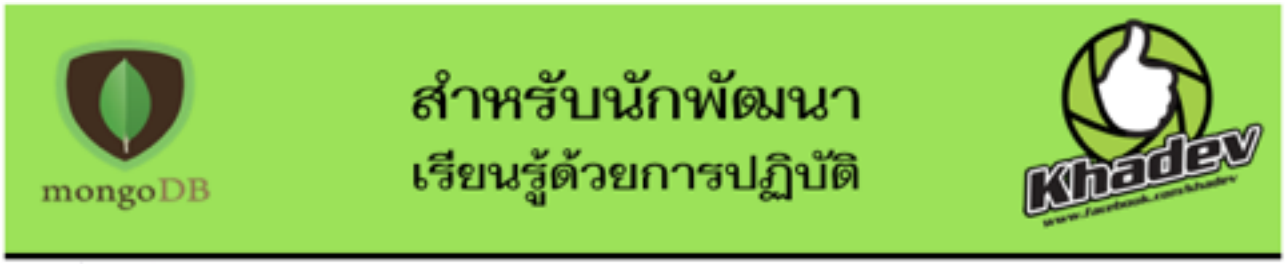
mongoDB

สำหรับนักพัฒนาแอพพลิเคชั่น เน้นฝึกปฏิบัติ

KhaDev



บทที่ 1 พื้นฐาน MongoDB

MongoDB คืออะไร ?

MongoDB คือ ฐานข้อมูลแบบ document-oriented database พัฒนาด้วยภาษา C++ เป็นฐานข้อมูลประเภท * NOSQL ซึ่งมีความแตกต่างจากฐานข้อมูลเชิงสัมพันธ์ (Relational database) ที่นักพัฒนาซอฟท์แวร์ทั่วไปรู้จักและใช้กันมาอย่างยาวนาน ฐานข้อมูลแบบ NOSQL นั้นถูกสร้างขึ้นโดยวัตถุประสงค์หลักเพื่อรองรับการขยายการจัดเก็บข้อมูลที่นับวันจะมีขนาดใหญ่ปริมาณมหาศาลได้ง่ายขึ้น รวดเร็ว รองรับโครงสร้างที่ไม่ตายตัว มีการเปลี่ยนแปลงอยู่ตลอดได้ดี งานพัฒนาซอฟท์แวร์เป็นงานที่ต้องปรับปรุงเปลี่ยนแปลงอยู่ตลอด ฐานข้อมูลประเภท NOSQL จึงมีความเหมาะสมกับการรองรับความเปลี่ยนแปลงอยู่เสมอ ฐานข้อมูลประเภท NOSQL นั้นได้เติบโตอย่างรวดเร็วและแย่งส่วนแบ่งการตลาดของฐานข้อมูลเชิงสัมพันธ์ไปได้มากทีเดียว แต่ไม่ได้หมายความว่า NOSQL จะมาแทนที่ Relational Database ได้ทั้งหมด แต่ก็ต้องยอมรับว่าบางส่วนก็ถูกทดแทนได้เช่นกัน Relational Database มีการใช้งานอย่างแพร่หลาย มีการพัฒนาอยู่ตลอดและใช้ในโครงการใหญ่ๆมากมาย และมีข้อดีในการประมวลผลกับข้อมูลปริมาณมากๆ อย่างเช่นฟังก์ชั่นประเภท Aggregate ฟังก์ชั่นได้ดีกว่า NOSQL หลายๆตัว นั่นคือเหตุผลคร่าวๆว่าไม่ใช่งานง่ายที่ NOSQL จะมาแทนที่ Relational Database แต่แน่นอนว่า NOSQL ก็กินส่วนแบ่งจาก Relational Database และเป็นคู่แข่งกันอย่างหลีกเลี่ยงไม่ได้เพราะหน้าที่ของฐานข้อมูลก็คือจัดเก็บข้อมูล นอกจากนั้นยังมีออกมาหลายตัวหลายแบบเลือกใช้ไม่ถูกกันเลยทีเดียว ที่สำคัญส่วนใหญ่ฟรี NOSQL จึงเป็นของชอบสำหรับบริษัทที่ต้องการลดต้นทุน เพราะนอกจากฟรีแล้วยังสามารถรองรับการขยายตัวในอนาคตได้ดีไม่แพ้ Relational Database ตัวดังๆ สำหรับ MongoDB นั้นสามารถนำมาใช้งานได้ฟรี และมีความโดดเด่นในเรื่อง Community ที่มีขนาดใหญ่ คนในวงการ NOSQL รู้จักเป็นอย่างดี มีการใช้ในโครงการระดับโลกมากมาย หากนักพัฒนาแอพพลิเคชั่น ยังไม่รู้จักฐานข้อมูลตัวนี้เรียกได้ว่าท่านยังไม่ได้ก้าวเข้ามาทำความรู้จักโลกของ NOSQL เลยทีเดียว

ทำไมต้องใช้ MongoDB ?

ปัจจุบันต้นทุนของการพัฒนาซอฟท์แวร์มีราคาต่ำลงกว่าเมื่อก่อนมาก เพราะเทคโนโลยีต่างๆเปิดซอร์สโค้ดให้มีโอกาสศึกษาและแบ่งปันกันมากขึ้น ทำให้โลกของเทคโนโลยีแบบ Opensource เติบโตและมีประสิทธิภาพมากขึ้น แน่นอนว่าองค์กรทุกองค์กรที่จัดเก็บข้อมูลย่อมมีการใช้ฐานข้อมูล ซึ่งในยุคก่อนมีค่าใช้จ่ายของซอฟท์แวร์ในราคาที่สูงมาก ทั้งค่าซอฟท์แวร์ ฮาร์ดแวร์ บุคลากรที่ดูแล ในโลกของ Opensource ได้ทำให้ราคาซอฟท์แวร์ และการใช้ร่วมกับฮาร์ดแวร์ลดลงอย่างมาก MongoDB ก็เป็นทางเลือกที่ยอดเยี่ยมสำหรับองค์กรใหญ่ที่อยากจะลดต้นทุนค่าใช้จ่ายของซอฟท์แวร์ลง และยังได้รับการพิสูจน์มาแล้วกับโครงการใหญ่ๆระดับโลกว่ามีประสิทธิภาพที่ดีสามารถใช้งานทดแทนซอฟท์แวร์ราคาแพงได้สบายๆ และมีการใช้งานค่อนข้างง่ายเมื่อเทียบกับซอฟท์แวร์ราคาแพงๆเหล่านั้น

รูปแบบการเก็บข้อมูลของ MongoDB

MongoDB เก็บข้อมูลในรูปแบบที่เรียกว่า BSON (binary-encoded serialization of JSON) ซึ่งเป็นเทคนิคในการนำข้อมูลในรูปแบบ JSON มาแปลงเป็น Binary ที่คอมพิวเตอร์อ่านเข้าใจ ส่วนรูปแบบ JSON (JavaScript Object Notation) ที่เราใช้จัดเก็บและอ่านเข้าใจได้ สำหรับนักพัฒนาเว็บไซต์มีความคุ้นเคยกันดีอยู่แล้ว ดังนี้

```
// JSON
{ "hello" : "world" }

// BSON
"\x16\x00\x00\x00\x02hello\x00
\x06\x00\x00\x00world\x00\x00"
```

- ขอบเขตของเอกสารหนึ่งชุด จะอยู่ภายใต้เครื่องหมาย Bracket {} (วงเล็บปีกกา)
- ชื่อฟิลด์ หรือ คอลัมน์ หรือ property จะสิ้นสุดด้วยเครื่องหมาย : (โคลอน) เช่น id:
- ค่าของฟิลด์นั้นๆ จะอยู่หลังจากเครื่องหมาย : ซึ่งจะหากเป็นข้อมูลประเภทข้อความจะต้องอยู่ภายใต้เครื่องหมาย “ หรือ ‘ อย่างใดอย่างหนึ่งเท่านั้น แต่ถ้าเป็นตัวเลข หรือ ค่าตรรกะ จริง เท็จ ไม่ต้องมีเครื่องหมายใดๆครอบ เช่น

```
firstName:"AAA" , lastName:'BBB' , age:30 , isResign:false
```

โครงสร้างการจัดเก็บข้อมูลของ MongoDB

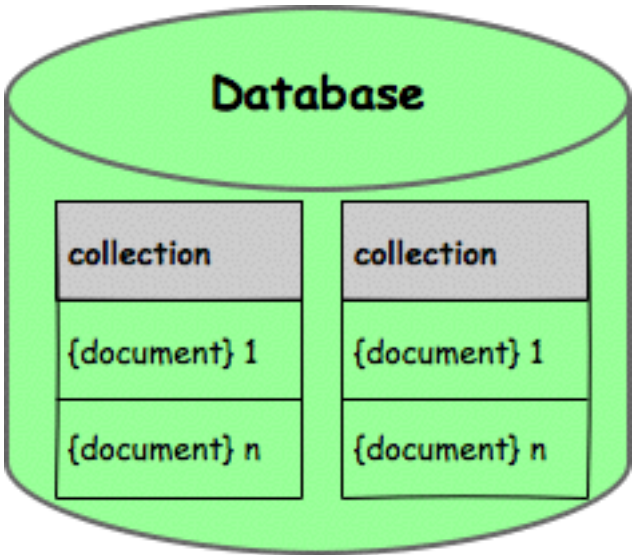
MongoDB แบ่งการจัดเก็บโครงสร้างของข้อมูลหลักๆเป็น 3 ระดับ คือ

- **Database**
  กลุ่มของชุดข้อมูลที่ได้จัดหมวดหมู่ไว้ เช่น ฐานข้อมูลระบบ Chat , ฐานข้อมูลระบบจัดการพนักงาน , ฐานข้อมูลระบบงานขาย เป็นต้น ในหนึ่งบริการของ MongoDB สามารถมีได้ หลายๆฐานข้อมูล
- **Collection**
  ชุดข้อมูลหากเปรียบเทียบกับฐานข้อมูลแบบ Relational ก็คือตาราง ซึ่งจะเป็นชุดข้อมูลที่เก็บเอกสารในกลุ่มข้อมูลเรื่องนั้นๆ เช่น Collection ของ สินค้า (Product) ก็ใช้เก็บข้อมูลเอกสาร (document) ของสินค้า สำหรับ Collection ใน MongoDB นั้นมีความยืดหยุ่นมาก ไม่จำเป็นต้องสร้างขึ้นมาก่อนเหมือนกับในฐานข้อมูลแบบRelational เลยก็ได้ เราสามารถเขียนคำสั่ง insert ข้อมูลลงใน Collection ที่ไม่มีอยู่เลยก็ได้ MongoDB ก็จะทำการสร้าง Collection และข้อมูลให้อัตโนมัต การสร้าง Collection อัตโนมัติ มีข้อเสีย หากถูกส่งข้อมูลโจมตีมาจาก

ระบบอื่น จะทำให้ข้อมูล ที่เราไม่ต้องการถูกสร้างขึ้น ป้องกันเบื้องต้นได้โดยกำหนด option เป็นแบบ strict

- **Document**

  ข้อมูลแต่ละรายการใน Collection นั้นๆ ซึ่งเปรียบเทียบได้กับ Row ในฐานข้อมูลแบบ Relational แต่สิ่งที่ต่างกันคือในแต่ละ Documemt สามารถมีโครงสร้างชื่อฟิลด์หรือคอลัมน์ไม่เหมือนกันเลยก็ได้ ต่างกับ Relational Database ที่ต้องมีชื่อคอลัมน์เหมือนกันในทุกรายการ



รูปที่ 1-1 โครงสร้างการจัดเก็บข้อูลของ MongoDB

ตารางเปรียบเทียบกับ Relational Database

#	Relational Database	MongoDB
1	Database	Database
2	Table	Collection
3	Row	Document
4	Column	Field
5	Table Join	Embedded ,
6	Primary Key	Primary Key (Default _id)

ตารางที่ 1-1 ตารางเปรียบเทียบ Relational กับ Mongo DB

ชนิดข้อมูลของ MongoDB

#	ชนิด	รายละเอียด	ตัวอย่าง
1	String	ชนิดข้อมูลตัวอักษรเป็นชนิดของข้อมูลที่มีการเก็บ ในฐานข้อมูลและใช้งานมากที่สุดที่ภาษาโปรแกรมมิ่ง mongodb ต้องเก็บเป็น encoding แบบ UTF-8	`{"data":"Hello MongoDB"}`
2	Integer	ชนิดข้อมูลแบบตัวเลขแบบจำนวนจริง สามารถเก็บได้แบบ 32 บิต (–2147483648 ถึง 2147483647) หรือ 64 บิต (-9223372036854775808 - ถึง 9223372036854775807) ขึ้นอยู่กับเครื่องเซิร์ฟเวอร์ที่ติดตั้ง mongodb รองรับ	`{"data":999}`
3	Boolean	ชนิดข้อมูลแบบตรรกศาสตร์ true / false	`{"data":true}`
4	Double	ชนิดข้อมูลตัวเลขแบบทศนิยม -1.7976931348623157E+308 ถึง 1.7976931348623157E+308	`{"data":3.14}`
5	Min / Max keys	ชนิดข้อมูลนี้ใช้เปรียบเทียบค่าน้อยที่สุดถึงค่าสูงสุด	
	Arrays	ชนิดข้อมูลแบบหลายค่า หรือเก็บเป็นชุดข้อมูล	`{"data":[0,1,2,3,4,5,6,7]}` `{"data":[{"id":1,"name":"A"},{"id":2,"name":"B"}]}`
6	Timestamp	ชนิดข้อมูลเก็บเวลาเมื่อทำการเพิ่มข้อมูลหรือแก้ไข	`{"data":Timestamp(6307772095426199, 1)}`

#	ชนิด	รายละเอียด	ตัวอย่าง
7	Object	ชนิดข้อมูลแบบ embedded documents คือชุดข้อมูลแบบ json ซ้อนกันในฟิลด์	`{"data": {"address":"123/1","city":"A" , "state":"B" ,"country":"C"}}`
8	Null	ชนิดข้อมูลสำหรับเก็บค่า Null	
9	Date	ชนิดข้อมูลวันที่ตามรูปแบบของระบบปฏิบัติการ Unix	`{"data" : ISODate("2016-07-16T07:56:30.966Z")}`
10	Object ID	ชนิดข้อมูลที่ mongoldb สร้างขึ้นให้อัตโนมัติในฟิลด์ชื่อ _id เพื่อเป็นคีย์หลักหรือจะระบุค่าเองก็ได้ แต่ต้องเป็นค่าที่ไม่ซ้ำกันในcollection นั้นๆ โดยค่าที่ระบุจะไม่สามารถแก้ไขได้	`{"_id" : ObjectId("5789bb03c502ffc637ba8d79"),"data" : "©"}`
11	Binary data	ชนิดข้อมูลแบบ Binary ส่วนมากใช้เก็บไฟล์	
12	Code	ชนิดข้อมูลแบบโค้ดหรือชุดคำสั่งในรูปแบบภาษาจาวาสคริปต์	
13	Regular expression	ชนิดข้อมูลแบบ Regular expression ไว้สำหรับตรวจสอบรูปแบบข้อมูล เช่น เบอร์โทรศัพท์	`{ mobileNo:{ $regex: /^\(\d\d\d\)\d\d\d-\d\d\d\d$/ }}`

แบบฝึกหัด 1-1 : ติดตั้ง MongoDB

วัตถุประสงค์
เพื่อติดตั้ง mongodb และกำหนดค่าเพื่อใช้งานเบื้องต้นได้

แนวทางปฏิบัติ

- ดาวน์โหลดซอฟท์แวร์จากเว็บ https://www.mongodb.com/download-center#community
- ดูขั้นตอนการติดตั้งในวิธีการติดตั้งของระบบปฏิบัติการต่างๆ

วิธีการติดตั้งสำหรับ Mac

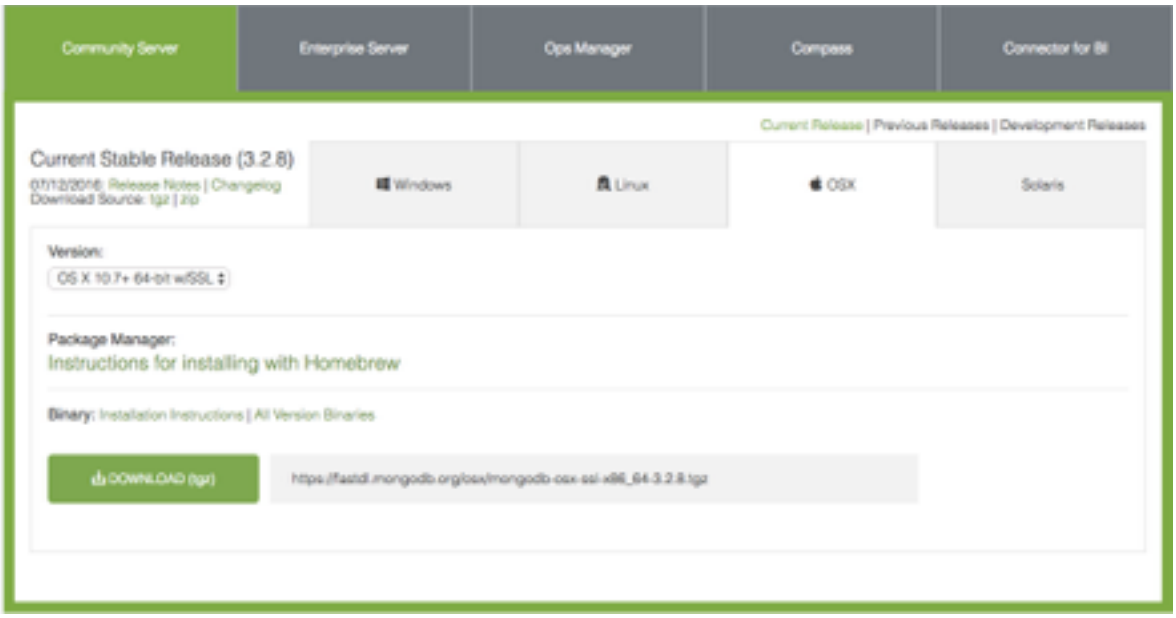
ติดตั้ง mongo ใน system shell

1. เปิดโปรแกรม terminal พิมพ์คำสั่ง

```
brew install mongodb
```

ติดตั้งโดยใช้ไฟล์แพคเกจ

1. ดาวโหลดไฟล์แพคเกจได้ที่ https://www.mongodb.com/download-center#community
2. คลิกปุ่ม DOWNLOAD



รูปที่ 1-1.1 ไฟล์แพกเกจสำหรับติดตั้ง mongodb ในระบบปฏิบัติการต่างๆ

3. ทำการแตกไฟล์ .tar ที่ได้ดาวน์โหลดมา พิมพ์คำสั่ง ดังนี้

```
tar -zxvf mongodb-osx-ssl-x86_64-3.2.8.tgz
```

* *mongodb-osx-ssl-x86_64-3.2.8.tgz* ให้ใช้ชื่อที่ได้ดาวน์โหลดมาและระบุไดเรคทอรี่ที่ไฟล์นั้นวางอยู่จริงให้ถูกต้อง

```
mkdir -p <mongodata directory>
```

4. ทำคัดลอกไฟล์ที่ทำการแตกออกมาแล้วไปวางในไดเรคทอรี่ที่ต้องการเก็บ mongodb พิมพ์คำสั่งดังนี้

```
mkdir -p mongodb
cp -R -n mongodb-osx-ssl-x86_64-3.2.8/ mongodb
```

