. Hapus semua simpul dengan label "Hobby" yang tidak memiliki relasi 'HAS_HOBBY'.
7. Tambahkan properti "gender" dengan nilai 'Male' pada semua orang yang berumur di atas 30 tahun.
8. Temukan teman-teman dari teman individu 'Alice' yang tinggal di kota yang sama.
9. Berapa jumlah individu yang memiliki properti "gender"?
10. Hapus semua simpul dan relasi dalam graph.

MODUL 10

CYpher Clause di Graph

10.1. Tujuan Praktikum

Mahasiswa mampu mengimplementasikan query clause pada database Graph

10.2. Dasar Teori

10.2.1. MATCH

MERGE digunakan untuk mencocokkan atau membuat simpul dan relasi jika belum ada, atau menggabungkan dengan yang sudah ada jika ada. Jika cocok, perintah MERGE akan mengembalikan simpul atau relasi yang cocok, dan jika tidak cocok, akan membuatnya. Ini memungkinkan operasi pencocokan dan pembuatan dalam satu perintah.

```
MERGE (p:Person {name: 'Alice'})
ON CREATE SET p.createdAt = timestamp()
ON MATCH SET p.updatedAt = timestamp()
RETURN p
```

Dalam contoh di atas, jika simpul "Person" dengan properti "name" adalah 'Alice' ada, maka simpul tersebut akan ditemukan dan properti "updatedAt" akan diperbarui. Jika tidak ada, simpul baru akan dibuat dengan properti "name" adalah 'Alice' dan properti "createdAt" diatur.

10.2.2. OPTIONAL MATCH

OPTIONAL MATCH digunakan untuk mencocokkan pola pencarian seperti yang dilakukan oleh perintah MATCH, tetapi dengan perbedaan bahwa jika tidak ada pencocokan yang ditemukan, perintah akan tetap dilanjutkan dan mengembalikan nilai kosong. Berikut contoh penggunaannya

```
OPTIONAL MATCH (p:Person)-[:FRIENDS_WITH]-(friend:Person)
RETURN p, friend
```

Dalam contoh di atas, perintah OPTIONAL MATCH mencari semua pasangan simpul "Person" yang saling berteman. Jika tidak ada teman yang ditemukan, perintah akan tetap dilanjutkan dan mengembalikan nilai kosong.

10.2.3. FOREACH

FOREACH digunakan untuk melakukan tindakan yang diulang pada setiap elemen dalam koleksi. Biasanya digunakan untuk mengubah atau memodifikasi simpul atau relasi berdasarkan data yang ada. Contoh penggunaannya:

```
MATCH (p:Person)
WHERE p.age < 18
FOREACH (person IN COLLECT(p) |
    SET person.isMinor = true
)
```

Dalam contoh di atas, perintah FOREACH digunakan untuk mengubah properti "isMinor" menjadi true pada setiap simpul "Person" yang memiliki usia di bawah 18 tahun.

10.2.4. UNION

UNION digunakan untuk menggabungkan hasil dari dua atau lebih perintah MATCH atau perintah UNION lainnya menjadi satu hasil tunggal. Contoh penggunaan:

```
MATCH (p:Person)
WHERE p.name = 'Alice'
UNION
MATCH (p:Person)
WHERE p.name = 'Bob'
RETURN p
```

Dalam contoh di atas, perintah UNION menggabungkan hasil pencarian dari dua perintah MATCH yang berbeda menjadi satu hasil tunggal yang berisi simpul "Person" dengan nama 'Alice' atau 'Bob'.

10.2.5. WITH

WITH digunakan untuk membagi hasil pencarian menjadi beberapa langkah query yang terpisah. Ini berguna ketika Anda ingin menggunakan hasil pencarian sebelumnya dalam perintah-perintah berikutnya. Contoh soalnya:

```
MATCH (p:Person)-[:FRIENDS_WITH]-(friend:Person)
WITH p, COLLECT(friend) AS friends
RETURN p, friends
```

Dalam contoh di atas, perintah WITH digunakan untuk mengumpulkan semua teman dari setiap simpul "Person" dan menyimpannya sebagai koleksi "friends" untuk digunakan dalam perintah-perintah selanjutnya.

10.2.6. DETACH DELETE

DETACH DELETE digunakan untuk menghapus simpul dan relasinya dari graph. Ini juga menghapus semua relasi yang terhubung dengan simpul yang dihapus. Contoh penggunaan:

```
MATCH (p:Person)
WHERE p.age > 60
DETACH DELETE p
```

Dalam contoh di atas, perintah DETACH DELETE digunakan untuk menghapus semua simpul "Person" yang memiliki usia di atas 60 tahun dan juga menghapus semua relasi yang terhubung dengan simpul-simpul tersebut.

10.3. Software

1. Neo4j

10.4. Tahapan Kerja

Berdasarkan pertemuan minggu lalu, buatlah query berikut ini:

1. Temukan semua orang yang tinggal di kota 'Jakarta'.

```
{  
    "identity": 2,  
    "labels": [  
        "Person"  
    ],  
    "properties": {  
        "city": "Jakarta",  
        "name": "Bob",  
        "age": 35  
    },  
    "elementId": "4:4fc1fbcd-32b4-41d6-b07f-14e6e283ee39:2"  
}  
  
Started streaming 1 records after 4 ms and completed after 5 ms.
```

```
MATCH (p:Person)
WHERE p.city = 'Jakarta'
RETURN p
```

2. Berapa total jumlah teman dari setiap individu:

```

1 MATCH (p:Person)-[:FRIENDS_WITH]-(friend:Person)
2 RETURN p.name AS person, COUNT(friend) AS totalFriends
3 ORDER BY totalFriends DESC;

```

| person | totalFriends |
|-----------|--------------|
| 1 "Alice" | 1 |
| 2 "Bob" | 1 |

| person | totalFriends |
|---------|--------------|
| "Alice" | 1 |
| "Bob" | 1 |

```

MATCH (p:Person)-[:FRIENDS_WITH]-(friend:Person)
RETURN p.name AS person, COUNT(friend) AS totalFriends
ORDER BY totalFriends DESC;

```

3. Update umur semua orang menjadi 40 tahun.

```

neo4j$ MATCH (p:Person) SET p.age =40 RETURN p

```

| Graph |
|---|
| p |
| (:Person {name: "Alice",age: 40}) |
| (:Person {city: "Jakarta",name: "Bob",age: 40}) |
| (:Person {name: "Olive",age: 40}) |

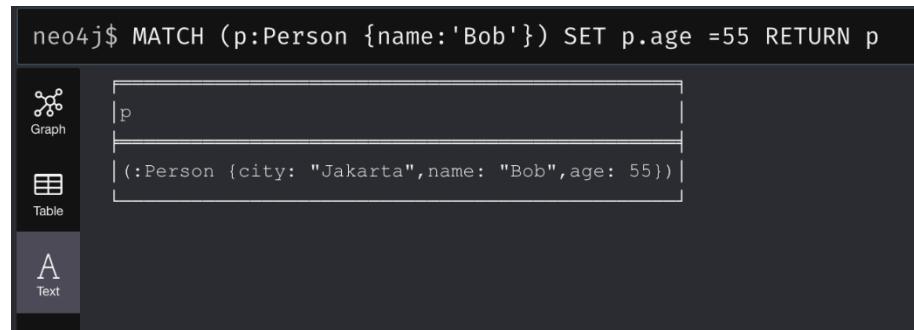
```

MATCH (p:Person)
WHERE p.age = 40
RETURN p

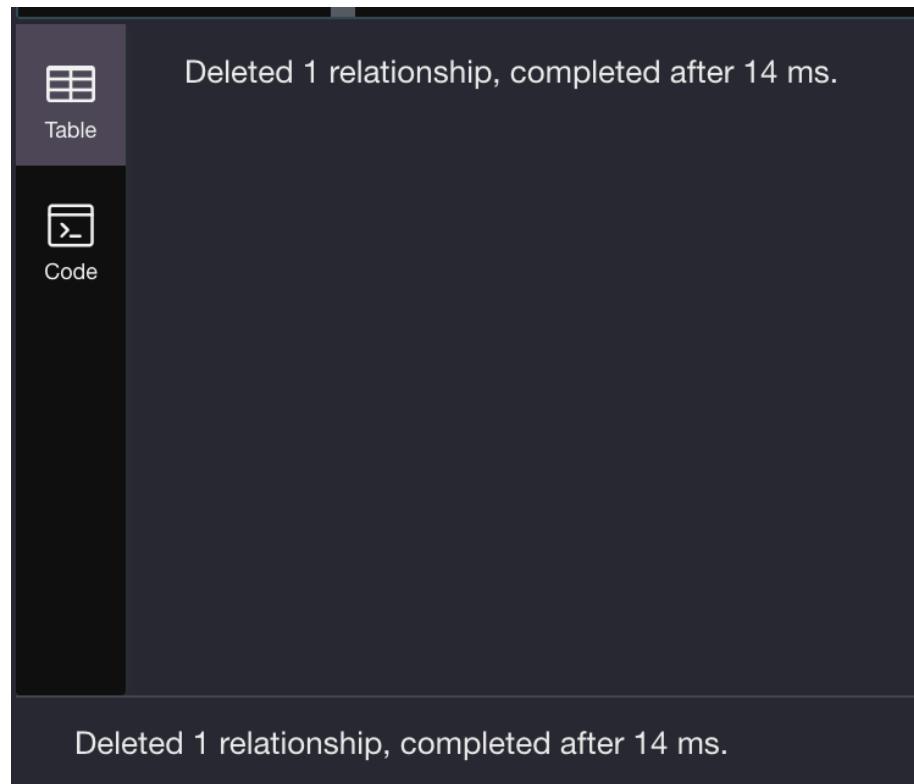
```

4. Update usia Bob menjadi 55 tahun

```
neo4j$ MATCH (p:Person {name:'Bob'}) SET p.age =55 RETURN p
```



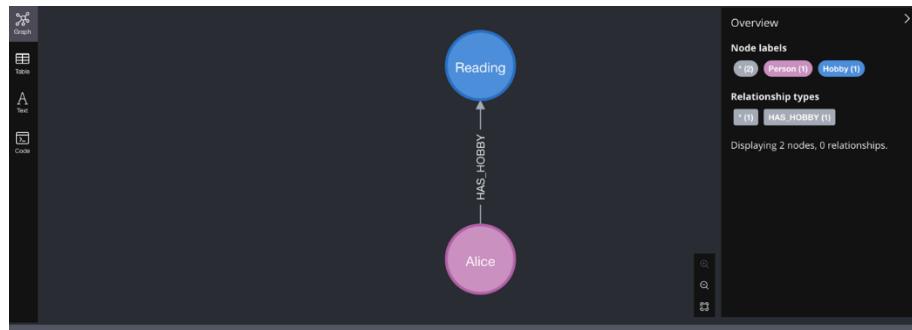
5. Hapus semua relasi 'FRIENDS_WITH' yang terhubung dengan individu yang berumur di atas 50 tahun.



```
Deleted 1 relationship, completed after 14 ms.  
Deleted 1 relationship, completed after 14 ms.
```

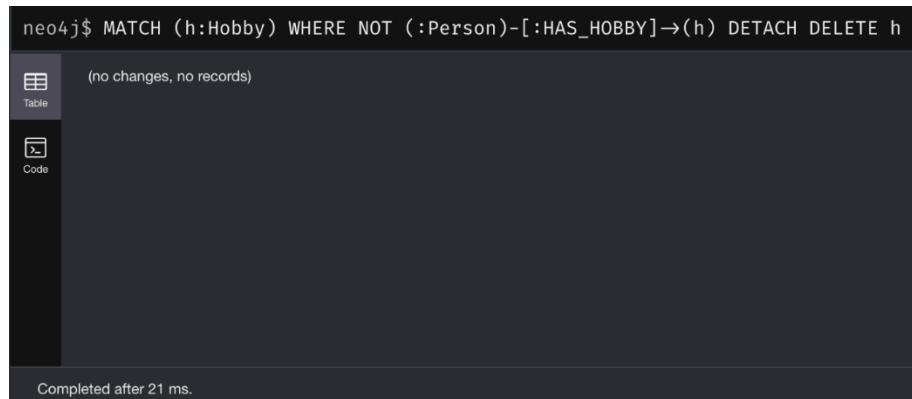
```
MATCH (p:Person)  
WHERE p.age > 50  
OPTIONAL MATCH(p)=[r:FRIENDS_WITH]-()  
DELETE r
```

6. Buatlah simpul baru dengan label "Hobby" dan properti "name" adalah 'Reading', kemudian hubungkan dengan individu 'Alice' dengan relasi 'HAS_HOBBY'.



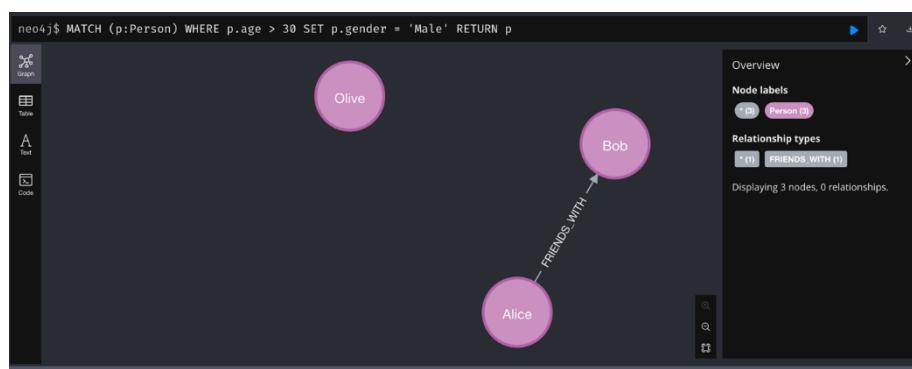
```
MATCH (p:Person {name: 'Alice'})
CREATE (hobby:Hobby {name: 'Reading'})
SET p-[:HAS_HOBBY]->(hobby)
RETURN p, hobby
```

- Hapus semua simpul dengan label "Hobby" yang tidak memiliki relasi 'HAS_HOBBY'



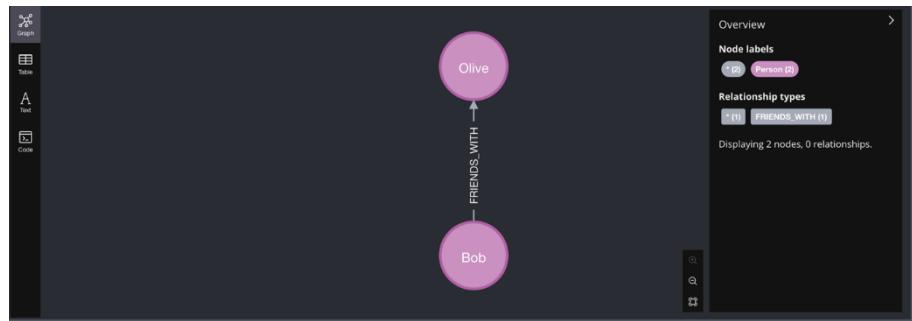
```
MATCH (h:Hobby)
WHERE NOT (:Person)-[:HAS_HOBBY]->(h)
DETACH DELETE h
```

- Tambahkan properti "gender" dengan nilai 'Male' pada semua orang yang berumur di atas 30 tahun.



```
MATCH (p:Person)
WHERE p.age > 30
SET p.gender = 'Male'
RETURN p
```