5. ทำการกำหนดตำแหน่งของพาธ bin ในตัวแปร PATH ใน Environment variable เพื่อให้สามารถเข้าถึงไดเรคทอรี่ bin จากพาธใดๆก็ได้ ขั้นตอนนี้เป็นทางเลือกหากไม่ทำเวลาต้องการรัน mongodb ให้เข้าไปยังพาธ bin ของ mongodb พิมพ์คำสั่งสำหรับกำหนดค่าใน PATH ดังนี้

```
export PATH=<mongodb-install-directory>/bin:$PATH
```

* <mongodb-install-directory> คือ พาธที่ทำการติดตั้ง *mongoldb* ไว้ ไม่ต้องระบุเครื่องหมาย < >

แบบฝึกหัด 1-2 : เริ่มใช้งาน mongodb สร้างฐานข้อมูลและเชื่อมต่อด้วย shell

วัตถุประสงค์

- เพื่อเปิดใช้งาน mongodb ได้
- สร้างฐานข้อมูลได้
- เข้าใช้งานด้วย mongoshell เพื่อสั่งงานได้

แนวทางปฏิบัติ

- เตรียมไดเรคทอรี่สำหรับเก็บข้อมูลและ mongodb มีสิทธิอ่านเขียนได้
- ศึกษาชุดคำสั่งสำหรับ start mongodb service เพื่อพร้อมใช้งาน
- ศึกษาชุดคำสั่งสำหรับ mongoshell เพื่อสั่งงาน mongodb

วิธีการปฏิบัติ

1. เปิด terminal สร้างไดเรคทอรี่สำหรับเก็บข้อมูลและ mongodb มีสิทธิอ่านเขียนได้

* <mongodata directory> คือ พาธที่ทำการเตรียมให้ *mongodb* อ่านเขียนไฟล์ ไม่ต้องระบุเครื่องหมาย < >

2. ทำการ start mongodb service ด้วยคำสั่ง ดังนี้

```
mongod --dbpath "/Databases/mongo_data/mychatdb"
```

mongodb จะแสดงข้อความ สำคัญๆ ดังนี้

```
MongoDB starting : pid=14422 port=27017 dbpath=/Databases/
mongo_data/mychatdb 64-bit host=Apaichons-MacBook-Air.local
db version v3.2.7
git version: 4249c1d2b5999ebbf1fdf3bc0e0e3b3ff5c0aaf2
OpenSSL version: OpenSSL 0.9.8zg 14 July 2015
allocator: system
modules: none
build environment:
distarch: x86_64
target_arch: x86_64
options: { storage:
          { dbPath: "/Databases/mongo_data/mychatdb" }
         }
```

mongodb จะใช้ port เริ่มต้นที่ 27017 เพื่อใช้ในการติดต่อฐานข้อมูล

3. เปิด terminal อีกหน้าจอเพื่อเข้า mongoshell พิมพ์คำสั่ง ดังนี้

```
mongo
```

```
MongoDB shell version: 3.2.7
connecting to: test
>
```

mongodb จะเชื่อมต่อเข้าฐานข้อมูลชื่อ test ในตอนเริ่มต้น

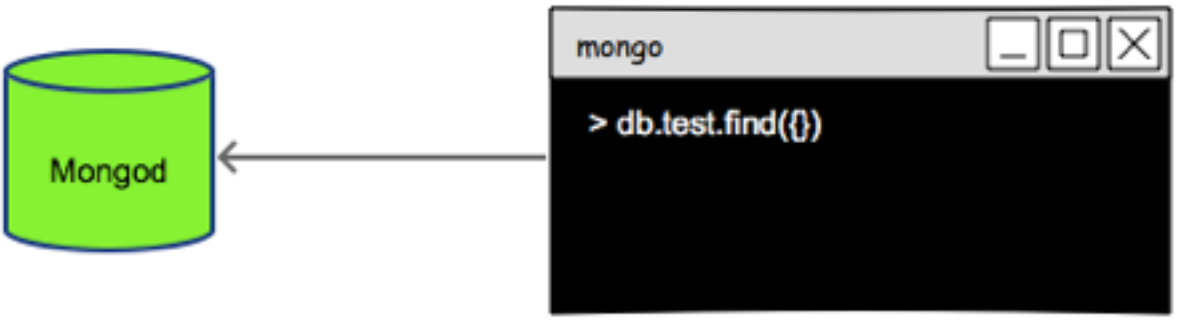
4. เปลี่ยนไปเชื่อมต่อฐานข้อมูลชื่อ guest พิมพ์คำสั่งใน mongoshell ดังนี้

```
use guest
```

```
> use guest
switched to db guest
>
```

สรุป

- mongod คือ service ที่เป็นเหมือนพื้นที่ที่เป็นอาณาจักรของฐานข้อมูล mongodb ที่เราได้กำหนดไดเรคทอรี่ไว้ ภายใน mongod ก็จะประกอบไปด้วย ชุดข้อมูล ฟังก์ชั่นการใช้งาน และออบเจคต่างๆที่สร้างขึ้น
- mongo คือ โปรแกรมแบบ command line ที่ใช้เชื่อมต่อไปยัง mongod เพื่อสั่งงานต่างๆ เช่น สร้างฐานข้อมูล เพิ่ม อ่าน แก้ไข ลบ ข้อมูล เป็นต้น



รูปที่ 1-2.1 mongod service และ mongo shell

แบบฝึกหัด 1-3 : ใช้เครื่องมือช่วยจัดการข้อมูลด้วย Robomongo

วัตถุประสงค์

การใช้ mongoshell ซึ่งเป็นโปรแกรมประเภท command line ถึงแม้จะมีความรวดเร็ว ใช้ทรัพยากรเครื่องไม่มาก ไม่มีปัญหาจุกจิกกวนใจจากการทำงานบางอย่างแบบแอพพลิเคชั่นประเภท GUI แต่ก็มีความไม่สะดวกหลายๆเรื่อง เช่น การดูข้อมูลและโครงสร้างฐานข้อมูล คงไม่สะดวกนักหากจะทำอะไรทีต้องมาพิมพ์คำสั่งตลอด แอพพลิเคชั่นประเภท GUI จึงมีประโยชน์อยู่มาก สำหรับแบบฝึกหัดนี้ ที่เลือก robomongo ทั้งที่มีเครื่องมือมากมายหลายตัวสำหรับจัดการ mongodb แต่เป็นความชอบส่วนตัวและถนัดของผู้เขียน ที่ลองใช้ดูแล้ว พบข้อดีต่างๆ ดังนี้

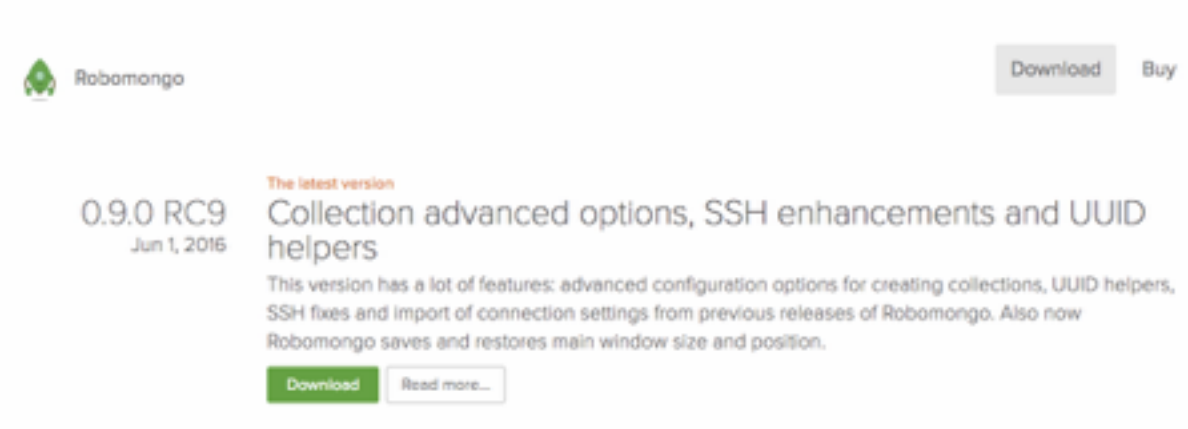
- รองรับหลายๆระบบปฏิบัติการ
- มีการแสดงรายการโครงสร้างข้อมูลและออบเจคต่างๆ ในฐานข้อมูล
- แสดงข้อมูลได้หลายรูปแบบ JSON แบบลำดับชั้น, ตาราง และ text
- มีการใช้ command line ในการสั่งงาน รันได้ทีละหลายๆคำสั่ง บางเครื่องมือสะดวกไปจนไม่รู้ว่าจะสั่งคำสั่งที่ต้องการอย่างไร
- ใช้ทรัพยากรของเครื่องน้อย ใช้มายังไม่เจอปัญหาว่าแอพพลิเคชั่นปิดตัวเองแบบไม่คาดคิด
- ฟรี

แนวทางปฏิบัติ

- ดาวน์โหลดซอฟท์แวร์จากเว็บ https://robomongo.org/download
- ดูขั้นตอนการติดตั้งในวิธีการติดตั้งของระบบปฏิบัติการต่างๆ ซึ่งจะคล้ายคลึงกัน
- เมื่อติดตั้งเสร็จทำการสร้าง connection เพื่อเชื่อมต่อไปยังฐานข้อมูล

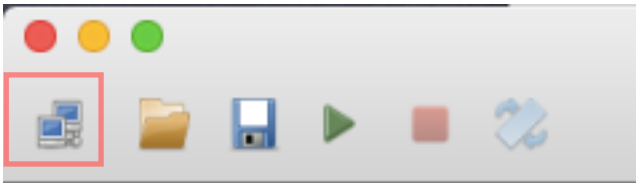
วิธีการติดตั้งสำหรับ Mac

1. ดาวน์โหลดซอฟท์แวร์จากเว็บ https://robomongo.org/download



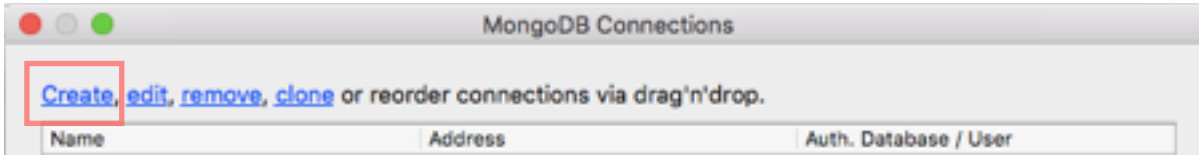
รูปที่ 1-3.1 หน้าเว็บไซต์สำหรับดาวน์โหลด Robomongo

2. ดับเบิ้ลคลิกบนไฟล์ที่ดาวน์โหลดเสร็จแล้ว จากนั้นดับเบิ้ลคลิกบนไฟล์ .dmg
3. ลากไอคอนเข้ายัง Application
4. เปิดโปรแกรม Robomongo
5. สร้าง connection เพื่อเชื่อมต่อไปยังฐานข้อมูล โดยคลิกที่รูปคอมพิวเตอร์ตามภาพ



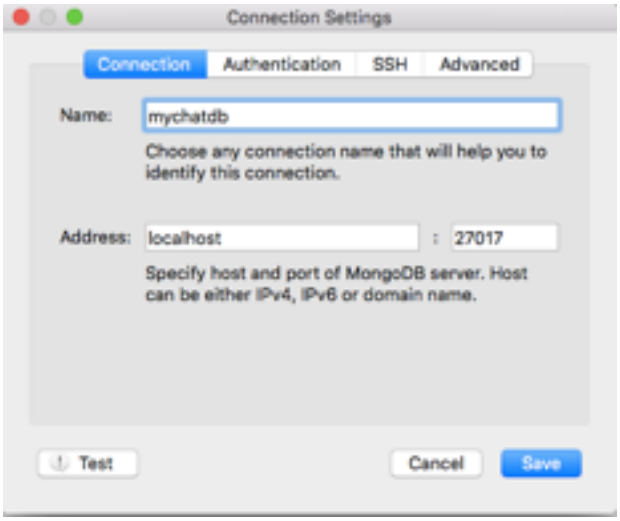
รูปที่ 1-3.2 ปุ่มสำหรับสร้าง connection เชื่อมต่อฐานข้อมูลต่างๆของ mongodb

6. กดปุ่ม **Create** เพื่อสร้างการเชื่อมต่อไปยังฐานข้อมูล mongodb



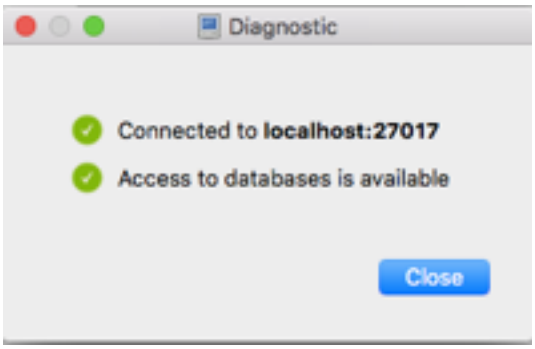
รูปที่ 1-3.3 ปุ่ม **Create** สำหรับสร้าง connection เชื่อมต่อฐานข้อมูลต่างๆของ mongodb

7. ตั้งชื่อการเชื่อมต่อ และกำหนดค่า address ซึ่งสามารถระบุเป็นชื่อเครื่องหรือ ip address และ port ของ mongodb



รูปที่ 1-3.4 หน้าต่างย่อยสำหรับกำหนดค่า Address และ Port ในการเชื่อมต่อฐานข้อมูล

8. จากรูปที่ 1-3.4 กดปุ่ม Test เพื่อตรวจสอบว่าเชื่อมต่อฐานข้อมูลได้หรือไม่ หากเชื่อมต่อได้ให้กดปุ่ม Save



รูปที่ 1-3.5 หน้าต่างย่อยแสดงผลการเชื่อมต่อฐานข้อมูล mongodb

แบบฝึกหัด 1-4 : เพิ่มเอกสารลงฐานข้อมูล

วัตถุประสงค์

เข้าใจวิธีการเพิ่มข้อมูลลงบนฐานข้อมูลลงบนฐานข้อมูล mongodb

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการเพิ่มข้อมูลลงบนฐานข้อมูล
- ทดสอบเพิ่มข้อมูลลงบนฐานข้อมูล
- ตรวจสอบความถูกต้องของข้อมูลที่ได้ทำการเพิ่มเข้าไป

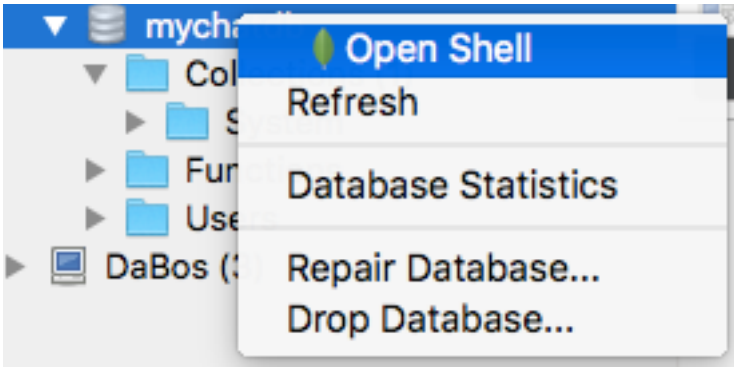
รูปแบบคำสั่ง insert

```
db.collection.insert(
   <document or array of documents>,
   {
     writeConcern: <document>,
     ordered: <boolean>
   }
)
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	document	document หรือ array	ข้อมูลที่ทำการเพิ่มลง collection สามารถระบุเป็น JSON หรือ array ได้ JSON คือ 1 รายการ array คือ หลายๆรายการ
2	writeConcern	document	ในส่วนของ writeConcern คู่มือเล่มนี้จะไม่ได้กล่าวถึงเนื่องจากมีรายละเอียดปลีกย่อยอีกพอสมควร และจะใช้เมื่อมีการ ขยายการใช้งานฐานข้อมูลออกเป็นหลายๆเครื่อง ซึ่งเป็นขั้นตอนการใช้งานในระดับสูง ในเบื้องต้นจะยังไม่ได้ใช้พารามิเตอร์ตัวนี้
3	ordered	boolean	เรียงลำดับ array หรือไม่ โดยค่าที่กำหนดเบื้องต้น คือ true และส่วนมากนักพัฒนาก็ต้องการให้เรียงตามที่ตนเองส่งค่าเข้าไป ดังนั้นค่านี้ก็มักจะไม่ได้กำหนดเช่นกันเมื่อใช้งานจริง

วิธีการปฏิบัติ

1. โปรแกรม Robomongo ให้คลิกขวาบนฐานข้อมูล mychatdb แล้วเลือก Open Shell



รูปที่ 1-4.1 หน้าต่างย่อยสำหรับเลือกว่าจะจัดการเรื่องใดกับฐานข้อมูล

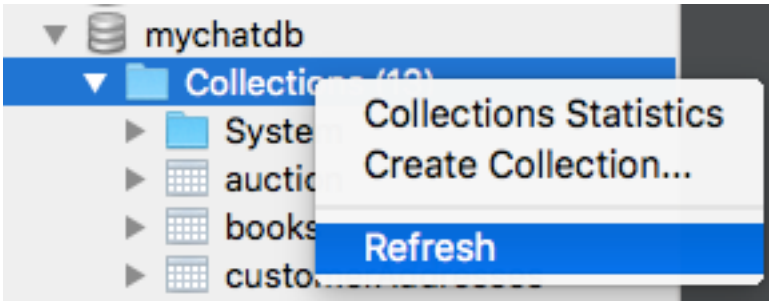
2. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.test.insert({name:"Test",createdAt: new Date()})
```

3. กดปุ่ม execute หรือ F5 (Window) , command + Enter (Mac)

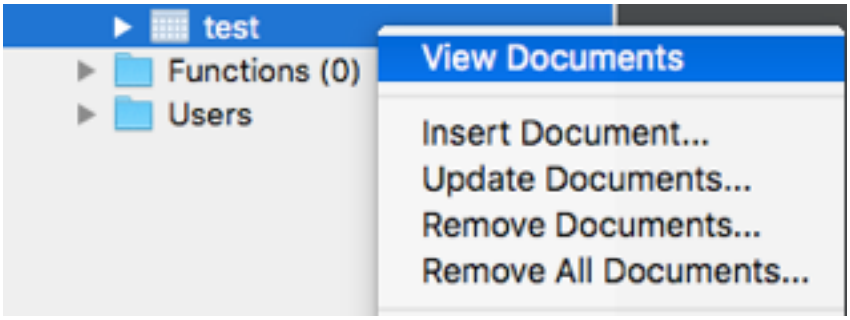
```
Inserted 1 record(s) in 39ms
```

4. เลือกที่ Collections คลิกขวา เลือก Refresh



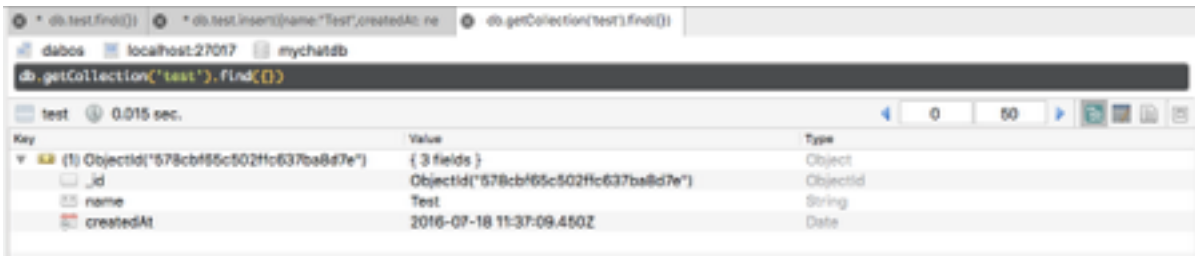
รูปที่ 1-4.2 หน้าต่างย่อยสำหรับ Refresh Collections ที่มีการสร้างขึ้นใหม่

5. เลือกที่ collection ชื่อ test แล้วเลือก View Documents จะสังเกตได้ว่าเมื่อเทียบกับ ฐานข้อมูลแบบ relational เราไม่ได้ทำการสั่งสร้าง collection ขึ้นมาก่อนเลย แต่ก็สามารถเพิ่มข้อมูลลงไปได้



รูปที่ 1-4.3 หน้าต่างย่อยสำหรับจัดการ Colleciton ที่เลือก

6. โปรแกรมจะเปิดหน้าต่างย่อยโดยการรันคำสั่ง db.getCollection('test').find({}) แล้วแสดงข้อมูลเป็น ลำดับชั้นแบบ tree hierarchy



รูปที่ 1-4.4 หน้าต่างย่อยสำหรับดูข้อมูลใน Colleciton

แบบฝึกหัด 1-5 : เพิ่มหลายๆเอกสารลงฐานข้อมูล

วัตถุประสงค์

เข้าใจวิธีการเพิ่มข้อมูลหลายๆรายการลงบนฐานข้อมูลลงบนฐานข้อมูล mongodb

แนวทางปฏิบัติ

- ศึกษาขั้นตอนทุกอย่างเหมือนกับแบบฝึกหัด 1-4 แต่เปลี่ยนข้อมูลที่ป้อนเข้าไปเป็น array

วิธีการปฏิบัติ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.test.insert([
{name:"S1",major:"Computer Science",subject:"Math",score:90},
{name:"S2",major:"Computer Engineer",subject:"Physics",score:75},
{name:"S3",major:"Information Technology",subject:"Physics",score:60},
{name:"S4",major:"Computer Science",subject:"Math",score:77},
{name:"S5",major:"Information Technology",subject:"Physics",score:88},
{name:"S6",major:"Information Technology",subject:"Programming I",score:65},
{name:"S7",major:"Computer Engineer",subject:"Physics",score:55},
{name:"S8",major:"Computer Science",subject:"Programming I",score:66},
{name:"S9",major:"Computer Education",subject:"Math",score:77},
{name:"S10",major:"Information Technology",subject:"Physics",score:59},
{name:"S11",major:"Computer Engineer",subject:"Programming I",score:56},
{name:"S12",major:"Computer Science",subject:"Physics",score:98},
{name:"S13",major:"Information Technology",subject:"Math",score:72},
{name:"S14",major:"Computer Engineer",subject:"Programming I",score:69},
{name:"S15",major:"Computer Engineer",subject:"Physics",score:67},
{name:"S16",major:"Computer Science",subject:"Math",score:73},
{name:"S17",major:"Computer Engineer",subject:"Physics",score:99},
{name:"S18",major:"Computer Engineer",subject:"Math",score:77},
{name:"S19",major:"Information Technology",subject:"Programming I",score:66},
{name:"S20",major:"Information Technology",subject:"Physics",score:66},
])
```

2. ทำการประมวลผลเหมือนแบบฝึกหัดที่ 1-4 จะพบว่าข้อมูลถูกป้อนเข้าฐานข้อมูลจากชุดคำสั่งนี้ 20 รายการ

แบบฝึกหัด 1-6 : การกำหนดเงื่อนไข ในการตรวจสอบชนิดข้อมูลที่ถูกต้อง

วัตถุประสงค์

จากแบบฝึกหัดที่ผ่านมาทั้ง 1-4 และ 1-5 จะเห็นได้ว่า mongodb หรือ ฐานข้อมูลประเภท NOSQL มีความยืดหยุ่นมากเกินไป ในการใช้งานจริงมักมีเงื่อนไขในการกำหนดให้ข้อมูลที่สำคัญบางฟิลด์มีความจำเป็นต้องมีรูปแบบที่ถูกต้องเท่านั้น

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการกำหนดรูปแบบการตรวจสอบข้อมูลของฟิลด์ที่ต้องการ
- ทดสอบเพิ่มข้อมูลที่มีรูปแบบไม่ถูกต้อง ต้องไม่สามารถเพิ่มข้อมูลได้
- ทดสอบเพิ่มข้อมูลที่มีรูปแบบถูกต้อง ต้องสามารถเพิ่มข้อมูลได้

รูปแบบคำสั่ง validator

```
db.createCollection( "collection",
   {
      validator: { <$or,$and>:
         [
            { <field>: { $type: "string" } },
            { <field>: { $exists: true } },
            { <field>: { $regex: /@mongodb\.com$/ } },
            { <field>: { $in: [ "Unknown", "Incomplete" ] } }
         ]
      },
      validationAction: <"warn","error">
   }
)
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	collection	string	ชื่อ collection ที่ต้องการสร้าง
2	$or,$and	array	• ใช้ $or สำหรับเมื่อต้องการกำหนดเงื่อนไขใน array ตามเงื่อนไขที่เป็นจริงเงื่อนไขใดเงื่อนไขหนึ่ง • ใช้ $and สำหรับเมื่อต้องการกำหนดเงื่อนไขใน array ตามเงื่อนไขที่เป็นจริงทั้งหมด
3	field	string	ชื่อฟิลด์ที่ต้องการระบุเงื่อนไข
4	$type	string	ระบุชนิดของข้อมูลของฟิลด์นั้น
5	$exists	boolean	ต้องมีการเก็บค่าฟิลด์นี้ในฐานข้อมูล สำหรับเงื่อนไขนี้ ถ้าไม่จำเป็นต้องมีค่าก็ไม่จำเป็นต้องระบุ

#	พารามิเตอร์	ชนิด	รายละเอียด
6	$regex	regular expression	รูปแบบ regular expression ที่ JavaScript รองรับ
7	$in	array	ค่าที่มีใน array เท่านั้น
8	validateAction	warn,error	เมื่อพบข้อมูลไม่ถูกต้องจะแค่เจ้งเตือน warn หรือแจ้งข้อผิดพลาด error หากระบุเป็น error จะไม่มีการเพิ่มหรือแก้ไขข้อมูลในกรณีที่มีข้อมูลผิดรูปแบบ

วิธีการปฏิบัติ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อกำหนดการตรวจสอบรูปแบบข้อมูล ดังนี้

```
db.createCollection( "customers",
   {
      validator: { $and:
         [
            { idCardNo: { $exists: true,$type:"string" } },
            { firstName: { $exists: true,$type:"string" } },
            { lastName: { $exists: true,$type:"string" } },
            { registeredDate: { $exists: true,$type:"date" } },
            { sex: { $in: [ "M", "F" ] } },
            { email:{ $regex: /@mongodb\.com$/ }},
            { mobileNo:{ $regex: /^\(\d\d\d\)\d\d\d-\d\d\d\d$/ }}
         ]
      },
      validationAction: "error"
   }
)
```

2. กดปุ่ม execute หรือ F5 (Window) , command + Enter (Mac)

```
{
    "ok" : 1.0
}
```

3. ป้อนคำสั่งเพื่อทดสอบการเพิ่มข้อมูล และทำการประมวลผล ดังนี้

```
db.customers.insert({idCardNo:”9999999999”,"firstName":
"สมชาย","lastName":"จรดปลายเท้า", registeredDate: new
Date(),sex:"M", mobileNo: "12323"})
```

```
Document failed validation
```

4. ป้อนคำสั่งเพื่อทดสอบการเพิ่มข้อมูล และทำการประมวลผล ดังนี้

```
db.customers.insert({idCardNo:”9999999999","firstName":
"สมชาย","lastName":"จรดปลายเท้า", registeredDate: new
Date(),sex:"M","email":"somchai@mongodb.com", "mobileNo":
"(081)123-4567"})
```

```
Inserted 1 record(s) in 3ms
```

สรุป

ในแบบฝึกหัดได้กำหนด validator ใช้เงื่อนไข $and เป็นหลัก ดังนั้นเงื่อนไขต่างๆที่กำหนดไว้ในทุกฟิลด์ต้องเป็นจริงเท่านั้นจึงจะทำการเพิ่มหรือแก้ไขข้อมูลได้ ในขั้นตอนที่ 3 ได้ลองทดสอบด้วยการป้อนข้อมูลที่ไม่ถูกต้องลงไป ทั้ง email ที่ไม่ต้องป้อน และ mobileNo ที่มีรูปแบบไม่ถูกต้อง ส่วนขั้นตอนที่ 4 ได้ปรับข้อมูลให้ถูกต้องจึงทำให้สามารถเพิ่มลงฐานข้อมูลได้

แบบฝึกหัด 1-7 : การแก้ไขข้อมูล

วัตถุประสงค์
เข้าใจวิธีการแก้ไขข้อมูลรายการเดียวและหลายๆรายการในฐานข้อมูล mongodb

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการแก้ไขข้อมูลรายการเดียวและหลายรายการ
- ศึกษาข้อควรระวังในการแก้ไขข้อมูล เพื่อป้องกันข้อผิดพลาดในการแก้ไขข้อมูลผิด
- ตรวจสอบข้อมูลที่ถูกแก้ไขแล้ว

รูปแบบคำสั่ง update

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>
   }
)
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	query	document	เงื่อนไขในการกรองข้อมูลในรูปแบบ JSON
2	update	document	ค่าที่ต้องการแก้ไขในรูปแบบ JSON
3	upsert	boolean	ในกรณีที่มีการค้นหาแล้วพอว่ามีค่านั้นอยู่แล้วในฐานข้อมูล ถ้ากำหนด เป็น true ไว้ จะทำการแก้ไขรายการนั้นให้ ถ้าไม่มีจะทำการเพิ่มข้อมูลรายการใหม่เข้าไป
4	multi	boolean	ถ้าต้องการแก้ไขหลายรายการให้กำหนดเป็น true หากไม่กำหนด รายการจะถูกแก้ไขเพียงรายการเดียวเท่านั้น
5	writeConcern	document	ในส่วนของ writeConcern คู่มือเล่มนี้จะไม่ได้กล่าวถึง เนื่องจากมีรายละเอียดปลีกย่อยอีกพอสมควร และจะใช้เมื่อมีการ ขยายการใช้งานฐานข้อมูลออกเป็นหลายๆ เครื่อง ซึ่งเป็นขั้นตอนการใช้งานในระดับสูง ในเบื้องต้นจะยังไม่ได้ใช้พารามิเตอร์ตัวนี้

วิธีการปฏิบัติ

1-7.1 แก้ไขโดยแทนที่ค่าทั้งหมด

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อแก้ไขข้อมูล ดังนี้

```
db.test.update({"name":"S19"},{"name":"S19","score":75})
```

2. กดปุ่ม execute หรือ F5 (Window) , command + Enter (Mac)

```
Updated 1 existing record(s) in 10ms
```

3. ใช้คำสั่ง findOne เพื่อตรวจสอบความถูกต้องของข้อมูล ดังนี้

```
db.test.findOne({name:"S19"});
```

```
{
    "_id" : ObjectId("578e408bc502ffc637ba8d91"),
    "name" : "S19",
    "score" : 75.0
}
```

* จะสังเกตว่าการทำการแก้ไขข้อมูลแบบนี้ ค่าถูกแทนที่ไปทั้งหมดเลย จากเดิมข้อมูลมีฟิลด์ ชื่อ “major” และ “subject” อยู่ด้วย ดังนั้นหากนำไปใช้ผิดวัตถุประสงค์ก็จะทำให้ได้ผลลัพธ์ที่ผิด หากยังคงต้องการเก็บข้อมูลของฟิลด์ที่ไม่เกี่ยวข้องไว้อยู่

1-7.2 แก้ไขเฉพาะฟิลด์ที่ต้องการ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อแก้ไขข้อมูล ดังนี้

```
db.test.update({"_id" :ObjectId("573817afbec88ff1af511d54")}
               ,{"$set" : {"name" : "Student 3","score":90
               ,"subject":"Physics 2"}
               })
```

2. กดปุ่ม execute หรือ F5 (Window) , command + Enter (Mac)

```
Updated 1 existing record(s) in 10ms
```

3. ใช้คำสั่ง findOne เพื่อตรวจสอบความถูกต้องของข้อมูล ดังนี้

```
db.test.findOne({name:"Student 3"});
```

```
{
    "_id" : ObjectId("578e408bc502ffc637ba8d81"),
    "name" : "Student 3",
    "major" : "Information Technology",
    "subject" : "Physics 2",
    "score" : 90.0
}
```

* จะสังเกตว่าการทำการแก้ไขข้อมูลแบบนี้ มีการกำหนดพารามิเตอร์ชื่อ $set ครอบค่า JSON อีกชั้นหนึ่ง ในรูปแบบที่เรียกกันว่า embleded ในส่วนของ ObjectId("") ให้เปลี่ยนเป็นค่า _id ของ S3 เนื่องจากในแต่ละเครื่องจะสร้างค่า _id ที่ไม่เหมือนกัน

1-7.3 แก้ไขแบบ Upsert ถ้ามีข้อมูลทำการแก้ไข ถ้าไม่มีทำการเพิ่มข้อมูล

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อแก้ไขข้อมูล ดังนี้

```
db.test.update({name:"S21"},{"name":"S21","score":75 ,
"major":"Statistic" , "subject":"Physics"},{"upsert":true})
```

2. กดปุ่ม execute หรือ F5 (Window) , command + Enter (Mac)

```
Updated 1 new record(s) in 2ms
```

3. ใช้คำสั่ง findOne เพื่อตรวจสอบความถูกต้องของข้อมูล ดังนี้

```
db.test.findOne({"name":"S21"})
```

```
{
    "_id" : ObjectId("578f42a3cf7292ee9d678d96"),
    "name" : "S21",
    "score" : 75.0,
    "major" : "Statistic",
    "subject" : "Physics"
}
```

4. ลองแก้ไขข้อมูลแล้วรันคำสั่งอีกครั้ง ดังนี้

```
db.test.update({name:"S21"},{"name":"S21","score":75 ,
"major":"Statistic" , "subject":"Physics"},{"upsert":true})
```

```
Updated 1 existing record(s) in 11ms
```

5. ใช้คำสั่ง findOne เพื่อค้นหาข้อมูล ดังนี้

```
db.test.findOne({"name":"S21"})
```

```
null
```

ข้อมูล "name":"S21" หาไม่พบเนื่องจากถูกแก้ไขเป็น S22 ไปเรียบร้อยแล้ว

6. ใช้คำสั่ง findOne เพื่อค้นหาข้อมูลของ S22 ดังนี้

```
db.test.findOne({"name":"S22"})
```

```
{
    "_id" : ObjectId("578f42a3cf7292ee9d678d96"),
    "name" : "S22",
    "score" : 66.0,
    "major" : "Statistic2",
    "subject" : "Physics2"
}
```

* จะสังเกตว่าในการแก้ไขครั้งแรก ทำการค้นหาข้อมูล “name":"S21" ไม่พบข้อมูลระบบจึงทำการเพิ่มข้อมูลเข้าไปยังฐานข้อมูล เมื่อลองแก้ไขค่าที่ต้องปรับเปลี่ยนจากชุดคำสั่งเดิม แต่เงื่อนไขในค้นหายังค้นหา “name":"S21" ในขั้นตอนที่ 4 นั้นพบข้อมูล “name":"S21” จึงได้ทำการเทนที่ค่าต่างๆที่ได้ระบุไว้

1-7.4 แก้ไขข้อมูลแบบหลายรายการ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อแก้ไขข้อมูล ดังนี้

```
db.test.update({"subject":"Math"},{$set:{"subject":"Math
II"}},{"multi":true})
```

```
Updated 6 existing record(s) in 11ms
```

2. ใช้คำสั่ง find เพื่อตรวจสอบความถูกต้องของข้อมูล ดังนี้

```
db.test.find({"subject":"Math II"})
```

* โดยปกติแล้ว mongodb จะกำหนดค่าเริ่มต้นในการแก้ไขข้อมูลให้สามารถแก้ไขได้เพียงรายการเดียว หากต้องการแก้ไขทุกค่าตามเงื่อนไข จะต้องกำหนดพารามิเตอร์ multi เป็น true ทั้งนี้เพื่อให้ผู้แก้ไขข้อมูลมั่นใจว่าต้องการแก้ไขทั้งหมดจริงๆเพื่อช่วยลดข้อผิดพลาดที่เกิดขึ้น ที่เห็นกันบ่อยๆจากการใช้คำสั่ง SQL ในฐานข้อมูลแบบ Relational

แบบฝึกหัด 1-8 : การลบข้อมูล

วัตถุประสงค์
เข้าใจวิธีการลบข้อมูลรายการเดียวและหลายๆรายการในฐานข้อมูล mongodb

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการลบข้อมูลรายการเดียวและหลายรายการ
- ศึกษาข้อควรระวังในการลบข้อมูล เพื่อป้องกันข้อผิดพลาดในการลบข้อมูลผิด
- ตรวจสอบข้อมูลที่ถูกลบแล้ว

รูปแบบคำสั่ง delete

```
db.collection.remove(
   <query>,
   <justOne>
)
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	query	document	เงื่อนไขในการกรองข้อมูลในรูปแบบ JSON
2	justOne	boolean	ต้องการลบเพียงรายเดียวหรือไม่ โดยปกติแล้วจะกำหนดค่าเป็น false คือจะลบทุกรายการที่ค้นหาเจอตามเงื่อนไข

วิธีการปฏิบัติ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อลบข้อมูล ดังนี้

```
db.test.remove({"subject":"Physics"},{"justOne":true})
```

```
Removed 1 record(s) in 12ms
```

2. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อลบข้อมูล โดยไม่มี option justOne ดังนี้

```
db.test.remove({"subject":"Physics"})
```

```
Removed 7 record(s) in 4ms
```

แบบฝึกหัด 1-9 : การค้นหาข้อมูล

วัตถุประสงค์

เข้าใจวิธีการค้นหาข้อมูลในฐานข้อมูล mongodb ด้วยคำสั่ง find

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการค้นหาข้อมูลในฐานข้อมูล mongodb
- ทดสอบค้นหาฐานข้อมูล

รูปแบบคำสั่ง Find

```
db.collection.find(query, projection)
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	query	document	รูปแบบการกรองข้อมูล หากต้องการดึงข้อมูลทั้งหมดให้ใช้ {} สำหรับเงื่อนไขในกาตค้นหาข้อมูลที่ซับซ้อนจะกล่าวถึงในบทถัดไป
2	projection	document	การเลือกแสดงผลของฟิลด์ ต้องการให้แสดงหรือไม่แสดง โดยระบุเป็น 1 เพื่อแสดง และ 0 ไม่แสดง

รูปแบบการใช้ projection

```
{ field1: <value>, field2: <value> ... }
```

วิธีการปฏิบัติ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อค้นหาข้อมูล ดังนี้

```
db.test.find()
db.test.findOne({name:"S6"})
db.test.find({subject:"Physics 2"})
db.test.find().limit(2)
```

- db.test.find() - แสดงข้อมูลทั้งหมด
- db.test.findOne({name:"S6"}) - แสดงข้อมูลรายการเดียว โดยค้นหาจากฟิลด์ name ที่มีค่าเท่ากับ "S6"
- db.test.find({subject:"Physics 2"}) - ค้นหาข้อมูลทั้งหมด ที่มีค่าฟิลด์ subject เท่ากับ "Physics 2"
- db.test.find().limit(2) - ค้นหาโดยไม่มีเงื่อนไข แสดงเพียง 2 รายการ

* ในการระบุค่าหลายๆฟิลด์ใน query จะเป็นการเชื่อมเงื่อนไขในการค้นหาด้วย AND

```
db.test.find({},{"name":1,"score":1})
db.test.find({},{"score":0})
db.test.find({},{ "_id":0,"name":1 })
```

2. ระบุเฉพาะฟิลด์ที่ต้องการด้วย projection ป้อนคำสั่งดังนี้
- {"name":1,"score":1} - แสดงข้อมูลฟิลด์ "name" และ "score"
- {"score":0} - ไม่แสดงข้อมูลฟิลด์ "score" นอกนั้นแสดงหมด
- { "_id":0,"name":1 } - ไม่แสดง "_id" แสดง "name"

* ฟิลด์ "_id" จะแสดงเป็นหลักเพราะเป็น primary key ที่ mongodb กำหนดให้ต้องมีทุกราย ในการเลือกใช้ควรเลือกใช้ 0 หรือ 1 อย่างใดอย่างหนึ่งเพื่อจะได้ไม่เกิดการสับสนต่อการกำหนดเงื่อนไข หลีกเลี่ยงการใช้ 0 และ 1 ปนกัน

บทที่ 2 การค้นหาแบบมีเงื่อนไข

การค้นหาแบบมีเงื่อนไขคืออะไร ?