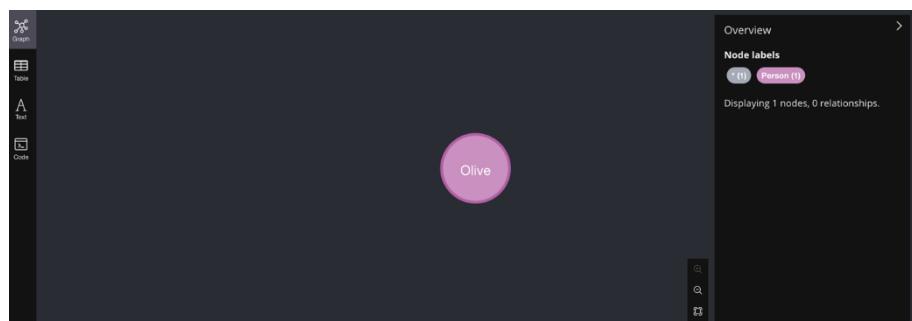
- Buat hubungan pertemanan antara Bob dan Olive



```

MATCH (p:Person {name: 'Bob'})
MATCH (p2:Person {name: 'Olive'})
CREATE (p)-[:FRIENDS_WITH]->(p2)
RETURN p,p2
  
```

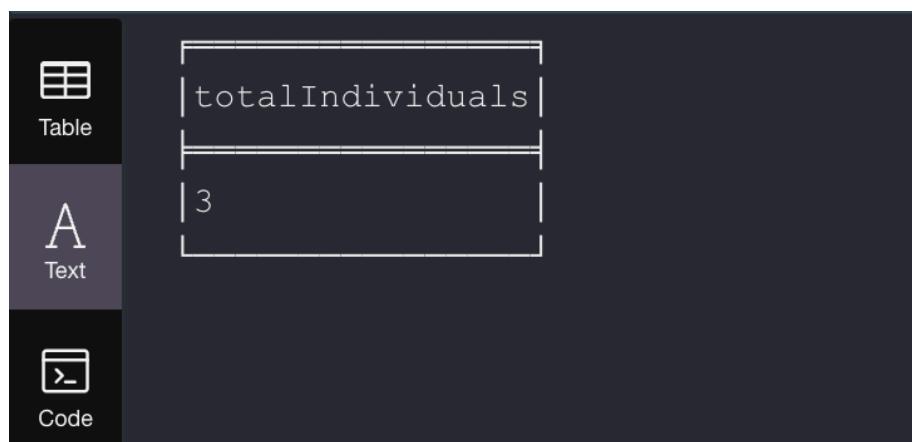
10. Temukan teman-teeman dari teman individu 'Alice' yang tinggal di kota yang sama.



```

MATCH (p1:Person {name: 'Alice'})-[:FRIENDS_WITH]-(friend:Person)-
[:FRIENDS_WITH]-(friendOfFriend:Person)
WHERE friend.city = p1.city
RETURN friendOfFriend
  
```

11. Berapa jumlah individu yang memiliki properti "gender"?



```

MATCH (p:Person)
WHERE p.gender IS NOT NULL
RETURN COUNT(p) AS totalIndividuals
  
```

```

MATCH (p:Person)
WHERE EXISTS(p.gender)
RETURN COUNT(p) AS totalIndividuals
  
```

```
MERGE UNIQUE(p:Person {name: 'John'})  
RETURN p
```

12. Mencocokkan atau membuat simpul "Person" dengan properti "name" adalah "John" jika belum ada dalam graph:

```
MERGE (p:Person {name: "Archie"})  
RETURN p
```



A screenshot of the Neo4j browser interface. On the left, there's a sidebar with icons for Graph, Table, Text, and Code. The main area is titled 'p' and shows a single node with the following properties:

```
{
    "identity": 5,
    "labels": [
        "Person"
    ],
    "properties": {
        "name": "Archie"
    },
    "elementId": "4:4fc1fbed-32b4-41d6-b07f-14e6e283ee39:5"
}
```

```
MERGE (p:Person {name: 'John'})  
RETURN p
```

13. Mencocokkan atau membuat simpul "Person" dengan properti "name" adalah "Alice" dan properti "age" adalah 25 jika belum ada dalam graph:



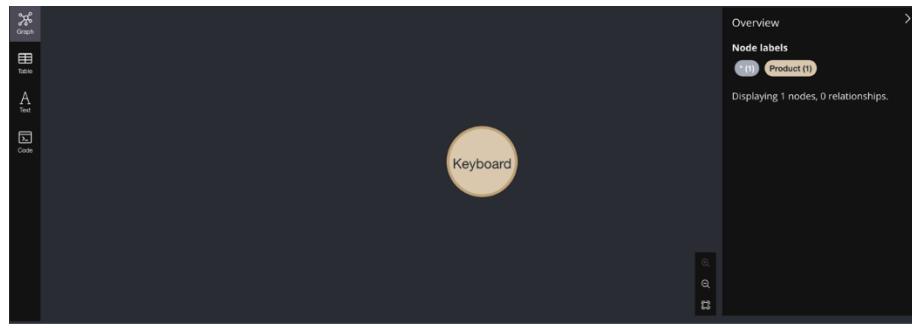
A screenshot of the Neo4j browser interface. On the left, there's a sidebar with icons for Graph, Table, Text, and Code. The main area is titled 'p' and shows a single node with the following properties:

```
{
    "identity": 0,
    "labels": [
        "Person"
    ],
    "properties": {
        "gender": "Male",
        "city": "Bandung",
        "name": "Alice",
        "age": 40
    },
    "elementId": "4:4fc1fbed-32b4-41d6-b07f-14e6e283ee39:0"
}
```

```
MERGE (p:Person {name: 'Alice'})  
ON CREATE SET p.age = 25  
RETURN p
```

14. Mencocokkan atau membuat simpul "Product" dengan properti "name" adalah "Keyboard" jika ada simpul "Product" dengan properti "price" di atas 50:

```
CREATE (p:Product {name: "Keyboard", price : 65})  
RETURN p
```



The screenshot shows the Neo4j Browser interface. On the left is a sidebar with icons for Graph, Table, Text, and Code. The main area displays a JSON object representing a node:

```

p
1
{
    "identity": 7,
    "labels": [
        "Product"
    ],
    "properties": {
        "price": 65,
        "name": "Keyboard"
    },
    "elementId": "4:4fc1fbed-32b4-41d6-b07f-14e6e283ee39:7"
}

```

Below the JSON, a status message reads: 'Added 1 label, created 1 node, set 2 properties, started streaming 1 records after 32 ms and completed after 52 ms.'

```

MERGE (p:Product {name: "Keyboard"})
WITH p
WHERE p.price > 50
SET p.brand = "Yamaha"
RETURN p;

```

The screenshot shows the Neo4j Browser interface. On the left is a sidebar with icons for Graph, Table, Text, and Code. The main area displays a JSON object representing a node:

```

p
1
{
    "identity": 7,
    "labels": [
        "Product"
    ],
    "properties": {
        "price": 65,
        "name": "Keyboard",
        "brand": "Yamaha"
    },
    "elementId": "4:4fc1fbed-32b4-41d6-b07f-14e6e283ee39:7"
}

```

Below the JSON, a status message reads: 'Set 1 property, started streaming 1 records after 6 ms and completed after 19 ms.'

15. Mencocokkan atau membuat simpul "Person" dengan properti "name" adalah "Bob". Jika simpul sudah ada, set properti "age" menjadi 30; jika tidak, atur properti "age" menjadi 25:

```

MERGE (p:Person {name: 'Bob'})
ON MATCH SET p.age = 30
ON CREATE SET p.age = 25
RETURN p

```

```

neo4j$ MERGE (p:Person {name:"Bob"}) ON MATCH SET p.age = 30 ON CREATE SET p.age = 25 RETURN p

```

```

p
1
{
  "identity": 2,
  "labels": [
    "Person"
  ],
  "properties": {
    "gender": "Male",
    "city": "Jakarta",
    "name": "Bob",
    "age": 30
  },
  "elementId": "4:4fc1fbcd-32b4-41d6-b07f-14e6e283ee39:2"
}

```

Set 1 property, started streaming 1 records after 1 ms and completed after 4 ms.