ในฐานข้อมูลระบบใดๆในองค์กรมักจะมีข้อมูลจำนวนมหาศาล ซึ่งมีความจำเป็นต้องมีความสามารถในการกรองข้อมูลที่ต้องการตามเงื่อนไขต่างๆ เช่น

- การค้นหาตัวเลขที่มีค่า มากกว่า น้อยกว่า เท่ากับ มากกว่าหรือเท่ากับ น้อยกว่าหรือเท่ากับ
- การค้นหาด้วยเงื่อนไข OR
- การค้นหาด้วยเงื่อนไข AND
- การค้นหาด้วยเงื่อนไข NOT
- หรือการค้นหาด้วยเงื่อนไขพิเศษตามรูปแบบของฐานข้อมูลแต่ตัว

ทำไมต้องค้นหาแบบมีเงื่อนไข ?

```
$and : [{field1:value1},{field2:value2},{fieldN:valueN} ]
```

ความหมายเหมือนกับ

```
if(field1==value1 && field2 == value2 && fieldN == valueN)
```

สำหรับนักพัฒนาแอพพลิเคชั่นหรือนักวิเคราะห์ข้อมูลแล้ว คงปฏิเสธไม่ได้ว่าในการทำงานกับฐานข้อมูลการค้นหาข้อมูลด้วยเงื่อนไขต่างๆเป็นสิ่งสำคัญมาก ดังนั้นการเข้าใจรูปแบบพื้นฐานการค้นหาข้อมูลโดยใช้เงื่อนไขต่างๆไม่ว่าจะเป็นฐานข้อมูลชนิดใดก็ล้วนแล้วมีความสำคัญ เพื่อที่ผู้ใช้งานจะได้ค้นหาข้อมูลและมาวิเคราะห์เพื่อใช้ประโยชน์ได้อย่างเต็มประสิทธิภาพ เพราะการใช้สายตาของมนุษย์ในการตรวจสอบเอง ย่อมเสียเวลาและมีข้อผิดพลาดเกิดขึ้นได้ง่ายๆ ยกตัวอย่างเช่น ระบบขายออนไลน์ หากเราต้องการค้นหาข้อมูลสินค้าที่มียอดขายตั้งแต่ 5,000 บาทขึ้นไปในย้อนหลังไป 1 เดือน หากข้อมูลมีหลักหมื่นรายการ การจะเข้าไปตรวจสอบแต่ละรายการเองด้วยตัวคนเดียวคงเสียเวลาเป็นเดือนแน่ และยังมีโอกาสเกิดข้อผิดพลาดอีกมาก ดังนั้นการค้นหาแบบมีเงื่อนไขมีความจำเป็นมากในการเพิ่มประสิทธิภาพในการทำงานสำหรับการวิเคราะห์ข้อมูล

แบบฝึกหัด 2.1 การค้นหาด้วยเงื่อนไขเปรียบเทียบ > มากกว่า, >= มากกว่าเท่ากับ, < น้อยกว่า, <= น้อยกว่าเท่ากับ

วัตถุประสงค์

เข้าใจวิธีการค้นหาข้อมูลในฐานข้อมูล mongodb ด้วยคำสั่ง find พร้อมเงื่อนไขในการเปรียบเทียบค่า โดยใช้ > , >= , < , <=

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการค้นหาข้อมูลในฐานข้อมูล mongodb สำหรับค้นหาค่า โดยใช้ > , >= , < , <=
- ทดสอบค้นหาข้อมูลโดยใช้เงื่อนไขดังกล่าว

รูปแบบคำสั่ง Find ด้วย พารามิเตอร์ >, >= , < , <=

```
db.collection.find({field:{$gt:value}})
db.collection.find({field:{$gte:value}})
db.collection.find({field:{$lt:value}})
db.collection.find({field:{$lte:value}})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$gt	field	ชื่อฟิลด์สงวนที่ใช้สำหรับระบุเงื่อนไขเพื่อเปรียบเทียบค่าที่มากกว่า การใช้งานจะระบุภายใต้ฟิลด์ที่ต้องการเปรียบเทียบอีกชั้นหนึ่งแบบ embleded
2	$gte	field	ชื่อฟิลด์สงวนที่ใช้สำหรับระบุเงื่อนไขเพื่อเปรียบเทียบค่าที่มากกว่าหรือเท่ากับ การใช้งานจะระบุภายใต้ฟิลด์ที่ต้องการเปรียบเทียบอีกชั้นหนึ่งแบบ embleded
3	$lt	field	ชื่อฟิลด์สงวนที่ใช้สำหรับระบุเงื่อนไขเพื่อเปรียบเทียบค่าที่น้อยกว่า การใช้งานจะระบุภายใต้ฟิลด์ที่ต้องการเปรียบเทียบอีกชั้นหนึ่งแบบ embleded
4	$lte	field	ชื่อฟิลด์สงวนที่ใช้สำหรับระบุเงื่อนไขเพื่อเปรียบเทียบค่าที่น้อยกว่าหรือเท่ากับ การใช้งานจะระบุภายใต้ฟิลด์ที่ต้องการเปรียบเทียบอีกชั้นหนึ่งแบบ embleded

วิธีการค้นหาด้วยเงื่อนไขเปรียบเทียบ > มากกว่า, >= มากกว่าเท่ากับ, < น้อยกว่า, <= น้อยกว่าเท่ากับ

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อค้นหาข้อมูล ดังนี้

```
db.test.find({score:{$gt:60}})
db.test.find({score:{$gte:60}})
db.test.find({score:{$lt:60}})
db.test.find({score:{$lte:60}})
```

2. กดปุ่ม execute หรือ F5 (Window) , command + Enter (Mac)

2.2 การค้นหาด้วยเชื่อมเงื่อนไขมากกว่าหนึ่งเงื่อนไขด้วย OR

วัตถุประสงค์
เข้าใจวิธีการค้นหาข้อมูลในฐานข้อมูล mongodb ด้วยคำสั่ง find พร้อมการเชื่อมเงื่อนไขด้วย OR

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการค้นหาข้อมูลในฐานข้อมูล mongodb สำหรับค้นหาค่า โดยใช้ OR เชื่อมเงื่อนไขมากกว่าหนึ่งเงื่อนไข
- ทดสอบค้นหาข้อมูลโดยใช้เงื่อนไขดังกล่าว

รูปแบบคำสั่ง Find ด้วย การเชื่อมเงื่อนไข OR

```
db.test.find({$or:[{field:value},{field:value}]})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$or	array [document,documeny]	รับค่าเป็น array document ที่ระบุเงื่อนไขต่างๆ โดยจะกรองข้อมูลที่มีเงื่อนไขใดตรงกับเงื่อนใดๆที่ระบุ
2	{field:value}	document	เงื่อนไขต่างๆที่ต้องการให้เชื่อมกับเงื่อนไข OR

วิธีการค้นหา Find ด้วยการเชื่อมเงื่อนไข OR

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อค้นหาข้อมูล ดังนี้

```
db.test.find({$or:[{major:"Computer Science"},
{subject:"Math II"}]})
```

	_id	name	major	subject	score
1	ObjectId(...	S1	Computer Science	Math II	90.0
2	ObjectId(...	S4	Computer Science	Math II	77.0
3	ObjectId(...	S8	Computer Science	Programming I	66.0
4	ObjectId(...	S9	Computer Education	Math II	77.0
5	ObjectId(...	S13	Information Technology	Math II	72.0
6	ObjectId(...	S16	Computer Science	Math II	73.0
7	ObjectId(...	S18	Computer Engineer	Math II	77.0

การค้นหาด้วยพารามิเตอร์ $or จะรับค่าเป็น array ซึ่งภายในให้ระบุ JSON field1:<value> , field2:<value> ,..fieldN:<value> จะแสดงข้อมูลตามเงื่อนไขใดเงื่อนไขหนึ่งภายใน array ที่เป็นจริง ในแบบฝึกหัดนี้เราได้ระบุเงื่อนไขของฟิลด์ major:"Computer Science” หรือ subject:"Math II”

$or : [{field1:value1},{field2:value2},{fieldN:valueN}]

ความหมายเหมือนกับ
if(field1==value1 ll field2 == value2 ll fieldN == valueN)

2.3 การค้นหาด้วยเชื่อมเงื่อนไขมากกว่าหนึ่งเงื่อนไขด้วย AND

วัตถุประสงค์

เข้าใจวิธีการค้นหาข้อมูลในฐานข้อมูล mongodb ด้วยคำสั่ง find พร้อมการเชื่อมเงื่อนไขด้วย AND

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการค้นหาข้อมูลในฐานข้อมูล mongodb สำหรับค้นหาค่า โดยใช้ AND เชื่อมเงื่อนไขมากกว่าหนึ่งเงื่อนไข
- ทดสอบค้นหาข้อมูลโดยใช้เงื่อนไขดังกล่าว

รูปแบบคำสั่ง Find ด้วย การเชื่อมเงื่อนไข AND

```
db.test.find({$and:[{field:value},{field:value}]})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$and	array [document,documeny]	รับค่าเป็น array document ที่ระบุเงื่อนไขต่างๆ โดยจะกรองข้อมูลที่มีเงื่อนไขตรงตามที่ระบุทุกเงื่อนไขเท่านั้น
2	{field:value}	document	เงื่อนไขต่างๆที่ต้องการให้เชื่อมกับเงื่อนไข AND

วิธีการค้นหา Find ด้วยการเชื่อมเงื่อนไข AND

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อค้นหาข้อมูล ดังนี้

```
db.test.find({"$and":[ {"major":"Computer Science"},
{"score":{"$gt":80}}]})
```

	_id	name	major	subject	score
1	ObjectId(...	S1	Compute...	Math II	90.0

โดยปกติแล้วหากเงื่อนไขไม่ซับซ้อนมากจะไม่ได้ใช้ เพราะว่าเวลาที่เราระบุเงื่อนไขในการค้นหาหลายๆฟิลด์จะเป็นการเชื่อมเงื่อนไขด้วย AND อยู่แล้ว แต่ถ้ามีการกรองข้อมูลที่ซับซ้อนมากขึ้น ตามโค้ด คือ ต้องการค้นหาผู้เรียนคณะ Computer Sciece ที่มีคะแนนมากกว่า 80 จะไม่สามารถใช้รูปแบบปกติได้ ดังนี้

```
db.test.find({“major":"Computer Science", {"score":{"$gt":
80}}})
```

การค้นหาด้วยพารามิเตอร์ $and จะรับค่าเป็น array ซึ่งภายในให้ระบุ JSON field1:<value> , field2:<value> ,..fieldN:<value> จะแสดงข้อมูลตามเงื่อนไขทั้งหมดใน array เป็นจริง ในแบบฝึกหัดนี้เราได้ระบุเงื่อนไขของฟิลด์ "major":"Computer Science" และ “score":{"$gt":80}

2.4 การค้นหาด้วยเงื่อนไข NOT

วัตถุประสงค์
เข้าใจวิธีการค้นหาข้อมูลในฐานข้อมูล mongodb ด้วยคำสั่ง find พร้อมการเชื่อมเงื่อนไขด้วย NOT

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการค้นหาข้อมูลในฐานข้อมูล mongodb สำหรับค้นหาค่า โดยใช้ NOT
- ทดสอบค้นหาข้อมูลโดยใช้เงื่อนไขดังกล่าว

รูปแบบคำสั่ง Find ด้วย การเชื่อมเงื่อนไข NOT
การค้นให้โดยใช้เงื่อนไข NOT จะให้ผลลัพธ์ตรงกันข้ามกับเงื่อนไขที่เราระบุ เช่น ต้องการค้นข้อมูล score > 80 หากใช้ร่วมกับ NOT สิ่งที่ได้ก็คือ score <= 80 (น้อยกว่าหรือเท่ากับ)

```
db.test.find({field:{$not:value,document}})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$not	field	รับค่าเป็น value หรือ document ที่ระบุเงื่อนไขต่างๆ
2	document	document	ค่าสำหรับฟิลด์ หรือ พารามิเตอร์เปรียบเทียบซ้อนกันในรูปแบบ embleded ที่ต้องการให้กรองข้อมูลที่

วิธีการค้นหา Find ด้วยการเชื่อมเงื่อนไข NOT
ค้นหาข้อมูลด้วยพารามิเตอร์ $not จะให้ผลตรงกันข้าม ตามเงื่อนไข {major:{$not:{$eq:"Computer Science"}}} ค้นหาข้อมูลในฟิลด์ major ที่ มีค่าไม่เท่ากับ "Computer Science"

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อค้นหาข้อมูล ดังนี้

```
db.test.find({major:{$not:{$eq:"Computer Science"}}})
```

	_id	name	major	subject	score
1	ObjectId(...	Student 3	Informati...	Physics 2	90.0
2	ObjectId(...	S6	Informati...	Program...	65.0
3	ObjectId(...	S9	Compute...	Math II	77.0
4	ObjectId(...	S11	Compute...	Program...	56.0
5	ObjectId(...	S13	Informati...	Math II	72.0
6	ObjectId(...	S14	Compute...	Program...	69.0
7	ObjectId(...	S18	Compute...	Math II	77.0

2.5 การค้นหาด้วยค่าพิเศษสำหรับข้อมูลบางชนิด

วัตถุประสงค์

เข้าใจวิธีการค้นหาข้อมูลในฐานข้อมูล mongodb ด้วยคำสั่ง find ด้วยค่าพิเศษสำหรับข้อมูลบางชนิด ในการค้นหาข้อมูลบางชนิดที่มีการเก็บค่าที่พิเศษ อย่างเช่นค่า null , regular expression , ขนาดของ array

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งที่ใช้ในการค้นหาข้อมูลในฐานข้อมูล mongodb สำหรับค้นหาจากค่าพิเศษ เช่น null , regular expression , array
- ทดสอบค้นหาข้อมูลโดยด้วยค่าพิเศษต่างๆ

วิธีการค้นหา Find ด้วยค่าชนิดพิเศษต่าง

1. ที่หน้าจอของ shell ให้ป้อนคำสั่งเพื่อเพิ่มข้อมูลบางชนิดเข้าไป ดังนี้

```
db.test.insert({"name":"S23",
     "classDates":["Monday",
     "Tuesday",
     "Wednesday",
     "Thursday",
     "Friday"]}
     )
```

2. ค้นหาค่าจาก field subject ที่มีค่า null ดังนี้

```
db.test.find({subject:null})
```

	_id	name	score	classDates
1	ObjectId(...	S19	75.0	
2	ObjectId(...	S23		[5 eleme...

3. ค้นหาค่าจาก regular expression ที่มีค่าอยู่ในคำที่ต้องการ ดังนี้

```
db.test.find({"major" : /tion/})
```

	_id	name	major	subject	score
1	ObjectId(...	Student 3	Information Technology	Physics 2	90.0
2	ObjectId(...	S6	Information Technology	Program...	65.0
3	ObjectId(...	S9	Computer Education	Math II	77.0
4	ObjectId(...	S13	Information Technology	Math II	72.0

4. ค้นหาค่าจากฟิลด์ที่เก็บค่า array แล้วมีขนาดเท่ากับ 5 ดังนี้

```
db.test.find({classDates:{$size:5}})
```

	_id	name	classDates
1	ObjectId(...	S23	[5 elements]

5. ค้นหาค่าจากฟิลด์ name:"S23" และแสดง classDates ตั้งแต่ index ตำแหน่งที่ 2 จำนวน 2 ตัว ดังนี้

```
db.test.find({name:"S23"},{classDates:{"$slice":[2,2]}})
```

	_id	name	score	classDates
1	ObjectId(...	S19	75.0	
2	ObjectId(...	S23		[5 eleme...

```
{
    "_id" : ObjectId("57919fe8c502ffc637ba8d98"),
    "name" : "S23",
    "classDates" : [
        "Wednesday",
        "Thursday"
    ]
}
```

6. ค้นหาค่าจากฟิลด์ classDates ที่มีค่า “Monday","Friday" อยู่ใน array ดังนี้

```
db.test.find({classDates:{$all:["Monday","Friday"]}})
```

7. ค้นหาข้อมูลด้วยพารามิเตอร์ $where ในการค้นหาข้อมูลบางครั้งต้องการเปรียบเทียบค่าระหว่างฟิลด์หนึ่งกับอีกฟิลด์หนึ่งใน document เดียวกัน หรือต้องการแปลงค่าบางอย่างก่อน อาจจำเป็นต้องเขียนฟังก์ชั่นที่ซับซ้อนกว่าการส่งค่าให้พารามิเตอร์ ในตัวอย่างนี้จะเป็นการตรวจสอบค่าการสั่งซื้อสินค้า แล้วทำการค้นหาข้อมูลที่มีการสั่งซื้อสินค้าเกินจำนวนที่มีใน Stock

```
db.orderProduct.insert([{"item":"jacket","qty":
5,"inStock":5}
,{"item":"polo","qty":20,"inStock":10}
,{"item":"t-shirt","qty":3,"inStock":99}
])
```

```
db.orderProduct.find({$where:function(){return this.qty
> this.inStock;} })
```

	_id	item	qty	inStock
1	ObjectId(...	polo	20.0	10.0

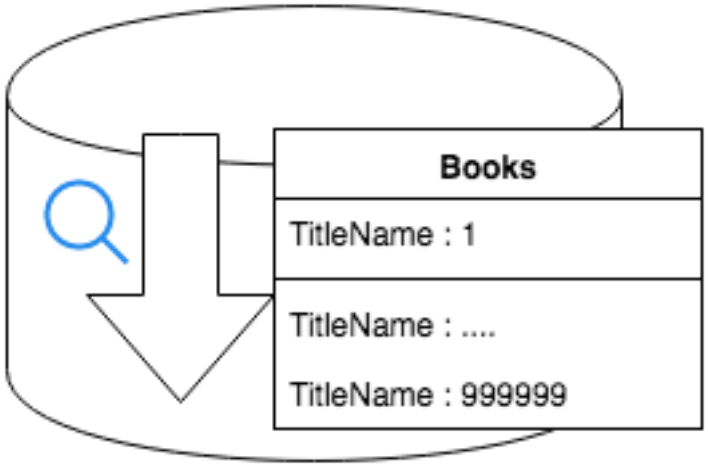
บทที่ 3 การใช้งาน Index

Index คืออะไร ?

ในการจัดเก็บข้อมูลในฐานข้อมูลเมื่อข้อมูลเรามีปริมาณมากขึ้นๆ การค้นหาข้อมูลย่อมช้าลงหากไม่มีการจัดระเบียบของข้อมูลให้เป็นหมวดหมู่เพื่อให้ค้นหาให้ง่ายขึ้น การทำ Index มีตัวอย่างให้เห็นในชีวิตจริงมากมาย เช่น พจนานุกรมมีการแบ่งคำศัพท์เรียงลำดับไว้ตามตัวอักษรโดยแบ่งกลุ่มตามพยัญชนะ ห้องสมุดมีการแบ่งหมวดหมู่ตามประเภทหนังสือ นิยาย ประวัติศาสตร์ การเงิน คอมพิวเตอร์ เป็นต้น จากตัวอย่างที่กล่าวมาการทำ Index ในฐานข้อมูลทุกยี่ห้อก็มีจุดประสงค์เหมือนกันคือต้องการเพิ่มความเร็วในการค้นหาข้อมูลให้รวดเร็วขึ้นหรือ ไม่ช้าลงกว่าเดิมเมื่อข้อมูลมีปริมาณเพิ่มมากขึ้น

ทำไมต้องใช้ Index ?

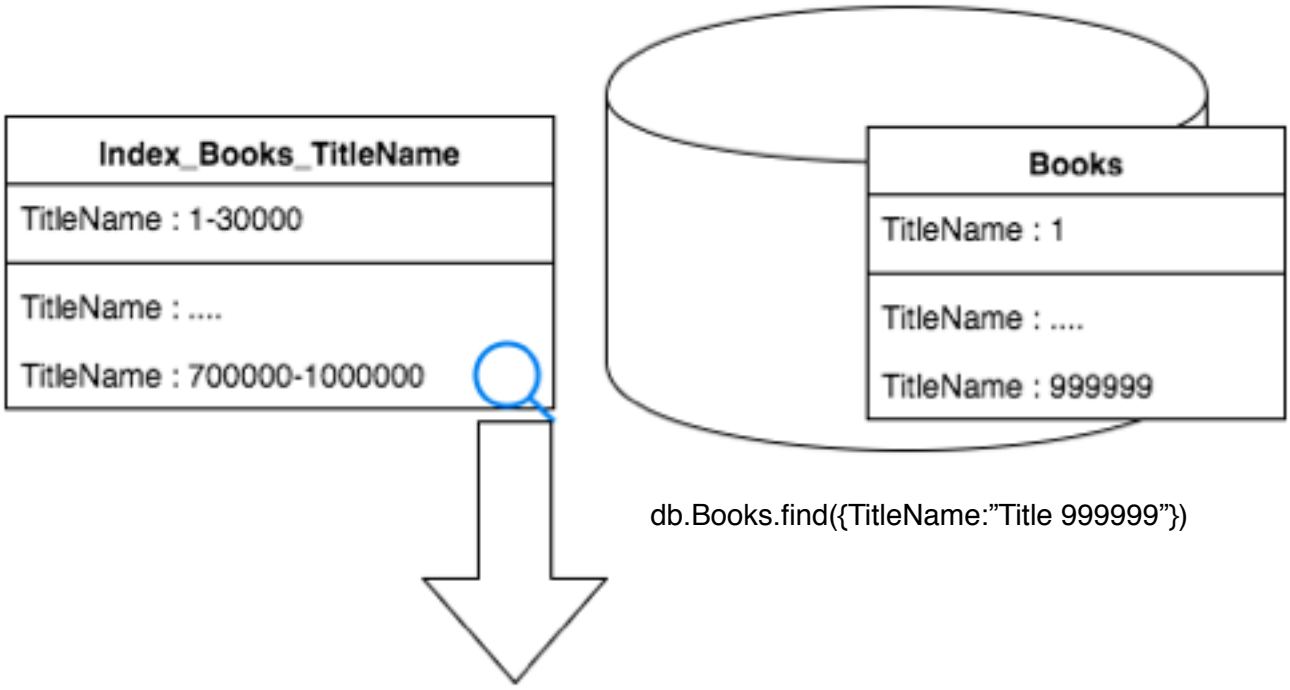
เมื่อข้อมูลเพิ่มมากขึ้นๆ การค้นหาข้อมูลก็จะยิ่งช้าลงหาไม่มีการทำ Index โดยปกติเมื่อเราค้นหาข้อมูลในตารางที่ไม่มี Index ฐานข้อมก็จะทำการแสกนหาข้อมูลจากทั้ง collection ซึ่งหากมีข้อมูลมากและฮาร์ดดิสมีความเร็วในการอ่านไม่มากก็จะทำให้อาจส่งผลกระทบกับแอพพลิเคชั่นอื่นที่เรียกอ่านข้อมูลจากฮาร์ดดิสก์ลูกเดียวกันได้



db.Books.find({TitleName:"Title 999999"})

รูปที่ 3-1 พฤติกรรมการค้นหาของฐานข้อมูลโดยแสกนทั้งตาราง

แต่เมื่อเราสร้าง Index ฐานข้อมูลก็จะจัดหมวดหมู่ของข้อมูลตามคีย์ที่เรากำหนดไปอยู่ในข้อมูลชุดใหม่ เพื่อให้สามารถค้นหาข้อมูลได้รวดเร็วขึ้น หากมีการค้นหาข้อมูลตามคีย์ที่กำหนด



รูปที่ 3-2 พฤติกรรมการค้นหาของฐานข้อมูลโดยหาจาก index

ข้อควรระวัง

1. จากประสบการณ์การทำงานจริง เมื่อมีการเปลี่ยนตำแหน่งผู้ดูแลฐานข้อมูล มักจะมีการตกหล่นในเลือกการแลกเปลี่ยนความรู้ในการออกแบบ index ของตารางนั้นๆ สิ่งที่เกิดขึ้นก็คือ การสร้าง index ขึ้นมาหลายชุดและมีคีย์เกิดขึ้นซ้ำๆกัน ทำให้มีเนื้อที่ index เพิ่มขึ้นหลายชุด และหากมีหลาย index เกินไปบาง ในบางชุดคำสั่งในการค้นหาฐานข้อมูลอาจจะไม่เลือกใช้ index ไหนเลย แต่ไปทำการแสกนทั้งตารางแทน ก็จะทำให้การสร้าง index นั้นไม่เกิดประโยชน์
2. การมี index ไม่ได้ช่วยให้การดึงข้อมูลทั้งตารางเร็วขึ้น หลายคนยังเข้าใจผิดหรือสื่อสารให้ผู้บริหารฟังแล้วเข้าใจผิด จะเร็วขึ้นด้วยการค้นหาแบบมีเงื่อนไขที่ตั้งไว้แล้วเลือกใช้ตรงกับ index เท่านั้น ฐานข้อมูลยิ่งโตขึ้นเท่าไรการต้องการดึงข้อมูลทั้งหมดยังง้ายก็ช้า ซึ่งในการทำงานจริงไม่มีใครอยากใช้ข้อมูล Raw Data ทั้งหมด แต่ก็มีการขอใช้จากหลายๆองค์กรอยู่เป็นประจำ หากจำเป็นต้องใช้ก็พยามต่อรองหรือทยอยให้เป็นช่วงๆเพราะมีผลต่อความเร็วและประสิทธิภาพของระบบ

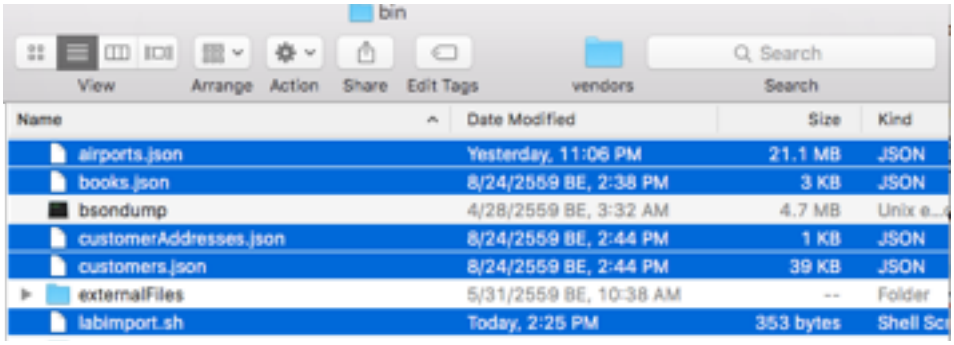
เตรียมข้อมูลเพื่อทำแบบฝึกหัดการสร้าง index

วัตถุประสงค์

เตรียมข้อมูลในการทำ lab การทำ index และ Aggregate function

ขั้นตอนการเตรียมข้อมูล

1. ดาวน์โหลดไฟล์ข้อมูลในรูปแบบ JSON และ script สำหรับ import ข้อมูลได้ที่
   https://github.com/apaichon/courses-mongo/raw/master/mongodata.zip
   https://github.com/apaichon/courses-mongo/blob/master/labimport.sh
2. แตกไฟล์ข้อมูล และ labimport.sh ไปไว้ที่พาธ bin ของ mongodb

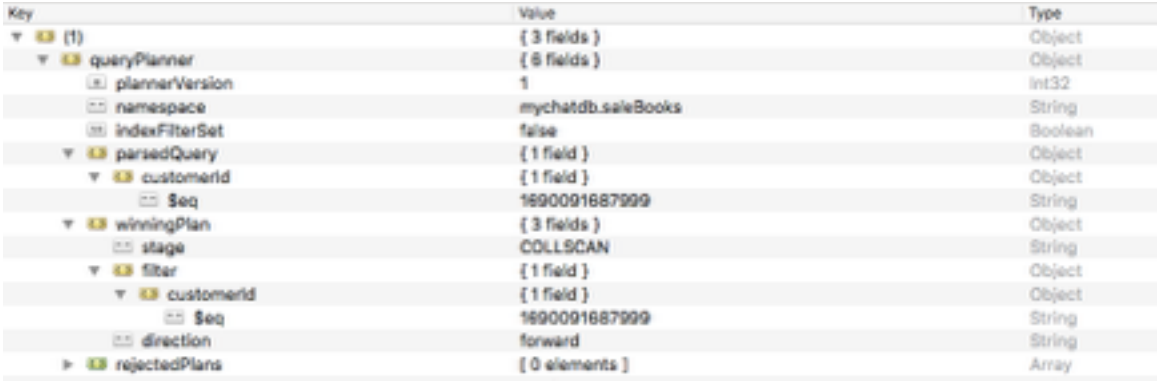


3. เปิดไฟล์ labimport.sh แก้ไขชื่อฐานข้อมูล เป็นชื่อที่ใช้ในการทำ lab

```
./mongoimport --db test --collection customers --file
customers.json
./mongoimport --db test --collection customerAddresses --file
customerAddresses.json
./mongoimport --db test --collection places --file places.json
./mongoimport --db test --collection airports --file
airports.json
./mongoimport --db test --collection saleBooks --file
saleBooks.json
```

4. เปิด terminal แล้วทำการรัน sh labimport.sh

```
sh labimport.sh
2016-08-26T14:26:09.779+0700  connected to: localhost
2016-08-26T14:26:09.797+0700  imported 101 documents
2016-08-26T14:26:09.814+0700  connected to: localhost
2016-08-26T14:26:09.820+0700  imported 2 documents
```



Key	Value	Type
(1)	{ 3 fields }	Object
queryPlanner	{ 6 fields }	Object
plannerVersion	1	Int32
namespace	mychatdb.saleBooks	String
indexFilterSet	false	Boolean
parsedQuery	{ 1 field }	Object
customerId	{ 1 field }	Object
$eq	1690091687999	String
winningPlan	{ 3 fields }	Object
stage	COLLSCAN	String
filter	{ 1 field }	Object
customerId	{ 1 field }	Object
$eq	1690091687999	String
direction	forward	String
rejectedPlans	[0 elements]	Array

5. ที่ โปรแกรม robomongo ตรวจสอบข้อมูลที่นำเข้า ดังนี้

```
db.airports.count()
db.customerAddresses.count()
db.customers.count()
db.places.count()
db.saleBooks.count()
```

* หากใช้ระบบปฏิบัติการวินโดวส์ให้ใช้เปลี่ยน ./mongoimport เป็น mongoimport และเปลี่ยนนามสกุลไฟล์ labimport.sh เป็น labimport.bat

3.1 การใช้คำสั่ง explain

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง explain ร่วมกับคำสั่ง find

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง explain ร่วมกับคำสั่ง find
- ทดสอบการใช้คำสั่ง explain

รูปแบบคำสั่ง explain ร่วมกับคำสั่ง find
การใช้คำสั่ง explain จะช่วยอธิบายพฤติกรรมลำดับขั้นตอนการค้นหาข้อมูลของ mongodb ว่ามีลำดับอย่างไร เพื่อจะช่วยให้ผู้ใช้ทราบสาเหตุเมื่อการค้นหาข้อมูลใช้เวลานาน หรือเงื่อนไขที่ค้นหานั้น mongodb เลือกใช้ index ที่เราได้ออกแบบไว้หรือไม่

```
db.collection.find(query, projection).explain()
```

วิธีการใช้**คำสั่ง explain ร่วมกับคำสั่ง find**

1. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.saleBooks.find({customerId:"1690091687999"}).explain()
```

ข้อมูลสำคัญๆภายใต้พารามิเตอร์ queryPlanner สำหรับนำมาวิเคราะห์ที่ได้จากคำสั่ง explain มีดังนี้

#	พารามิเตอร์	ชนิด	รายละเอียด
1	parsedQuery	document	พารามิเตอร์นี้จะบอกว่าเราได้ส่ง query ที่มีเงื่อนไขอย่างไรบ้าง
2	winningPlan	document	พารามิเตอร์นี้จะบอกว่า mongodb เลือกใช้วิธีการค้นหาอย่างไร ในพารามิเตอร์ย่อย stage ที่บอกว่า COLLSCAN คือจะทำการแสกนหาทั้ง collection ซึ่งหากมีข้อมูลมากก็จะใช้เวลานาน
3	rejectedPlans	array	พารามิเตอร์นี้ในกรณีที่มีหลายๆ index จะบอกว่าไม่เลือกใช้ index ใดบ้าง ซึ่งหากเรามี query หลายๆตัวที่เกี่ยวข้องกับ collection และ index นั้นๆก็สามารถนำมาวิเคราะห์ความจำเป็นที่ต้องมี index ที่อยู่ใน rejectedPlans อยู่หรือไม่

2. ป้อนคำสั่งเพื่อค้นหาข้อมูล ดังนี้

```
db.saleBooks.find({customerId:"1690091687999"})
```

จากการทดสอบค้นหาข้อมูลพบว่าใช้เวลาหลายวินาที เราจะทำการสร้าง index ให้สามารถค้นหาจากคีย์ customerId ให้ได้รวดเร็วขึ้น

3.2 สร้าง Index แบบ single field

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง createIndex และหลักการของการสร้าง index แบบ single field

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง createIndex
- ทดสอบการใช้คำสั่ง createIndex และค้นหาข้อมูล

รูปแบบคำสั่ง createIndex

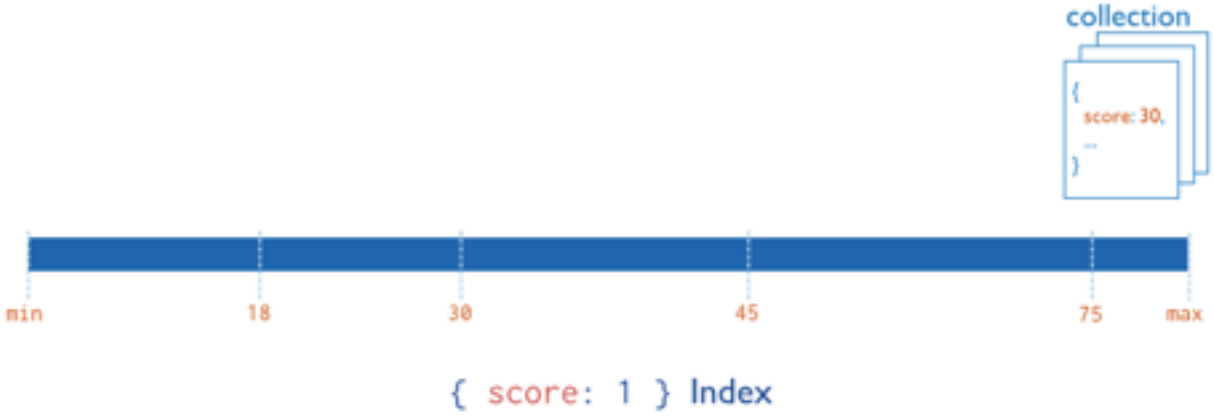
```
db.collection.createIndex(keys, options)
```

options ที่ใช้บ่อยๆในการสร้าง index

#	พารามิเตอร์	ชนิด	รายละเอียด
1	background	boolean	ระบุชื่อฟิลด์ที่ต้องการทำ index โดยค่าที่ระบุหากกำหนดเป็น 1 หมายถึง เรียงลำดับจากน้อยไปหามาก -1 หมายถึง เรียงลำดับจากมากไปหาน้อย เช่น {“field”:1} ,หรือ {“field”:-1}
2	unique	boolean	ฟิลด์ที่ระบุเป็นคีย์ซ้ำกันได้หรือไม่
3	name	string	ตั้งชื่อ index
4	partialFilterExpression	document	การกำหนดเงื่อนไขในการกรองข้อมูลสำหรับแบ่งข้อมูลออกเป็นช่วงเพื่อค้นหาข้อมูลได้เร็วขึ้น เช่น ข้อมูลฟิลด์ score > 90
5	sparse	boolean	สำหรับ sparse index จะเก็บเฉพาะฟิลด์ที่มีค่าเท่านั้น หากมีค่า null หรือไม่มีคีย์ดังกล่าวใน document จะไม่แสดง document นั้น หากใช้ mongodb ตั้งแต่เวอร์ชั่น 3.2 แนะนำให้ใช้ partialIndex

#	พารามิเตอร์	ชนิด	รายละเอียด
6	expireAfterSeconds	integer	กำหนดระยะเวลาที่จะลบข้อมูลเอกสารอัตโนมัติ หน่วยเป็นวินาที จากประสบการณ์ที่เคยใช้ในเวอร์ชั่น 3.2 ต้องใช้กับฟิลด์ที่เก็บค่าเวลา ซึ่งจะกล่าวถึงในแบบฝึกหัดที่ 4-5 ต่อไป

วิธีการสร้าง index แบบ single field

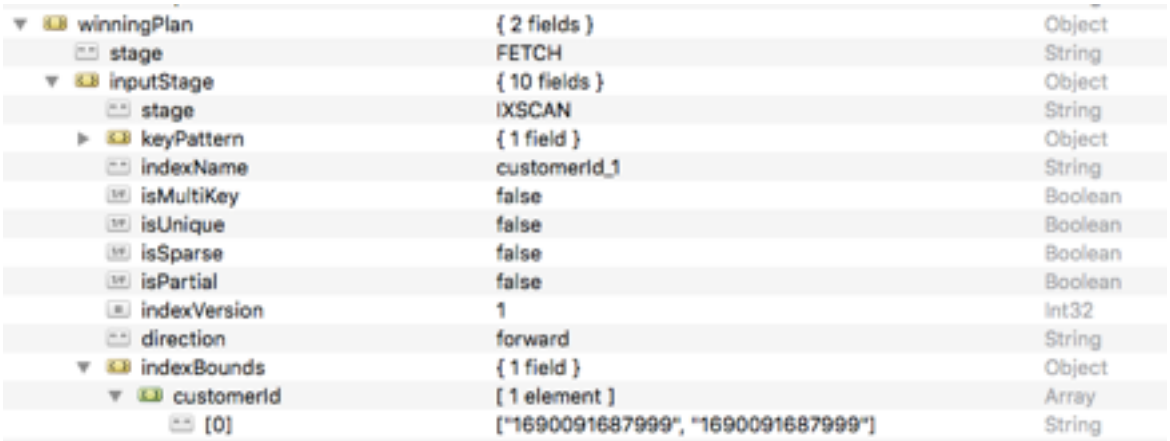


1. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.saleBooks.createIndex({"customerId":1})
```