16. Temukan semua simpul "Person" yang tinggal di kota 'Jakarta' dan buat simpul "Location" untuk setiap kota yang belum ada dalam graph:

```

MATCH (p:Person)
WHERE p.city = 'Jakarta'
CREATE (:Location {city: p.city})
RETURN p

```

```

p
1
{
  "identity": 2,
  "labels": [
    "Person"
  ],
  "properties": {
    "gender": "Male",
    "city": "Jakarta",
    "name": "Bob",
    "age": 30
  },
  "elementId": "4:4fc1fbcd-32b4-41d6-b07f-14e6e283ee39:2"
}

```

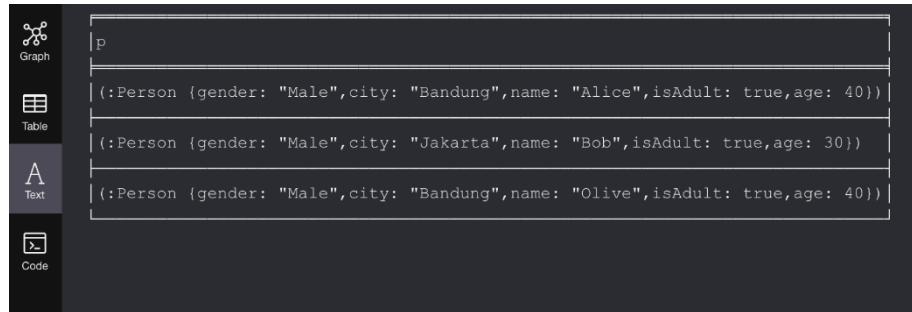
Added 1 label, created 1 node, set 1 property, started streaming 1 records after 4 ms and completed after 44 ms.

17. Temukan semua simpul "Person" yang berumur di bawah 25 tahun dan set properti "isYoung" menjadi true:

```

MATCH (p:Person)
WHERE p.age < 25
SET p.isYoung = true
RETURN p

```

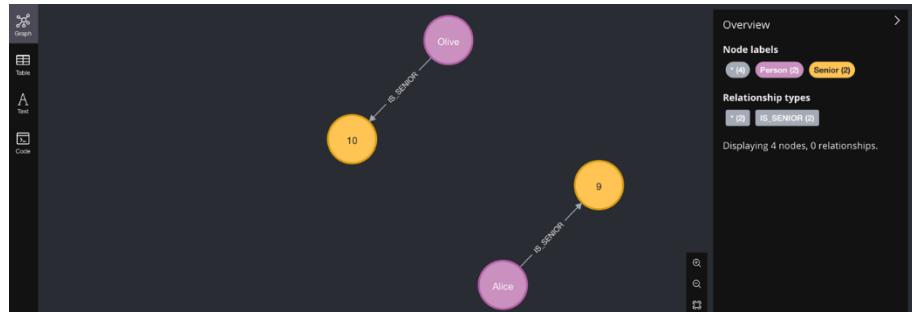
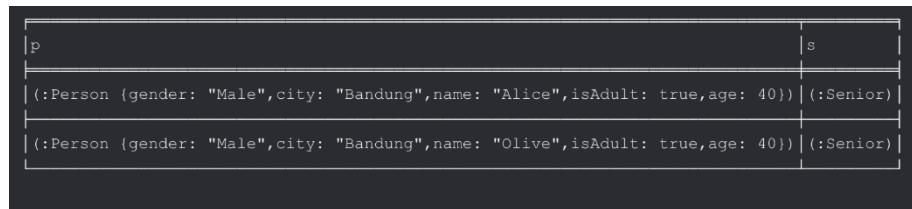


18. Temukan semua simpul "Person" yang memiliki umur di atas 30 tahun, buat simpul "Senior" untuk setiap orang, dan atur relasi "IS_SENIOR" antara mereka:

```

MATCH (p:Person)
WHERE p.age > 30
CREATE (s:Senior)
CREATE (p)-[:IS_SENIOR]->(s)
RETURN p,s

```

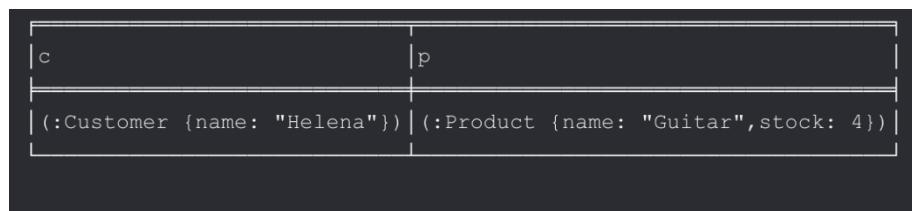


19. Buat customer lalu simulasikan operasi pembelian dengan mengurangkan stok produk setiap kali terjadi pembelian:

```

CREATE (c:Customer {name: 'Helena'})
CREATE (p:Product {name: 'Guitar', stock: 5})
CREATE (c)-[:PURCHASED]->(p)
SET p.stock = p.stock - 1
RETURN c, p;

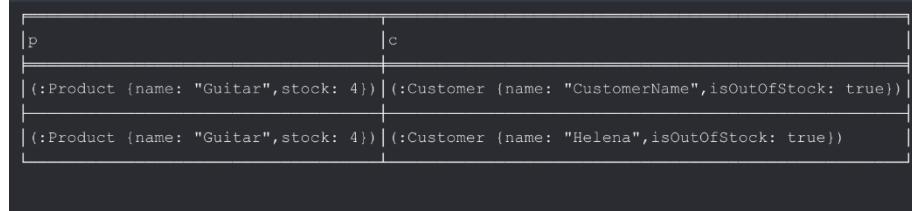
```



20. Temukan semua simpul "Product" yang memiliki stok di bawah 10, hapus simpul tersebut, dan atur properti "isOutOfStock" menjadi true pada simpul yang terhubung:

```

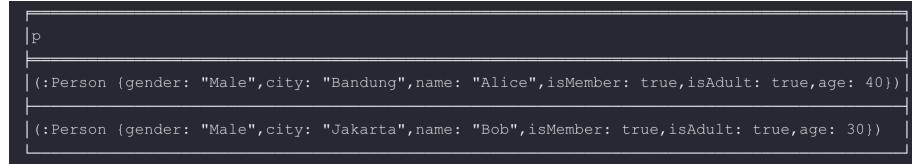
MATCH (p:Product)
WHERE p.stock < 10
OPTIONAL MATCH (p)<-[PURCHASED]->(c:Customer)
SET c.isOutOfStock = true
RETURN p
    
```



21. Temukan semua simpul "Person" yang memiliki nama 'Alice' atau 'Bob', dan atur properti "isMember" menjadi true:

```

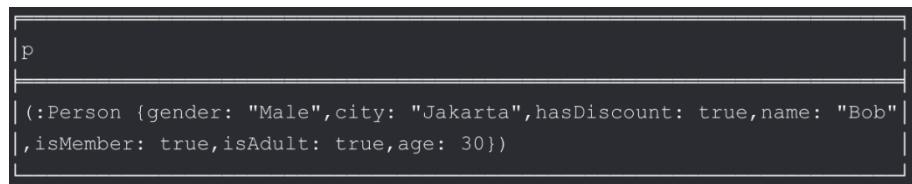
MATCH (p:Person)
WHERE p.name IN ['Alice', 'Bob']
SET p.isMember = true
RETURN p
    
```



22. Temukan semua simpul "Person" dan atur properti "hasDiscount" menjadi true untuk setiap simpul yang memiliki umur di bawah 40 tahun:

```

MATCH (p:Person)
WHERE p.age < 40
SET p.hasDiscount = true
RETURN p;
    
```



23. Temukan semua simpul "Person" yang memiliki umur di atas 40 tahun, buat simpul "Elder" untuk setiap orang, gabungkan hasilnya dengan simpul "Senior", dan atur relasi "IS_ELDER" antara simpul-simpul tersebut:

```

MATCH (p:Person)
WHERE p.age > 30
CREATE (e:Elder)
WITH p, e
MATCH (s:Senior)
CREATE (p)-[:IS_ELDER]->(e)
RETURN p, e, s;