2. ทดสอบโดยใช้คำสั่ง explain อีกครั้ง ดังนี้

```
db.saleBooks.find({customerId:"1690091687999"}).explain()
```



3. ทดสอบค้นหาข้อมูล ดังนี้

```
db.saleBooks.find({customerId:"1690091687999"})
```

จากขั้นตอนที่ 2 เราพบว่า ในฟิลด์ inputStage.stage เป็นค่า IXSCAN และมีรายระเอียดของ index ที่ถูกเลือกในการค้นหา และเมื่อเราลองทดสอบค้นหาข้อมูลผลที่ได้คือได้ค้นหาข้อมูลได้เร็วขึ้น

3.3 สร้าง Index แบบ Unique Index

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง createIndex และหลักการของการสร้าง index แบบ unique

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง createIndex และการเพิ่ม options สำหรับ unique
- ทดสอบการใช้คำสั่ง createIndex และค้นหาข้อมูล

รูปแบบคำสั่ง createIndex แบบ unique index

```
db.collection.createIndex(keys, {"unique":true})
```

การสร้าง index แบบ unique ฟิลด์ที่กำหนดเป็นคีย์จะต้องไม่มีค่าที่ซ้ำกัน

วิธีการสร้าง createIndex แบบ unique index

1. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.customers.createIndex({"idCardNo":1},{"unique":true})
```

2. ทดสอบป้อนข้อมูลที่มีค่า idCardNo ซ้ำกัน ป้อนคำสั่ง ดังนี้

```
db.customers.insert({“idCardNo":"1698042855799",
"name":"test"})
```

```
E11000 duplicate key error index: mychatdb.customers.$idCardNo_1 dup key: { : "1698042855799" }
```

3.4 สร้าง Index แบบ Compound Index

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง createIndex และหลักการของการสร้าง index แบบ compound index

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง createIndex แบบ compound index
- ทดสอบการใช้คำสั่ง createIndex และค้นหาข้อมูลจาก compound index

รูปแบบคำสั่ง createIndex แบบ compound index

```
db.collection.createIndex( { <field1>: <type>, <field2>: <type2>,
... } )
```

การสร้าง compound index นิยมใช้สำหรับการค้นหาครั้งเดียวหลายๆฟิลด์ โดยจะเลือกฟิลด์ที่ได้รับความนิยมและแอพพลิเคชั่นนั้นได้ออกแบบหรือกำหนดมาตรฐานไว้ เช่น ใน collection ข้อมูลการขายหนังสือ ฟิลด์ที่มักค้นหาบ่อยๆคือ วันที่ขาย และชื่อหนังสือ

#	พารามิเตอร์	ชนิด	รายละเอียด
1	fieldN	string	ระบุชื่อฟิลด์ที่ต้องการทำ index
2	type	integer	1 หมายถึง เรียงลำดับจากน้อยไปหามาก -1 หมายถึง เรียงลำดับจากมากไปหาน้อย เช่น {“field”:1} ,หรือ {“field”:-1}

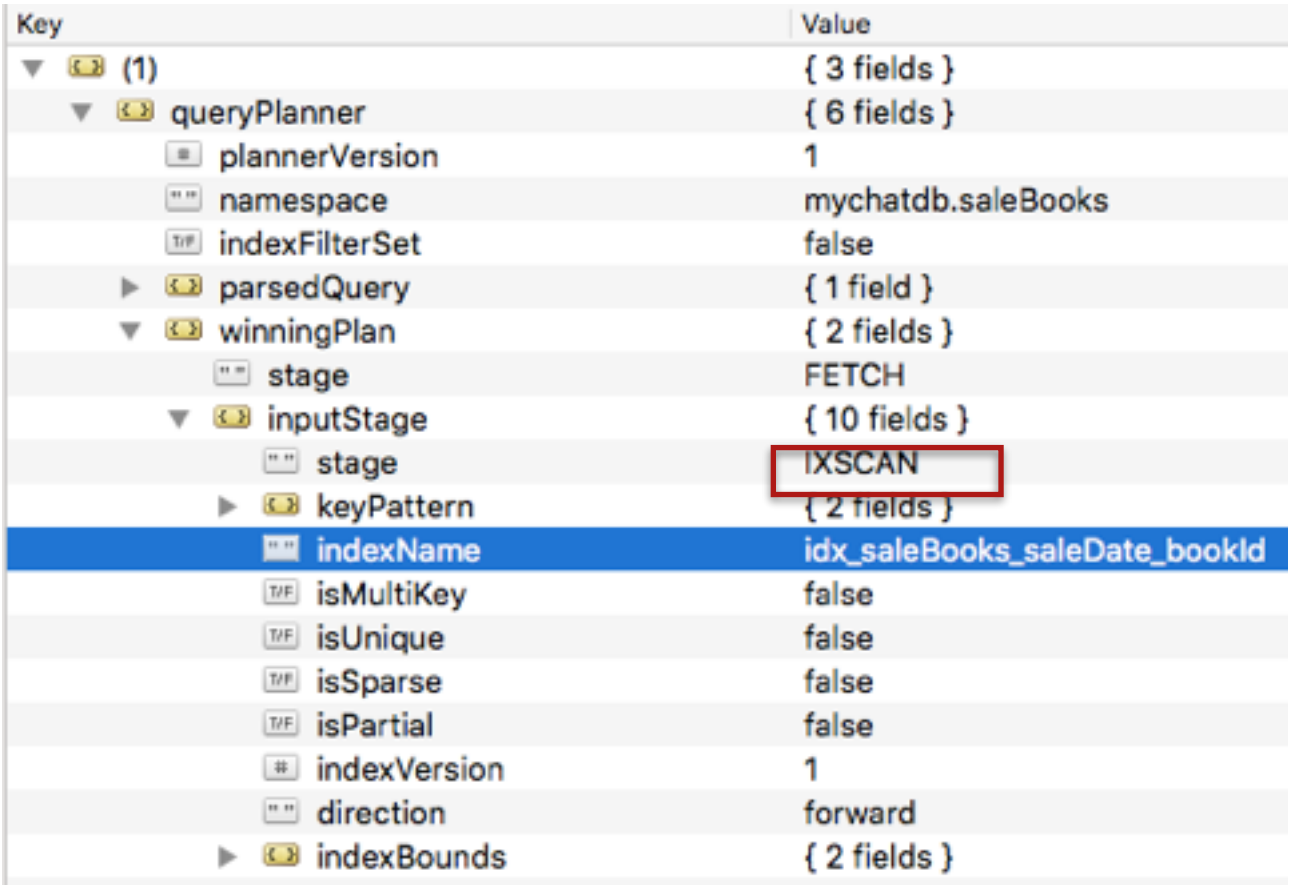
วิธีการสร้าง compound index

1. ที่ collection saleBooks สร้าง compound index ให้กับฟิลด์ saleDate และ bookId ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.saleBooks.createIndex({"saleDate":-1,"bookId":1},
{"name":"idx_saleBooks_saleDate_bookId"})
```

2. ใช้คำสั่ง explain ตรวจสอบการค้นหาข้อมูลจากฟิลด์ saleDate ให้ป้อนคำสั่ง ดังนี้

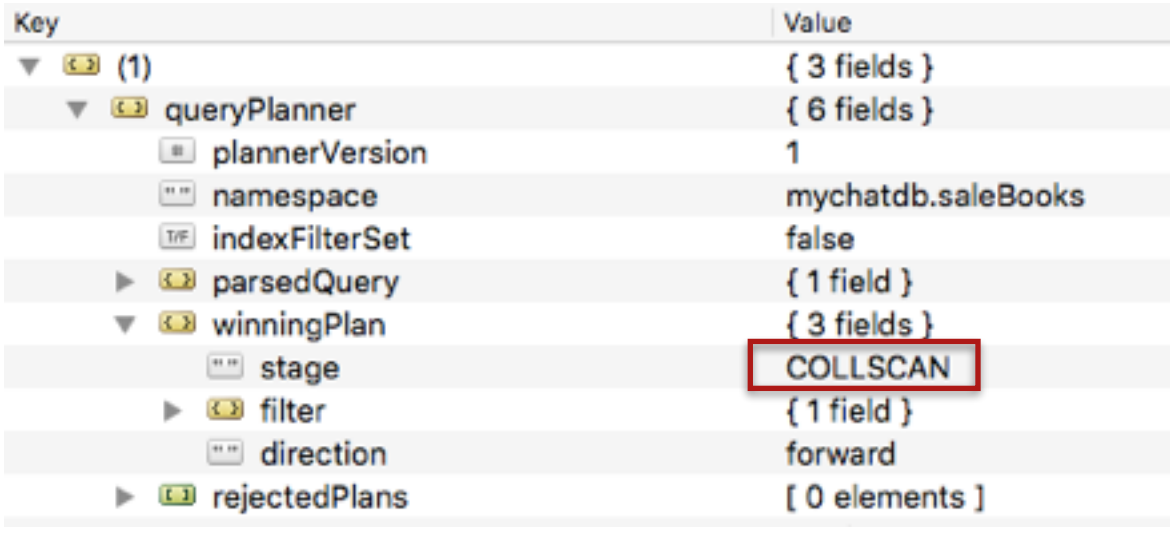
```
db.getCollection('saleBooks').find( {"saleDate": {"$gte":
new Date("2015-01-01")} }).explain()
```



Key	Value
(1)	{ 3 fields }
queryPlanner	{ 6 fields }
plannerVersion	1
namespace	mychatdb.saleBooks
indexFilterSet	false
parsedQuery	{ 1 field }
winningPlan	{ 2 fields }
stage	FETCH
inputStage	{ 10 fields }
stage	IXSCAN
keyPattern	{ 2 fields }
indexName	idx_saleBooks_saleDate_bookId
isMultiKey	false
isUnique	false
isSparse	false
isPartial	false
indexVersion	1
direction	forward
indexBounds	{ 2 fields }

3. ใช้คำสั่ง explain ตรวจสอบการค้นหาข้อมูลจากฟิลด์ bookId ให้ป้อนคำสั่ง ดังนี้

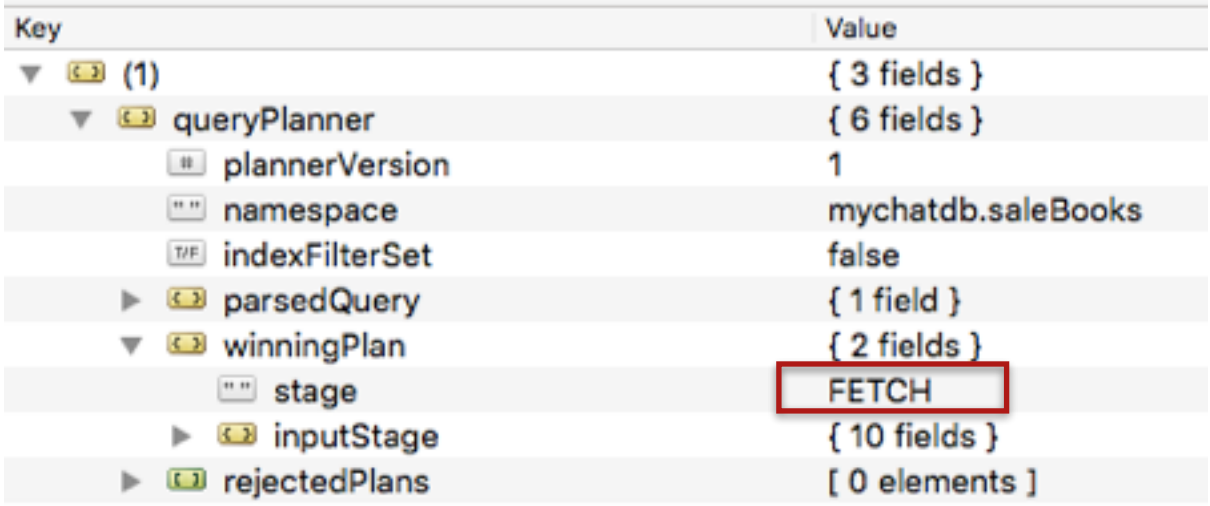
```
db.saleBooks.find({"bookId":"978-1-49194-892-7"}).explain()
```



Key	Value
(1)	{ 3 fields }
queryPlanner	{ 6 fields }
plannerVersion	1
namespace	mychatdb.saleBooks
indexFilterSet	false
parsedQuery	{ 1 field }
winningPlan	{ 3 fields }
stage	COLLSCAN
filter	{ 1 field }
direction	forward
rejectedPlans	[0 elements]

4. ใช้คำสั่ง explain ตรวจสอบการค้นหาข้อมูลจากฟิลด์ ทั้งจาก saleDate และ bookId ให้ป้อนคำสั่งดังนี้

```
db.getCollection('saleBooks').find(
     {"$and":[ {"saleDate": {"$gte": new Date("2015-01-01")} }
                ,{"bookId":"978-1-49194-892-7"}
              ]
     }).explain()
```



จากแบบฝึกหัดที่ได้ลองจะสังเกตว่าเราสร้าง compound index ขึ้นมาโดยใช้ 2 ฟิลด์คือ saleDate และ bookId โดยที่ saleDate เรียงลำดับจากมากไปน้อย เพราะส่วนใหญ่ในการค้นหาข้อมูลแบบวันต่อวันจะสนใจวันล่าสุดก่อน งาวางแผนสำหรับอนาคตจึงค้นหาข้อมูลในอดีตมาประกอบการวิเคราะห์ ส่วนฟิลด์ bookId เรียงลำดับจากน้อยไปมาก

- ในขั้นตอนที่ 2 หากเราค้นหาเฉพาะข้อมูลของฟิลด์ saleDate ผลปรากฏว่า mongodb ทำการแสกนข้อมูลจาก index ที่เราสร้างไว้ ก็จะทำให้เราได้ข้อมูลรวดเร็วกว่าการค้นหาแบบไม่มี index
- ในขั้นตอนที่ 3 หากเราค้นหาเฉพาะข้อมูลของฟิลด์ bookId ผลปรากฏว่า mongodb ทำการแสกนข้อมูลแบบ COLLSCAN คืออ่านทั้ง collection ทำให้ใช้เวลานานกว่าจะได้ผลลัพธ์ที่เราต้องการ
- สิ่งที่สังเกตได้จากการสร้าง compound index คือ ฟิลด์แรกจะเป็นคีย์หลักในการใช้ในการค้นหา ดังนั้นฟิลด์อื่นจะไม่เข้า index หากมีฟิลด์อื่นเกี่ยวข้อง ควรต้องระบุฟิลด์หลักเข้าไปด้วยเพื่อให้ค้นหาข้อมูลได้รวดเร็วขึ้นดังขั้นตอนที่ 4

* การใช้คำสั่ง explain เป็นสิ่งที่จำเป็นมากสำหรับนักพัฒนา เพราะจะทำให้เราทราบพฤติกรรมการค้นหาข้อมูลของ mongodb และปรับปรุงประสิทธิภาพ index ได้ถูกทาง

3.5 สร้าง Index แบบ Sparse Index

วัตถุประสงค์

เข้าใจวิธีการใช้คำสั่ง createIndex และหลักการของการสร้าง index แบบ sparse

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง createIndex และการเพิ่ม options สำหรับ sparse
- ทดสอบการใช้คำสั่ง createIndex และค้นหาข้อมูล

รูปแบบคำสั่ง createIndex แบบ sparse index

```
db.collection.createIndex(keys, {"sparse":true})
```

วิธีการสร้าง createIndex แบบ sparse index

1. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.customers.createIndex({"email":1},
{"name":"customer_idx_email","sparse":true})
```

2. ทดสอบใช้คำสั่ง find และ explain เพื่อดูว่าเมื่อค้นหา email mongodb เลือก index ที่เราสร้างไว้หรือไม่ ดังนี้

```
db.customers.find({"email":{"$regex":/co.uk/}}).explain()
```

3. ทดสอบใช้คำสั่ง find และ explain ด้วยหลายๆเงื่อนไข ดังนี้

```
db.customers.find({"$or":[ {"email":{"$regex":/co.uk/}},
{"sex":"F"}]}).explain()
```

จากข้อ 2 จะพบว่าหากเราทำการระบุเพียงฟิลด์ email mongodb จะทำการค้นหาจาก index ที่เราได้สร้างไว้ ส่วนในข้อ 3 เมื่อใส่เงื่อนไขเพิ่มเข้าไปอีกหนึ่งฟิลด์ ผลปรากฏว่าmongodb ไม่เลือกใช้ index ที่เราสร้างไว้ หากข้อมูลมีปริมาณมากจะเห็นระยะเวลาที่ใช้ในการค้นหาแตกต่างกันอย่างชัดเจน หากข้อมูลเยอะมากแล้วไม่ต้องการสร้าง index เพิ่มหลายคีย์วิธีการ query จะต้องแก้ไขลำดับ คือ

1. ค้นหาจากฟิลด์ที่มี index ก่อน
2. กรองข้อมูลตามเงื่อนไขเพิ่มเติมอีกครั้งเพื่อให้เหลือเฉพาะข้อมูลที่ต้องการ