```

| p | e | s |
|---|---|---|
| (:Person {gender: "Male",city: "Bandung",name: "Alice",isMember: true, (:Elder) (:Senior) isAdult: true,age: 40}) | | |
| (:Person {gender: "Male",city: "Bandung",name: "Alice",isMember: true, (:Elder) (:Senior) isAdult: true,age: 40}) | | |
| (:Person {gender: "Male",city: "Bandung",name: "Olive",isAdult: true,a (:Elder) (:Senior) ge: 40}) | | |
| (:Person {gender: "Male",city: "Bandung",name: "Olive",isAdult: true,a (:Elder) (:Senior) ge: 40}) | | |

10.5. Tugas dan Latihan

Buatlah Query berikut:

1. Temukan semua simpul "User" yang memiliki nama 'John' atau 'Alice' dan atur properti "isActive" menjadi true.
2. Temukan semua simpul "Person" yang memiliki umur di atas 30 tahun dan atur properti "isAdult" menjadi true pada simpul tersebut.
3. Temukan semua simpul "Movie" yang memiliki genre 'Action' atau 'Adventure', dan atur properti "isPopular" menjadi true.
4. Temukan semua simpul "Product" yang memiliki stok di atas 100 dan atur properti "isAvailable" menjadi true pada simpul tersebut.
5. Temukan semua simpul "User" yang tinggal di kota 'London' dan memiliki usia di atas 25 tahun, dan atur properti "isPremium" menjadi true.

MODUL 11

SENARAI DAN MAP DI GRAPH

11.1. Tujuan Praktikum

Mahasiswa mampu mengimplementasikan Senarai dan Map pada database Graph

11.2. Dasar Teori

11.2.1. Senarai

Dalam Graph, terdapat dua jenis tipe data yang berguna dalam penanganan data yang kompleks: Senarai (List) dan Map. Senarai (List) adalah tipe data yang digunakan untuk menyimpan kumpulan nilai dalam urutan tertentu. Setiap elemen dalam senarai dapat memiliki tipe data yang berbeda-beda. Senarai dapat digunakan untuk menyimpan nilai properti yang memiliki banyak nilai atau menyimpan hasil pencarian yang mengembalikan beberapa nilai. Contoh penggunaan:

```
CREATE (p:Person {name: 'Alice', interests: ['reading', 'music', 'travel']})
```

Dalam contoh di atas, sebuah simpul "Person" dengan properti "name" adalah 'Alice' dan properti "interests" berisi sebuah senarai yang menyimpan tiga minat: 'reading', 'music', dan 'travel'.

Senarai Nama-Nama Properti (Property Name List) adalah sebuah konsep yang memungkinkan pengguna untuk melakukan manipulasi dinamis terhadap properti-properti suatu simpul atau relasi. Gunakanlah Senarai Nama-Nama Properti ini untuk mengambil, mengubah, atau melakukan operasi lainnya pada sejumlah properti secara fleksibel.

Berikut adalah penjelasan lengkap dan dasar teori untuk Senarai Nama-Nama Properti:

1. Penggunaan Senarai Nama-Nama Properti:

- Senarai Nama-Nama Properti digunakan ini untuk menyimpan daftar nama-nama properti yang ingin dimanipulasi pada suatu simpul atau relasi.
- Senarai ini digunakan untuk melakukan operasi pada properti-properti tersebut tanpa harus secara eksplisit menyebutkan setiap nama properti secara terpisah dalam query.

2. Contoh penggunaan Senarai Nama-Nama Properti:

- Berikut adalah contoh penggunaan Senarai Nama-Nama Properti dalam Neo4j:

```
MATCH (p:Person)
SET p[{propNames}] = 'New Value'
RETURN p
```

- Dalam contoh di atas, `'{propNames}'` adalah sebuah daftar nama-nama properti yang ingin diubah nilainya menjadi 'New Value' pada simpul "Person".

3. Manfaat dari Senarai Nama-Nama Properti:

- Fleksibilitas: Dengan menggunakan Senarai Nama-Nama Properti, sejumlah properti dapat dimanipulasi dengan cara yang lebih dinamis dan fleksibel.
- Pengurangan duplikasi kode: Tidak perlu menuliskan query yang berulang-ulang jika ingin melakukan operasi pada banyak properti yang serupa. Hanya perlu mengubah Senarai Nama-Nama Properti sesuai kebutuhan.
- Skalabilitas: Dalam contoh tersebut, penggunaan Senarai Nama-Nama Properti memungkinkan penyesuaian query dengan mudah saat struktur graph berubah atau saat ingin memodifikasi properti-properti tertentu tanpa perlu melakukan perubahan query secara manual.

Dengan menggunakan Senarai Nama-Nama Properti, memungkinkan manipulasi properti-properti dalam graph Neo4j dengan efisiensi dan fleksibilitas. Hal ini sangat berguna dalam skenario di mana operasi serupa diterapkan pada banyak properti atau ketika ingin membuat query yang lebih dinamis. Penting untuk memahami konsep dan sintaksis penggunaan Senarai Nama-Nama Properti agar dapat memanfaatkannya dengan baik dalam pengembangan Graph.

11.2.2. Map

Map adalah tipe data yang digunakan untuk menyimpan pasangan kunci-nilai dalam Neo4j. Map memungkinkan Anda untuk mengorganisasi dan mengakses data dengan menggunakan kunci sebagai referensi. Contoh penggunaan Map:

```
CREATE (p:Person {name: 'Alice', properties: {age: 30, city: 'London'}})
```

Dalam contoh di atas, properti "properties" dari simpul "Person" adalah sebuah Map yang berisi pasangan kunci-nilai seperti 'age' dengan nilai 30 dan 'city' dengan nilai 'London'.

Literal Map adalah representasi literal dari Map di dalam sintaksis query. Literal Map ditulis dalam kurung kurawal {} dan menggunakan tanda titik dua (:) untuk memisahkan kunci dan nilai. Contoh penggunaan Literal Map:

```
MATCH (p:Person)
RETURN {name: p.name, age: p.age} AS personMap
```

Dalam contoh di atas, Literal Map {name: p.name, age: p.age} digunakan untuk membuat Map baru yang berisi properti "name" dan "age" dari setiap simpul "Person".

Fungsi-fungsi predikat digunakan untuk melakukan pengecekan dan pemfilteran pada data graph. Beberapa contoh fungsi predikat yang umum digunakan dalam Neo4j adalah EXISTS, ALL, ANY, dan NONE.

11.3. Software

1. Neo4j

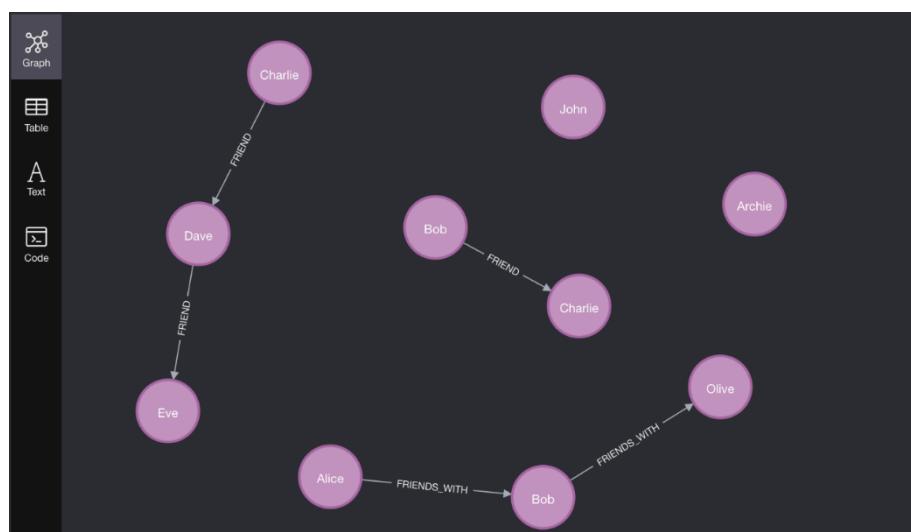
11.4. Tahapan Kerja

1. Persiapan Awal:
 - Buka browser Neo4j dan akses ke antarmuka pengguna grafis (Graphical User Interface - GUI) untuk menjalankan query.
2. Buat simpul-simpul dan hubungan-hubungan yang sesuai dengan data yang ingin digunakan dalam praktikum.

```
CREATE (p1:Person {name: 'Alice', age: 38})
CREATE (p2:Person {name: 'Bob', age: 35})
CREATE (p3:Person {name: 'Charlie', age: 28})
CREATE (p4:Person {name: 'Dave', age: 40})
CREATE (p5:Person {name: 'Eve', age: 32})

CREATE (p1)-[:FRIEND]->(p2)
CREATE (p2)-[:FRIEND]->(p3)
CREATE (p3)-[:FRIEND]->(p4)
CREATE (p4)-[:FRIEND]->(p5)
```

```
MATCH (p:Person)
RETURN p
```



3. Menggunakan Reduce(). Gunakan fungsi reduce() untuk menggabungkan elemen-elemen dalam sebuah senarai menjadi satu nilai tunggal.

```

MATCH (p:Person)
RETURN REDUCE(totalAge = 0, person IN COLLECT(p) | totalAge + person.age) AS
    .
    .

```

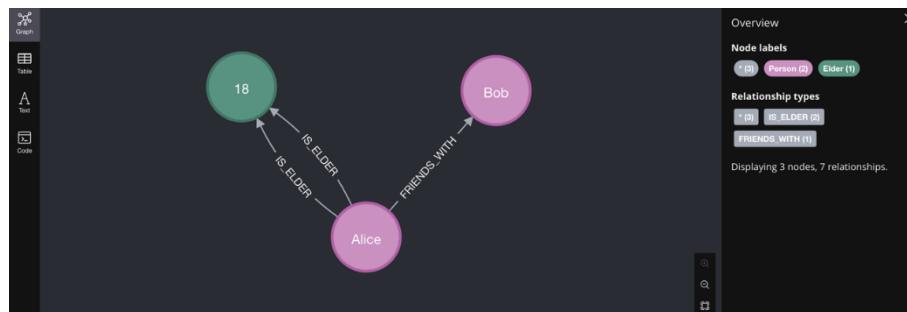
| |
|-------|
| total |
| 310 |

4. Hubungan Lintasan. Gunakan operator " $=>$ " untuk mencari apakah ada lintasan (path) yang menghubungkan dua simpul.

```

MATCH path = (:Person {name: 'Alice'})-[*]-(:Person {name: 'Bob'})
RETURN path
    .
    .

```



5. Pembalikan Urutan Senarai. Gunakan fungsi REVERSE() untuk membalikkan urutan elemen-elemen dalam sebuah senarai.

```

RETURN REVERSE(COLLECT(p.name)) AS reversedNames
    .
    .

```

| |
|---|
| reversedNames |
| ["Charlie", "Bob", "Eve", "Dave", "Charlie", "John", "Archie", "Olive"] , "Bob", "Alice"] |

6. Menggunakan size(), head(), tail(), last(). Gunakan fungsi size() untuk menghitung jumlah elemen dalam sebuah senarai. Gunakan fungsi head() untuk mendapatkan elemen pertama dalam sebuah senarai. Gunakan fungsi tail() untuk mendapatkan semua elemen kecuali elemen pertama dalam sebuah senarai. Gunakan fungsi last() untuk mendapatkan elemen terakhir dalam sebuah senarai.

```

MATCH (p:Person)
RETURN
    size(COLLECT(p)) AS totalPersons,
    head(COLLECT(p)) AS firstPerson,
    tail(COLLECT(p)) AS allButFirstPerson,
    last(COLLECT(p)) AS lastPerson;

```

| totalPersons | firstPerson | allButFirstPerson | lastPerson |
|--------------|--|---|--------------------------------------|
| 10 | (:Person {gender: "Male", city: "Bandung", name: "Alice", isMember: true, isAdult: true, age: 40}) | [(:Person {gender: "Male", city: "Jakarta", name: "Bob", isMember: true, isAdult: true, age: 30}), (:Person {gender: "Male", city: "Bandung", name: "Olive", isAdult: true, age: 40}), (:Person {name: "Archie", age: 25}), (:Person {name: "John", age: 25}), (:Person {name: "Charlie", age: 28}), (:Person {name: "Dave", age: 40}), (:Person {name: "Eve", age: 32}), (:Person {name: "Bob", age: 25}), (:Person {name: "Charlie", age: 25})] | (:Person {name: "Charlie", age: 25}) |

7. Menghitung jumlah teman dari setiap orang.

```

MATCH (p:Person)-[:FRIEND]->(friend)
RETURN p.name AS person, count(friend) AS friedCount

```

| person | friedCount |
|-----------|------------|
| "Charlie" | 1 |
| "Dave" | 1 |
| "Bob" | 1 |

Query ini mencocokkan setiap orang dengan teman-temannya dan menghitung jumlah teman yang dimiliki oleh setiap orang.

8. Menggabungkan properti tertentu dari simpul-simpul yang terhubung.

```

MATCH (p1:Person)-[:FRIEND]->(p2:Person)
RETURN p1.name AS person1, collect(p2.name) AS friends

```

| person1 | friends |
|-----------|-------------|
| "Charlie" | ["Dave"] |
| "Dave" | ["Eve"] |
| "Bob" | ["Charlie"] |

Query ini mencocokkan setiap pasangan teman dan menggabungkan nama teman-teman dari setiap orang ke dalam sebuah senarai.

9. Menemukan usia tertua.

```
MATCH (p:Person)
RETURN max(p.age) AS maxAge
```

| |
|--------|
| maxAge |
| 40 |

Query ini mencari orang dengan usia tertua di antara semua orang yang ada dalam graph.

10. Mengurutkan orang dari usia tertua.

```
MATCH (p:Person)
WITH p
ORDER BY p.age DESC
RETURN p.name AS oldestPerson, p.age AS maxAge;
```

| oldestPerson | maxAge |
|--------------|--------|
| "Alice" | 40 |
| "Olive" | 40 |
| "Dave" | 40 |
| "Eve" | 32 |
| "Bob" | 30 |
| "Charlie" | 28 |
| "Archie" | 25 |
| "John" | 25 |

Query ini mengurutkan person berdasarkan usia tertua orang yang ada dalam graph.

11. Menghitung jumlah teman yang unik dari setiap orang.

```
MATCH (p:Person)-[:FRIEND]->(friend)
RETURN p.name AS person, count(DISTINCT friend) AS uniqueFriendCount
```

| person | uniqueFriendCount |
|-----------|-------------------|
| "Charlie" | 1 |
| "Dave" | 1 |
| "Bob" | 1 |

Query ini mencocokkan setiap orang dengan teman-temannya dan menghitung jumlah teman unik yang dimiliki oleh setiap orang.

12. Menghitung jumlah orang dengan usia tertentu.

```
MATCH (p:Person)
RETURN p.age AS age, count(p) AS count
```

| age | count |
|-----|-------|
| 40 | 3 |
| 30 | 1 |
| 25 | 4 |
| 28 | 1 |
| 32 | 1 |

Query ini menghitung jumlah orang yang memiliki usia tertentu dalam graph.

13. Menghitung rata-rata usia dari semua orang

```
MATCH (p:Person)
RETURN avg(p.age) AS averageAge
```

| averageAge |
|------------|
| 31.0 |

Query ini menghitung rata-rata usia dari semua orang yang ada dalam graph.

14. Menemukan orang-orang dengan usia di atas rata-rata.

```

MATCH (p:Person)
WITH avg(p.age) AS averageAge
MATCH (person:Person)
WHERE person.age > averageAge
RETURN person.name AS person

```

| person |
|---------|
| "Alice" |
| "Olive" |
| "Dave" |
| "Eve" |

Query ini mencari orang-orang yang memiliki usia di atas rata-rata usia dari semua orang dalam graph.