```
var mails = db.customers.find({"email":{"$regex":/co.uk/}})
var mailsOnlyFemale  =mails.toArray().filter(function(doc){
    return doc.sex == "F";}
    );
    mailsOnlyFemale
```

* สำหรับ sparse index หากใช้ mongodb เวอร์ชั่น 3.2 ขึ้น mongoldb แนะนำว่าให้ใช้ partial index แทน

3.6 สร้าง Index แบบ Time to Live Index

Time to Live Index เป็น Index ที่นิยมนำไปใช้ในงานที่ต้องการให้มีการลบข้อมูลอัตโนมัติเมื่อหมดเวลา สามารถประยุกต์ใช้ได้หลายๆเรื่อง เช่น เก็บ Session ผู้เข้าใช้งานระบบ , เก็บราคาสินค้าของวันนี้สำหรับสินค้าประเภทที่มีการเปลี่ยนแปลงราคาทุกวัน, เก็บข้อมูลการประมูล ณ วันปัจจุบัน , ระบบโหวตวันต่อต่อวัน การใช้คุณสมบัตินี้จะช่วยให้ลดการเขียนโค้ดของระบบที่นำไปใช้ได้อย่างมาก

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง createIndex และหลักการของการสร้าง index แบบ Time to Live

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง createIndex และการเพิ่ม options สำหรับ Time to Live
- ทดสอบการค้นหาข้อมูลเมื่อหมดเวลา ข้อมูลหายไปจริงหรือไม่

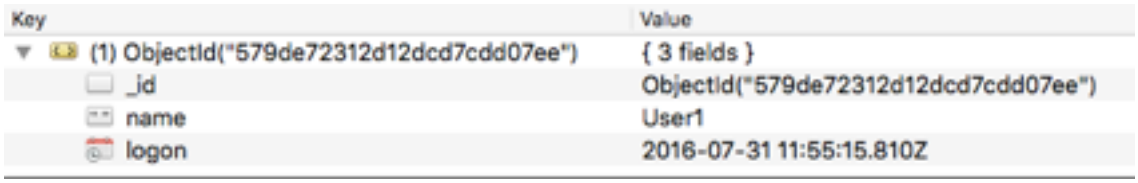
รูปแบบคำสั่ง createIndex แบบ Time to Live index

```
db.collection.createIndex(keys,{"expireAfterSeconds":true})
```

วิธีการสร้าง createIndex แบบ Time to Live index

1. ที่หน้าจอของ shell ให้ป้อนคำสั่ง ดังนี้

```
db.session.createIndex({logon:1},{expireAfterSeconds:60})
db.session.insert({name:"User1",logon: new Date()})
db.session.find()
```



2. หลังจากทำขั้นตอนที่ 1 ไป 60 วินาทีให้ค้นหาข้อมูลด้วยคำสั่ง ดังนี้

```
db.session.find({"name":"User1"})
```

```
Fetched 0 record(s) in 1ms
```

3-7 สร้าง Index แบบ Full-text search Index

ในบางระบบอาจมีความจำเป็นหรือเพิ่มคุณสมบัติให้กับแอพพลิเคชั่นสามารถค้นหาข้อมูลแบบ Full-text search ได้ ในการค้นหาคำที่ต้องการในหลายๆฟิลด์หากตารางนั้นมีหลายฟิลด์มากการเชื่อมเงื่อนไขด้วย การใช้พารามิเตอร์ $or อาจจะไม่สะดวกในการเขียน query ยาวๆ ระบบที่นิยมใช้คุณสมบัตินี้ ได้แก่ระบบ blog , document management , หนังสือออนไลน์ เป็นต้น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการสร้าง index แบบ Full-text search

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งเพื่อสร้าง index แบบ Full-text search
- ทดสอบการค้นหาข้อมูลในรูปแบบ Full-text search

รูปแบบคำสั่งการสร้าง index แบบ Full-text search

```
db.collection.createIndex({"field1" : "text","fieldN":"text"
```

ภายใน document ระบุฟิลด์ต่างๆที่ต้องการสร้าง index แบบ Full-text search

รูปแบบคำสั่งค้นหาข้อมูล แบบ Full-text search

```
db.collection.find( {"$text": { "$search": "string" } } )
```

วิธีการสร้าง สร้าง index แบบ Full-text search

1. ลองค้นหาข้อมูลโดยยังไม่เริ่มสร้าง index โดยค้นหาข้อมูลคำว่า จาก collection post ดังนี้

```
db.airports.find({$or:[{ident:/Drillmore Acres/},{name:/
Drillmore Acres/}]})
```

2. สร้าง index แบบ Full-text search ให้ฟิลด์ title และ comment ดังนี้

```
db.airports.createIndex({"ident":"text","type" :
"text","name":"text" })
```

3. ค้นหาข้อมูลหลังจากสร้าง Full-text search index ดังนี้

```
db.post.find( {$text: { $search: "string" } } ).count()
```

ส่วนตัวแล้วยังไม่ประทับใจกับการค้นหาข้อมูลด้วย Full-text search เพราะยังหาคำไทยไม่ค่อยเจอ โดยส่วนมากเวลาค้นหา จะสนใจเป็นคำๆมากกว่า หากค้นหาทั้งประโยคภาษาไทยเขียนติดกันหมดมีผลทำให้หาไม่เจอ ไม่เหมือนภาษาอังฤษที่มีการแยกศัพท์ คำเชื่อมต่างๆด้วยกัน สำหรับภาษาไทยใช้ regular expression ในการค้นหายังได้ผลมากกว่า ส่วนผู้ที่สนใจเรื่องการค้นหาที่ซับซ้อนลองใช้เครื่องมืออื่นๆที่เน้นเรื่องการค้นหาเป็นพิเศษจะทำได้ดีกว่า

3-8 สร้าง Index แบบ Geospatial Index

สำหรับ index ประเภทนี้ได้รับความนิยมมากในปัจจุบัน นำไปใช้ในเรื่องการเก็บข้อมูล ละติจูด ลองจิจูด เพื่อช่วยให้สามารถปักหมุดลงบนแผนที่หรือค้นหาระยะทางจากจุดหนึ่งไปยังจุดหนึ่งได้

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการสร้าง index แบบ Geospatial Index ที่ใช้สำหรับละติจูด ลองจิจูด และนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งเพื่อสร้าง index แบบ Geospatial Index
- ทดสอบการค้นหาข้อมูลในรูปแบบ Geospatial Index

รูปแบบคำสั่งการสร้าง index แบบ Geospatial Index

```
db.collection.createIndex( { <location field> : "geoHaystack" ,
                             <additional field> : 1 } ,
                           { bucketSize : <bucket value> } )
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	<location field>	string	ชื่อฟิลด์ที่ใช้ในการทำ index แบบ geoHayStack ซึ่งภายในฟิลด์นั้นจะต้องมีฟิลด์ย่อยที่เก็บค่าของ ละติจูดและลองจิจูดอยู่ด้วย
2	<additional field>	string	ฟิลด์ที่ระบุเพิ่มเติมเพื่อใช้เป็น index ด้วย เช่น เราอาจเก็บแยกกลุ่มชนิดของสถานที่ type:"resturant", type:"theater"
3	<bucket value>	integer	กำหนดขนาด bucket สำหรับจัดกลุ่มของ location ในส่วนนี้ยังหาข้อมูลและตัวอย่างที่ช่วยเสริมความเข้าใจให้มากขึ้นไม่ได้ จากการทดลองกำหนดค่ามากหรือน้อยยังไม่เห็นผลที่ต่าง แนะนำให้กำหนดค่าเป็น 1 ไปก่อน

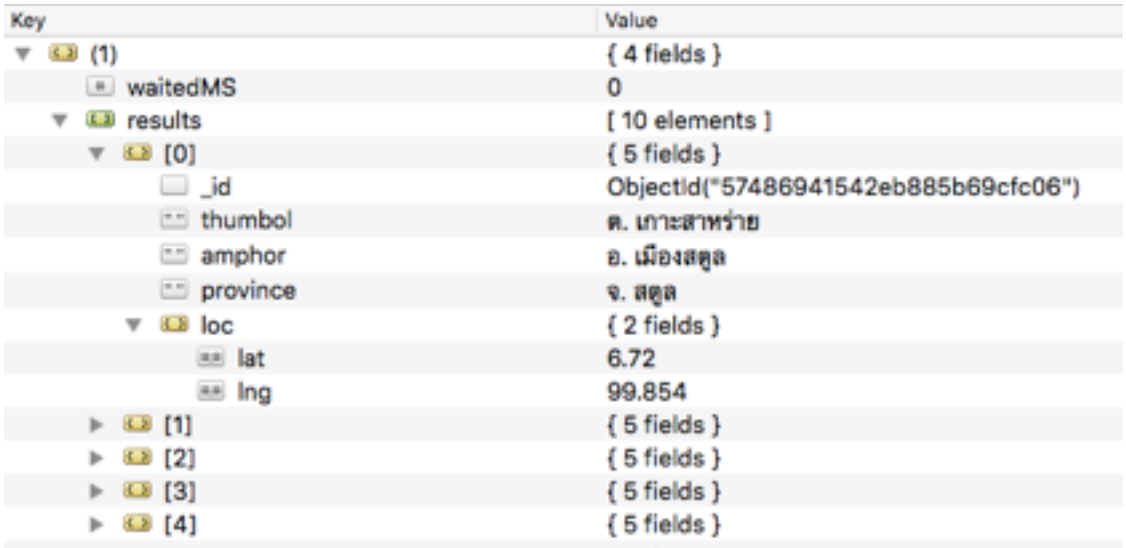
วิธีการสร้าง index แบบ Geospatial Index

1. ป้อนคำสั่งเพื่อสร้าง index แบบ Geospatial Index ดังนี้

```
db.places.createIndex( { loc : "geoHaystack","province":
1 } ,{ bucketSize : 1 } )
```

2. ค้นหาสถานที่ที่ใกล้กับจุดที่ต้องการ โดยหน่วยของ $maxDistance เป็นหน่วยของรัศมี ป้อนคำสั่งดังนี้

```
db.runCommand( { geoSearch : "places" ,
     search : { province: "จ. สตูล" } ,
     near : [6.546,99.706] ,
     maxDistance : 10,
     limit:10 } )
```



* หากไม่สร้าง index แบบ Spatial Index จะไม่สามารถใช้คำสั่ง geoSearch เพื่อหาข้อมูลที่ใกล้ในจุดที่เราต้องการได้

3-9 การแก้ไข Index

สำหรับการแก้ไข index ในฐานข้อมูลแทบทุบค่ายที่เคยใช้มาจะไม่มีการแก้ไข ถ้าจำเป็นต้องเปลี่ยนก็ทำการลบทิ้งแล้วสร้างใหม่ หรือ index เดิมก็ต้องหมั่นทำการ Rebuild อยู่เสมอหาก collection นั้นมีอัตราการเปลี่ยนแปลงข้อมูลสูง ไม่ว่าจะเป็นการเพิ่มข้อมูลใหม่ การแก้ไขข้อมูล การลบข้อมูล เพราะการทำ index คือการจัดระเบียบของข้อมูลให้ค้นหาให้ง่ายขึ้น หากไม่หมั่นดูแลรักษาเมื่อข้อมูลปรับเปลี่ยนบ่อยก็อาจทำให้ค้นหาไม่ได้รวดเร็วเหมือนตอนสร้างใหม่ๆ จากประสบการณ์มีบางฐานข้อมูลในเวอร์ชั่นเก่าๆเมื่อมีข้อมูลเปลี่ยนบ่อยและไม่ดูแลก็ทำให้มีเคสที่ค้นหาข้อมูลไม่เจอได้เช่นกัน

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการสร้าง Rebuild index และลบ index

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง Rebuild และลบ index
- ทดสอบการ Rebuild และลบ index

รูปแบบคำสั่งการ Rebuild index

```
db.collection.reIndex();
```

รูปแบบคำสั่งการ ลบ index

```
db.collection.dropIndex( "indexName" )
```

วิธีการ rebuild และลบ index

1. ป้อนคำสั่งเพื่อ rebuild index ดังนี้

```
db.post.reIndex();
```

2. ป้อนคำสั่งเพื่อลบ index ดังนี้

```
db.post.dropIndex("title_text_comment_text")
```

* แนะนำว่าคำสั่ง reIndex หรือการ Rebuild index ไม่ควรทำในขณะที่ฐานข้อมูลเปิดให้ทำการอ่านเขียน เพราะ mongodb จะทำการ lock การเขียนข้อมูลของ collection นั้น ถ้าจำเป็นต้องทำในระหว่างเปิดให้ระบบอื่นเข้าถึงได้ ควรทำเมื่อการค้นหาด้วย index ตัวนั้นช้าจริงๆ ให้ทำการลบ index และสร้างใหม่ใน background mode จะทำให้มีผลกระทบกับระบบโดยรวมน้อยกว่าการใช้คำสั่ง reIndex

---

3-10 การใช้คำสั่ง hint

ในกรณีที่มี index หลายๆตัวเกิดขึ้น โดยตั้งใจหรือไม่ตั้งใจ กรณีนี้อาจพบในงานที่ต้องส่งต่อผู้ดูแลระบบมาหลายคนแล้วคนดูแลล่าสุดอาจจะไม่กล้าเปลี่ยนแปลงอะไร หรือระบบอาจมีความจำเป็นต้องมี index หลายตัวจริงๆ บางครั้งการค้นหาข้อมูล mongodb ไม่เลือก index ตัวที่เราต้องการ สำหรับคำสั่ง hint นี้จะเป็นการเจาะจงว่าให้ใช้ index ตัวไหนในการค้นหา

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง hint

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่ง hint
- ทดสอบการใช้คำสั่ง hint

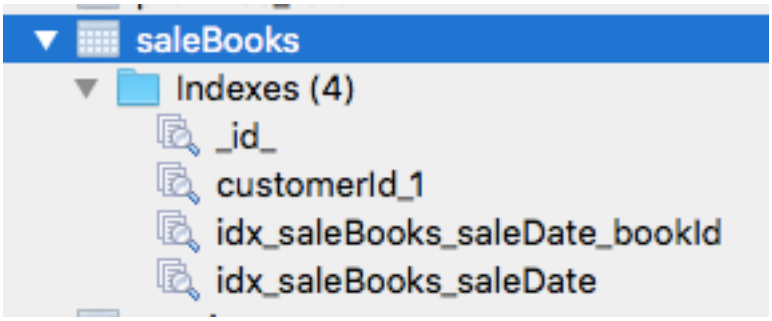
รูปแบบคำสั่ง hint

```
db.collection.find(query).hint("indexName")
```

วิธีการใช้คำสั่ง hint

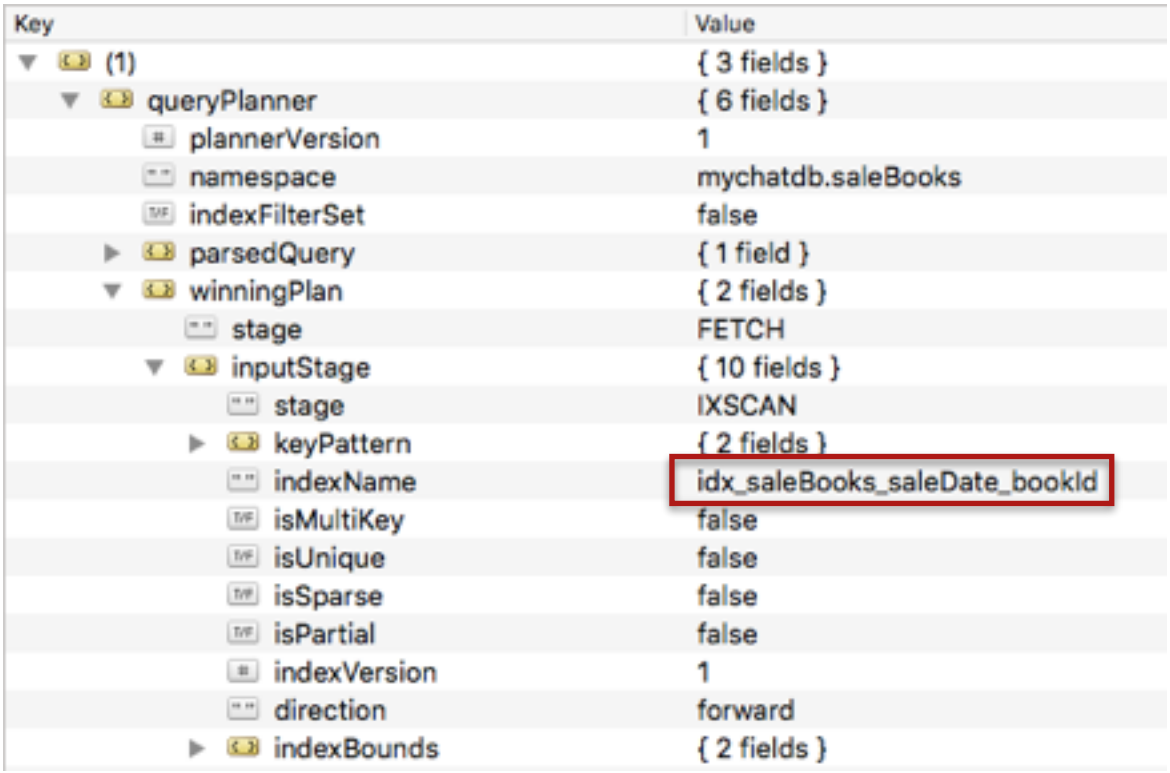
1. สร้าง index ที่ฟิลด์ saleDate ด้วยคำสั่ง ดังนี้

```
db.saleBooks.createIndex({"saleDate":-1},
{"name":"idx_saleBooks_saleDate"})
```



2. ตรวจสอบว่าเมื่อค้นหาด้วยฟิลด์ saleDate แล้ว mongodb เลือกใช้ index ตัวใด ด้วยคำสั่ง ดังนี้

```
db.getCollection('saleBooks').find( {"saleDate": {"$gte": new Date("2015-01-01")} }).explain()
```



3. จะเห็นว่า mongodb ยังเลือกใช้ index จากแบบฝึกหัดการทำ compound index ใช้คำสั่ง hint เพื่อระบุ index ที่เราต้องการใช้ ดังนี้

```
db.getCollection('saleBooks').find( {"saleDate": {"$gte": new Date("2015-01-01")} }).hint("idx_saleBooks_saleDate").count()
```

4. ทดลองค้นหาจาก index เดิม ดังนี้

```
db.getCollection('saleBooks').find( {"saleDate": {"$gte": new Date("2015-01-01")} }).count()
```

จะเห็นได้ว่า index ใหม่ที่ระบุเฉพาะเจาะจงเพียงฟิลด์เดียวสามารถค้นหาได้รวดเร็วกว่า แต่ในการใช้งานจริง การมี index หลายตัวมากเกินความจำเป็นไม่ใช่เรื่องที่ดีเสมอไป ถึงแม้ว่าเราต้องการค้นหาให้ได้รวดเร็วที่สุด แต่ก็ต้องแลกมาด้วยเนื้อที่ดิสก์ที่เสียไป ยิ่ง index มากยิ่งเสียเนื้อที่มาก และต้องหมั่นดูแลรักษาให้ดี เราควรมีข้อมูลด้วยว่ามีแอพลพิเคชั่นใด ฟังก์ชั่นใดใช้ index ที่เราสร้างขึ้นด้วย เพื่อให้บริหารจัดการได้มีประสิทธิภาพ และทราบถึงผลกระทบเมื่อต้องเปลี่ยนแปลงในอนาคต

3-11 การใช้คำสั่ง list index

วัตถุประสงค์
เข้าใจวิธีการเขียนคำสั่งเพื่อดึง index เป็นราย collection และทุกๆ collection ในฐานข้อมูลได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งระดับ collection getIndexes และ ระดับฐานข้อมูล getCollectionNames
- ทดสอบการใช้คำสั่ง getIndexes และ getCollectionNames

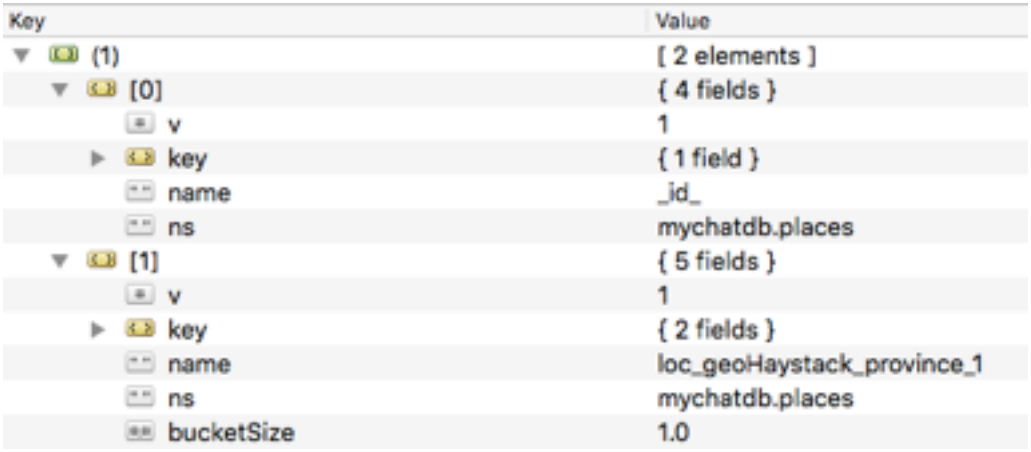
รูปแบบคำสั่ง getIndexes และ getCollectionNames

```
db.collection.getIndexes()
db.getCollectionNames()
```

วิธีการใช้คำสั่ง getIndexes และ getCollectionNames

1. ค้นหา index จาก collection places ด้วยคำสั่ง ดังนี้

```
db.places.getIndexes()
```



2. ค้นหา index จากทั้งฐานข้อมูล ด้วยคำสั่ง ดังนี้

```
db.getCollectionNames().forEach(function(collection) {
   indexes = db[collection].getIndexes();
   print("Indexes for " + collection + ":");
   printjson(indexes);
});
```

ค้นหา mongodb ไม่มีฟังก์ชั่นที่ดึงรายชื่อ index ทั้งฐานข้อมูล แต่สามารถประยุกต์ได้โดย ใช้คำสั่ง db.getCollecitonNames ได้ ซึ่งจะคืนค่ามาเป็น array จากนั้นนำมาเขียนโปรแกรมวนลูปเพื่อดู index ที่อยู่ภายใน collection

บทที่ 4 การทำงานกับ Aggregation Framework

Aggregation Framework คืออะไร ?