- Menghitung jumlah teman bersama antara dua orang.

```

MATCH (p1:Person {name: 'Alice'})-[:FRIEND]->(friend)<&[:FRIEND]-(p2:Person
{name: 'Bob'})
RETURN count(FRIEND) AS commonFriendsCount

```

| commonFriendsCount |
|--------------------|
| 0 |

Query ini mencocokkan orang Alice dan Bob, lalu menghitung jumlah teman bersama yang dimiliki oleh keduanya.

- Buat hobby untuk tiap person. Sebagai contoh Alice

```

MATCH (p:Person {name: 'Alice'})
SET p.hobbies = 'Reading'
RETURN p;

```

| |
|--|
| p |
| {:Person {gender: "Male",hobbies: "Reading",city: "Bandung",name: "Alice",isMember: true,isAdult: true,age: 40)} |

Lakukan untuk person yang lain.

17. Menggunakan fungsi all(). Gunakan fungsi all() untuk memeriksa apakah semua elemen dalam sebuah senarai memenuhi kondisi tertentu.

```
MATCH (p:Person)
WHERE all(hobby IN p.hobbies WHERE hobby = 'Reading')
RETURN p.name AS personName
```

| personName |
|------------|
| "Alice" |
| "Olive" |
| "Charlie" |
| "Charlie" |

Query ini mencocokkan orang-orang yang memiliki semua hobi "Reading".

18. Menggunakan fungsi any(). Gunakan fungsi any() untuk memeriksa apakah setidaknya satu elemen dalam sebuah senarai memenuhi kondisi tertentu.

```
MATCH (p:Person)
WHERE any(hobby IN p.hobbies WHERE hobby = 'Gardening')
RETURN p.name AS personName
```

| personName |
|------------|
| "Archie" |
| "John" |
| "Dave" |
| "Eve" |

Query ini mencocokkan orang-orang yang memiliki setidaknya satu hobi "Gardening".

19. Menggunakan fungsi exists(). Gunakan fungsi exists() untuk memeriksa apakah ada setidaknya satu simpul yang memenuhi kondisi tertentu.

```
MATCH (p:Person)
WHERE p.age IS NOT NULL
RETURN p.name AS personName;
```

| |
|------------|
| personName |
| "Alice" |
| "Bob" |
| "Olive" |
| "Archie" |
| "John" |
| "Charlie" |
| "Dave" |

Query ini mencocokkan orang-orang yang memiliki properti usia (age) yang ada.

20. Menggunakan fungsi none(). Gunakan fungsi none() untuk memeriksa apakah tidak ada elemen dalam sebuah senarai yang memenuhi kondisi tertentu.

```
MATCH (p:Person)
WHERE none(hobby IN p.hobbies WHERE hobby = 'Sports')
RETURN p.name AS personName
```

| |
|------------|
| personName |
| "Alice" |
| "Bob" |
| "Olive" |
| "Archie" |

Query ini mencocokkan orang-orang yang tidak memiliki hobi "Sports".

21. Menggunakan fungsi single(). Gunakan fungsi single() untuk memeriksa apakah hanya ada satu elemen dalam sebuah senarai yang memenuhi kondisi tertentu.

```

MATCH (p:Person)
WHERE Single(hobby IN p.hobbies WHERE hobby = 'Swimming')
RETURN p.name AS personName

```

| |
|------------|
| personName |
| "Bob" |
| "Archie" |
| "Dave" |
| "Eve" |
| "Bob" |

Query ini mencocokkan orang-orang yang hanya memiliki satu hobi yaitu "Swimming".

11.5. Tugas dan Latihan

Buatlah Query berikut:

1. Tuliskan query untuk menampilkan semua simpul yang memiliki hubungan keluarga (misalnya, relasi "Orang tua", "Anak", "Saudara") dengan simpul tertentu.
2. Buat query untuk menghitung jumlah total hubungan yang dimiliki oleh simpul tertentu.
3. Tuliskan query untuk menemukan simpul dengan tingkat kedekatan tertentu dengan simpul sumber, dihitung berdasarkan jumlah tingkat hubungan yang terhubung.
4. Buat query untuk mencari jalur terpendek antara dua simpul dalam graph.
5. Tuliskan query untuk mencari simpul yang memiliki properti tertentu dan melakukan pengurutan berdasarkan nilai properti tersebut.
6. Buat query untuk menemukan simpul dengan hubungan terbanyak di antara simpul-simpul lain dalam graph.
7. Tuliskan query untuk menghapus simpul beserta seluruh hubungannya dengan simpul lain berdasarkan kondisi tertentu.
8. Buat query untuk menghitung rata-rata nilai dari properti numerik pada simpul-simpul yang memenuhi kriteria tertentu.
9. Tuliskan query untuk menggabungkan hasil dua query terpisah menggunakan operator UNION.

10. Buat query untuk menggabungkan data dari beberapa sumber yang berbeda menjadi satu graph.

MODUL 12

FUNGSI-FUNGSI AGREGAT

12.1. Tujuan Praktikum

Mahasiswa mampu mengimplementasikan fungsi-fungsi agregat pada database Graph

12.2. Dasar Teori

12.2.1. Fungsi agregat

Fungsi-fungsi agregat dalam database graph, seperti COUNT(), SUM(), AVG(), MAX(), MIN(), pengelompokan data (STDEV(), STDEVPOP(), STDEVSAMP()) adalah alat penting untuk melakukan perhitungan statistik dan analisis data. Berikut adalah penjelasan teori tentang fungsi-fungsi agregat tersebut:

1. COUNT(): Fungsi COUNT() digunakan untuk menghitung jumlah baris atau elemen yang memenuhi kondisi tertentu dalam dataset. Misalnya, COUNT() dapat digunakan untuk menghitung jumlah simpul atau hubungan dalam graph, atau jumlah entri dalam suatu kolom.
2. SUM(): Fungsi SUM() digunakan untuk menjumlahkan nilai-nilai numerik dalam dataset. Biasanya digunakan untuk menjumlahkan nilai properti numerik pada simpul atau hubungan dalam graph.
3. AVG(): Fungsi AVG() digunakan untuk menghitung rata-rata nilai numerik dalam dataset. Dengan menggunakan AVG(), Anda dapat menghitung rata-rata nilai properti numerik pada simpul atau hubungan dalam graph.
4. MAX(): Fungsi MAX() digunakan untuk mencari nilai maksimum dalam dataset. Misalnya, MAX() dapat digunakan untuk mencari nilai maksimum dari suatu properti numerik pada simpul atau hubungan dalam graph.
5. MIN(): Fungsi MIN() digunakan untuk mencari nilai minimum dalam dataset. Contohnya, MIN() dapat digunakan untuk mencari nilai minimum dari suatu properti numerik pada simpul atau hubungan dalam graph.
6. Pengelompokan data (standard deviation): Pengelompokan data atau deviasi standar adalah ukuran statistik yang mengukur sejauh mana nilai-nilai dalam dataset tersebar secara merata di sekitar nilai rata-rata. Pengelompokan data memberikan informasi tentang variabilitas data. Biasanya digunakan untuk mengukur sejauh mana nilai properti numerik pada simpul atau hubungan dalam graph berbeda dari rata-rata.

12.2.2. Persentil

Persentil dalam graph mengacu pada nilai-nilai yang membagi data ke dalam persentase tertentu. Persentil digunakan untuk mengukur posisi relatif suatu nilai dalam kumpulan data.

Secara umum, persentil dibagi menjadi 100 bagian yang setiap bagianya mewakili persentase 1% dari data. Misalnya, persentil ke-25 (P25) merupakan nilai di mana 25% data berada di bawahnya, sedangkan persentil ke-75 (P75) merupakan nilai di mana 75% data berada di bawahnya.

Dalam konteks graph, persentil dapat digunakan untuk menganalisis distribusi data pada simpul atau hubungan. Misalnya, jika Anda memiliki data yang mewakili tingkat gaji karyawan dalam suatu perusahaan, Anda dapat menggunakan persentil untuk mengetahui di mana posisi gaji individu tertentu berada relatif terhadap seluruh karyawan. Dengan demikian, persentil membantu Anda memahami distribusi data gaji dan melihat sejauh mana suatu nilai gaji berada dalam konteks perusahaan tersebut.

Dalam Neo4j, Anda dapat menggunakan fungsi-fungsi agregat seperti PERCENTILE_CONT() atau PERCENTILE_DISC() untuk menghitung persentil pada data dalam graph. Fungsi PERCENTILE_CONT() menghasilkan interpolasi kontinu dari persentil, sedangkan fungsi PERCENTILE_DISC() menghasilkan nilai diskrit yang sama dengan persentil yang ditentukan.

Dengan menggunakan persentil dalam analisis data graph, Anda dapat mendapatkan wawasan yang lebih dalam tentang distribusi data, mengidentifikasi outliers, membandingkan posisi relatif suatu nilai dalam dataset, dan melakukan analisis statistik yang lebih komprehensif.

12.3. Software

1. Neo4j

12.4. Tahapan Kerja

1. Persiapan Awal:
 - Buka browser Neo4j dan akses ke antarmuka pengguna grafis (Graphical User Interface - GUI) untuk menjalankan query.
2. Buat data berikut:

```

// Membuat simpul Product
CREATE (:Product {id: 'P001', name: 'Product 1', price: 10})
CREATE (:Product {id: 'P002', name: 'Product 2', price: 20})
CREATE (:Product {id: 'P003', name: 'Product 3', price: 15})
CREATE (:Product {id: 'P004', name: 'Product 4', price: 25})
CREATE (:Product {id: 'P005', name: 'Product 5', price: 30})

// Membuat simpul Seller
CREATE (:Seller {id: 'S001', name: 'Seller 1'})
CREATE (:Seller {id: 'S002', name: 'Seller 2'})
CREATE (:Seller {id: 'S003', name: 'Seller 3'})
CREATE (:Seller {id: 'S004', name: 'Seller 4'})
CREATE (:Seller {id: 'S005', name: 'Seller 5'})

```

```

MATCH (s:Seller {id: 'S001'}), (p:Product {id: 'P001'})
CREATE (s)-[:SOLD_BY]->(:Sale {id: 'SA001', quantity: 5})-[:SOLD_PRODUCT]->(p)
WITH 1 AS dummy

MATCH (s:Seller {id: 'S002'}), (p:Product {id: 'P002'})
CREATE (s)-[:SOLD_BY]->(:Sale {id: 'SA002', quantity: 10})-[:SOLD_PRODUCT]->(p)
WITH 1 AS dummy

MATCH (s:Seller {id: 'S003'}), (p:Product {id: 'P003'})
CREATE (s)-[:SOLD_BY]->(:Sale {id: 'SA003', quantity: 8})-[:SOLD_PRODUCT]->(p)
WITH 1 AS dummy

MATCH (s:Seller {id: 'S004'}), (p:Product {id: 'P004'})
CREATE (s)-[:SOLD_BY]->(:Sale {id: 'SA004', quantity: 12})-[:SOLD_PRODUCT]->(p)
WITH 1 AS dummy

MATCH (s:Seller {id: 'S005'}), (p:Product {id: 'P005'})
CREATE (s)-[:SOLD_BY]->(:Sale {id: 'SA005', quantity: 15})-[:SOLD_PRODUCT]->(p);

```

3. Menggunakan fungsi COUNT():

Menampilkan jumlah total penjualan

```

MATCH (:Sale)
RETURN COUNT(*) AS totalSales

```

| |
|------------|
| totalSales |
| |
| 5 |

COUNT() dengan kondisi

```
MATCH (p:Product)
WHERE p.price > 10
RETURN COUNT(p.price) AS totalProductPrice
```

| |
|-------------------|
| totalProductPrice |
| 4 |

4. Menggunakan fungsi SUM()

```
MATCH (p:Product)
RETURN SUM(p.price) AS totalSum
```

| |
|----------|
| totalSum |
| 100 |

5. Menggunakan fungsi AVG()

```
MATCH (p:Product)
RETURN AVG(p.price) AS averageValue
```

| |
|--------------|
| averageValue |
| 20.0 |

6. Menggunakan fungsi MIN()

```
MATCH (p:Product)
RETURN MIN(p.price) AS minValue
```

| |
|----------|
| minValue |
| 10 |

7. Menggunakan fungsi MAX()

```
MATCH (p:Product)
RETURN MAX(p.price) AS maxValue
```

| |
|----------|
| maxValue |
| 30 |

8. Menggunakan fungsi deviasi standar

```
MATCH (p:Product)
WITH stdev(p.price) AS standardDeviation
RETURN standardDeviation
```

| |
|-------------------|
| standardDeviation |
| 7.905694150420948 |

9. Menggunakan fungsi persentil

```
MATCH (p:Product)
RETURN percentileCont(p.price, 0.25) AS P25,
       percentileCont(p.price, 0.50) AS P50,
       percentileCont(p.price, 0.75) AS P75
```

| | | |
|-----|-----|-----|
| P25 | P50 | P75 |
| 15 | 20 | 25 |

10. Menggunakan pengelompokan data

```
MATCH (p:Product)
RETURN p.name, COUNT(p) AS countPerCategory
```

| p.name | countPerCategory |
|-------------|------------------|
| "Product 1" | 1 |
| "Product 2" | 1 |
| "Product 3" | 1 |
| "Product 4" | 1 |
| "Product 5" | 1 |

12.5. Tugas dan Latihan

Buatlah Query berikut:

1. Tuliskan query untuk menampilkan semua simpul yang memiliki hubungan keluarga (misalnya, relasi "Orang tua", "Anak", "Saudara") dengan simpul tertentu.
2. Buat query untuk menghitung jumlah total hubungan yang dimiliki oleh simpul tertentu.
3. Tuliskan query untuk menemukan simpul dengan tingkat kedekatan tertentu dengan simpul sumber, dihitung berdasarkan jumlah tingkat hubungan yang terhubung.
4. Buat query untuk mencari jalur terpendek antara dua simpul dalam graph.
5. Tuliskan query untuk mencari simpul yang memiliki properti tertentu dan melakukan pengurutan berdasarkan nilai properti tersebut.
6. Buat query untuk menemukan simpul dengan hubungan terbanyak di antara simpul-simpul lain dalam graph.
7. Tuliskan query untuk menghapus simpul beserta seluruh hubungannya dengan simpul lain berdasarkan kondisi tertentu.
8. Buat query untuk menghitung rata-rata nilai dari properti numerik pada simpul-simpul yang memenuhi kriteria tertentu.
9. Tuliskan query untuk menggabungkan hasil dua query terpisah menggunakan operator UNION.
10. Buat query untuk menggabungkan data dari beberapa sumber yang berbeda menjadi satu graph.

DAFTAR PUSTAKA

- Banker, Kayle., dkk. 2016. MongoDB In Action : Second Edition. Manning Publication: Shelter Island.
- Robinson, Ian., dkk. 2015. Graph Databases: Second Edition. O'Reilly Media, Inc.
- Stack Overflow Contributors. 2021. Learning MongoDB. Stack Overflow.
- Silberschatz, Avi., dkk. 2019. Database System Concepts: Seventh Edition. McGraw-Hill.
- Suherman. 2010. Introducing SQL Server 2008 Feature : Mirroring. Microsoft User Group Indonesia.
- Tutorial Points. 2018. MongoDB: nosql document database. Tutorial Points.

Website :

- <https://neo4j.com/docs/>
- <https://roadmap.sh/mongodb>
- <https://www.mongodb.com/docs/>



INSTITUT BISNIS DAN INFORMATIKA KESATUAN

Jl. Rangga Gading No.01, Gudang, Kecamatan
Bogor Tengah, Kota Bogor, Jawa Barat 16123