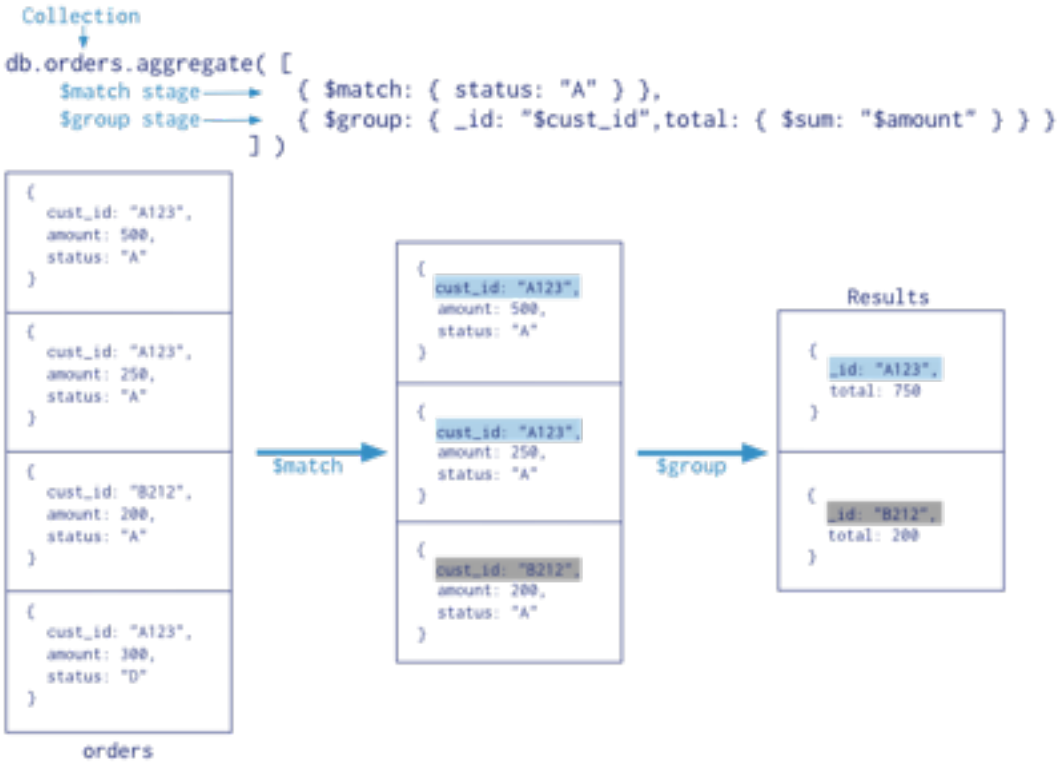
หากพูดถึง aggregation สำหรับผู้ใช้งานฐานข้อมูลย่อมได้ยินมานานแล้ว สำหรับ aggregation framework ของ MongoDB ก็มีจุดประสงค์เหมือนฐานข้อมูลอื่นๆคือมีขั้นตอนการทำงานหลักๆสองขั้นตอน คือ

- คัดกรองหรือจัดกลุ่มข้อมูลที่เหมือนกันตามเงื่อนไข
- นำข้อมูลที่คัดกรองมาคำนวณหรือประมวลผลให้ได้ตามผลลัพธ์ที่ต้องการ

ทำไมต้องใช้ Aggregation Framework ?

Aggregation มีความจำเป็นอย่างมากในการนำข้อมูลจำนวนมากมาวิเคราะห์ ซึ่งย่อมมีความเกี่ยวข้องกับการนำหลักสถิติมาใช้งาน ซึ่งเป็นเรื่องที่หลีกเลี่ยงไม่ได้เลยสำหรับนักพัฒนาหรือผู้ดูแลฐานข้อมูล ไม่ว่าระบบใดๆย่อมมักเจอคำถามเชิงสถิติในระบบที่เราดูแลอยู่เสมอ เช่น มีผู้ใช้ในระบบเท่าไร ยอดขายวันนี้ สัปดาห์นี้ เดือนนี้ ปีนี้ ปีก่อนเทียบกับปีนี้ เป็นอย่างไร สินค้าตัวใดขายดี ช่วงเวลาไหนขายดี ขายไม่ดี ซึ่งคำถามเหล่านี้ นักพัฒนาหรือผู้ดูแลฐานข้อมูลจำเป็นต้องมีความรู้ในการใช้งานในส่วนของ Aggregation Framework

Aggregation Pipeline



สำหรับ Aggregation จะใช้การประมวลผลโดยการดึงข้อมูลมาเก็บไว้ในหน่วยความจำ ซึ่ง mongodb จำกัดไว้ที่ 100 MB https://docs.mongodb.com/manual/core/aggregation-pipeline-limits/#memory-restrictions หากทำการใช้ aggregation กับข้อมูลมากกว่านี้ต้องกำหนด option allowDiskUse ให้เก็บไว้ในไฟล์ชั่วคราว แต่ก็จะทำให้ความเร็วในการประมวลผลลดลง ลำดับการทำงานของ Aggregation ตามภาพ คือ

- match ข้อมูลตามเงื่อนไข
- จัดกลุ่มข้อมูล

4-1 การใช้งานพารามิเตอร์ $match

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $match

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $match
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $match
สำหรับพารามิเตอร์ $match ใช้สำหรับระบุเงื่อนไขที่ใช้ในการกรองหรือจัดกลุ่มข้อมูลที่เหมือนกันตามเงื่อนไขที่เราระบุ

```
db.collection.aggregate({ $match: query} );
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	query	document	ระบุเงื่อนไขให้การกรองข้อมูลเพื่อจัดกลุ่มของข้อมูล

วิธีการใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $match

1. ป้อนคำสั่งเพื่อใช้ aggregation กรองข้อมูลที่มียอดขายมากกว่า 1500 แต่น้อยกว่า 1700 ดังนี้

```
db.saleBooks.aggregate({ "$match": {
     "price": {
          "$gt": 1500, "$lt": 1700 }
     }
  });
```

4-2 การใช้งานพารามิเตอร์ $group และ $project

จากตัวอย่างที่ผ่านมาการเราได้เรียนรู้การใช้พารามิเตอร์ $match จะเห็นได้ว่ามีความคล้ายคลึงกับการใช้พารามิเตอร์ในการ query ค้นหาข้อมูลในฟังก์ชั่น find แต่เมื่อใช้ร่วมกับ aggregation จะต้องใช้ภายใต้ $match แต่การใช้เพียงแค่พารามิเตอร์ $match นั้นจะไม่ตรงจุดประสงค์ของการใช้ aggregation เพราะจุดประสงค์หลักนั้นเราต้องการจัดกลุ่มข้อมูลและนำมาคำนวณเพื่อวิเคราะห์ข้อมูลและผลลัพธ์จากกลุ่มข้อมูลนั้น ดังนั้นแบบฝึกหัดนี้จะมีการใช้ $group และ $project เพื่อนำ aggregation ไปใช้ได้ตรงจุดประสงค์มากขึ้น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $group และ $project

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $group และ $project
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $group
สำหรับพารามิเตอร์ $group ใช้สำหรับจัดกลุ่มข้อมูลที่เหมือนกันตามเงื่อนไข และสามารถตั้งชื่อฟิลด์ใหม่หรือระบุพารามิเตอร์สำหรับการคำนวณค่าของกลุ่มข้อมูลได้

```
db.collection.aggregate({ $group: {"displayName":"$field"
     ,"displayName":{query}}} );
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	displayName	string	สามารถตั้งชื่อฟิลด์ใหม่เพื่อสื่อความหมายใหม่ได้
2	$field	string	ชื่อฟิลด์ที่ต้องการจัดกลุ่ม ซึ่งต้องระบุเครื่องหมาย $ นำหน้าชื่อฟิลด์เสมอ
3	query	document	ระบุเงื่อนไขเพื่อทำการคำนวณเพิ่มเติม เช่น {$sum:"$price"}

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project
สำหรับพารามิเตอร์ $project ใช้สำหรับเลือกว่าต้องการให้แสดงข้อมูลฟิลด์ใดบ้าง หรือจะแปลงฟิลด์ให้เป็นฟิลด์อื่น เช่น ผลลัพธ์การคูณของฟิลด์ totalPrice * qty เป็นต้น

```
db.collection.aggregate({ $project: {"displayName":"$field"
     ,"displayName":{query}}} );
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	displayName	string	สามารถตั้งชื่อฟิลด์ใหม่เพื่อสื่อความหมายใหม่ได้
2	$field	string	ชื่อฟิลด์ที่ต้องการแสดง ซึ่งต้องระบุเครื่องหมาย $ นำหน้าชื่อฟิลด์เสมอ
3	query	document	ระบุเงื่อนไขเพื่อทำการคำนวณเพิ่มเติม เช่น {$sum:"$price"}

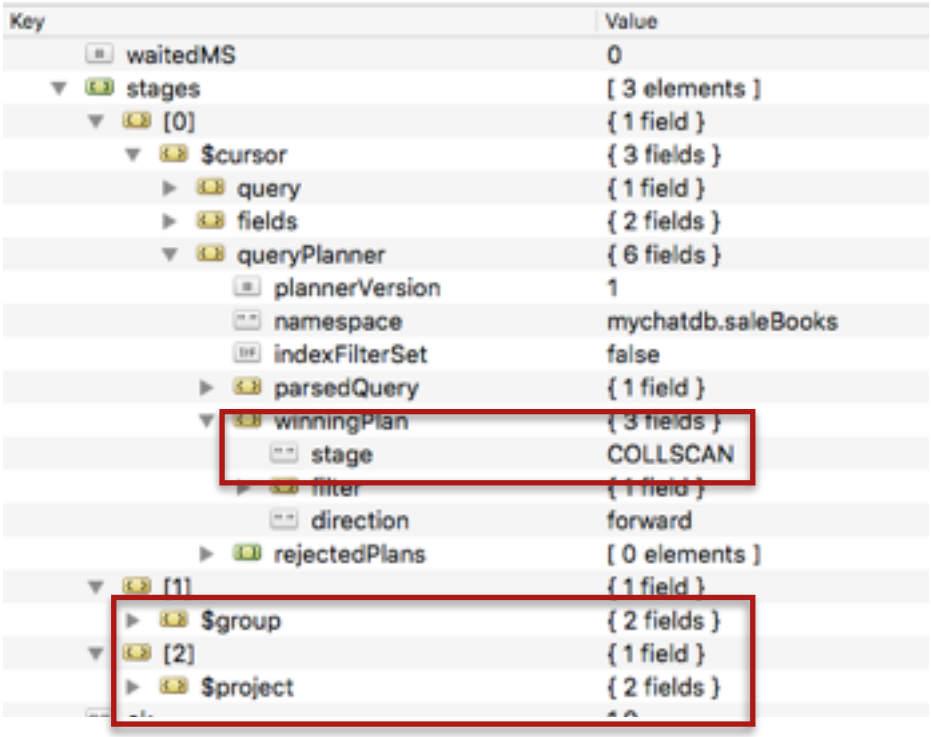
วิธีการใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $match

1. ป้อนคำสั่งเพื่อใช้ aggregation กรองข้อมูลที่มียอดขายมากกว่า 1500 แต่น้อยกว่า 1700 ดังนี้

```
var result =  db.saleBooks.aggregate(
     { $match: { price: { $gt: 1500, $lt: 1700 } } },
     { $group: { _id: null, count: { $sum: 1 } } },
     {$project:{"total":"$count","_id":0}}
     );
result
```

2. ป้อนพารามิเตอร์เพื่อดูลำดับการทำงานของชุดคำสั่ง ดังนี้

```
var result =  db.saleBooks.aggregate(
   [{ $match: { price: { $gt: 1500, $lt: 1700 } } }
   ,{ $group: { _id: null, count: { $sum: 1 } } }
   ,{$project:{"total":"$count","_id":0}}]
   ,{explain: true}
   );
result
```



ขั้นตอนการทำงานของฟังก์ชั่น

1. $match กรองข้อมูล price ตั้งแต่ 1501 - 1699 ซึ่งหากดู winningPlan แล้ว stage เป็น COLLSCAN ทำให้เกิดการอ่านทั้ง Collection ทำให้ใช้เวลานานหากข้อมูลเยอะ
2. $group ทำการจัดกลุ่มของข้อมูล อย่างน้อยต้องกำหนดฟิลด์ให้แก่ _id ซึ่งในตัวอย่างนี้ กำหนดให้ _id เป็นค่า null และเพิ่มฟิลด์ count: { $sum: 1 } ซึ่งใช้รวมค่า 1 แต่ละรายการ เพื่อนับจำนวน
3. $project แสดงข้อมูลฟิลด์ชื่อ total จากฟิลด์ count และไม่แสดง _id

4-3 การใช้งาน Pipeline Expression กับการคำนวณ

ในการดึงข้อมูลในหนึ่ง collection หรือจากหลายๆ collection เราอาจต้องการข้อมูลจากการคำนวณ บวก ลบ คูณ หาร จากฟิลด์หนึ่งกับอีกฟิลด์หนึ่ง สำหรับแบบฝึกหัดนี้เราจะทดลองใช้ aggregation กับการคูณระหว่างฟิลด์ที่เก็บราคาต่อหน่วยกับจำนวนสินค้าที่ขายได้

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับ $project เพื่อแสดงผลคูณระหว่างฟิลด์ 2 ฟิลด์

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $project และการคูณระหว่างฟิลด์
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project เพื่อทำการคูณ
สำหรับพารามิเตอร์ $group ใช้สำหรับจัดกลุ่มข้อมูลที่เหมือนกันตามเงื่อนไข และสามารถตั้งชื่อฟิลด์ใหม่หรือ

```
db.collection.aggregate({ $project: {"displayName":{"$multiply":
["$field1","$field2","$fieldN"]}}} );
```

ระบุพารามิเตอร์สำหรับการคำนวณค่าของกลุ่มข้อมูลได้
วิธีการใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $project เพื่อทำการคูณ

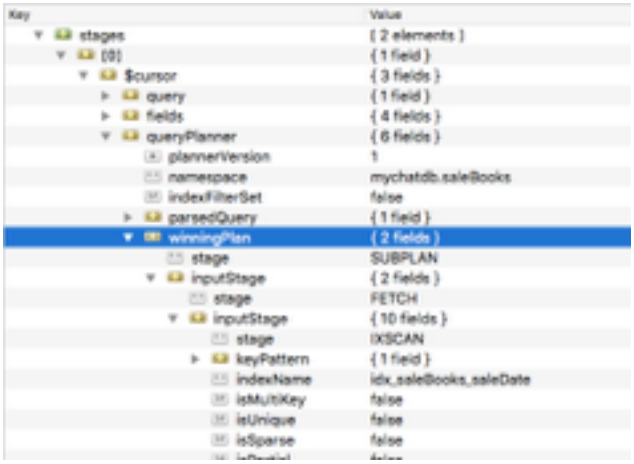
1. ป้อนคำสั่งเพื่อใช้ aggregation กรองข้อมูลที่มียอดขายระหว่างวันที่ 1/1/2015 - 31/12/2015 และนำฟิลด์ price คูณกับ qty ดังนี้

```
var result =  db.saleBooks.aggregate(
      [ { $match:
            {"$or":[
                  {"saleDate":{"$gte": new Date ("2015-01-01")}},
                  {"saleDate":{"$lte": new Date ("2015-12-31")}}
            ] }
      },
            {$group: { _id: "$bookId",unitPrice:{$sum:"$price"},
                        totalQty:{"$sum":"$qty"} }},
            {$project:{"totalPrice":"$unitPrice",
                  "totalQty":"$totalQty",
                  "totalSales": {"$multiply":["$unitPrice","$totalQty"]}}}
            ]
  );
  result
```

Key	Value
(1) 978-1-4493-9321-2	{ 4 fields }
_id	978-1-4493-9321-2
totalQty	3415.0
totalPrice	1687874.0
totalSales	5764089710.0
(2) 978-1-49194-892-7	{ 4 fields }
(3) 978-1-4571-0402-2	{ 4 fields }
(4) 978-1-78398-024-6	{ 4 fields }
(5) 978-1-118-83207-3	{ 4 fields }
(6) 978-1-119-07355-0	{ 4 fields }
(7) 978-1-4302-6502-3	{ 4 fields }
(8) 978-1-78588-619-5	{ 4 fields }
(9) 978-1-119-19431-6	{ 4 fields }
(10) 978-0-596-51668-0	{ 4 fields }

2. การทำงานกับข้อมูลเยอะๆอย่าลืมวิเคราะห์การทำงานของฟังก์ชั่นต่างๆที่เราใช้งานว่ามีการแสกนไปยัง index ที่สร้างไว้หรือไม่ เพิ่ม option explain ดังนี้

```
var result =  db.saleBooks.aggregate(
     [ { $match:
           {"$or":[
                       {"saleDate":{"$gte": new Date ("2015-01-01")}},
                       {"saleDate":{"$lte": new Date ("2015-12-31")}}
                 ]}
           },
           { $group: { _id: “$bookId",unitPrice:{$sum:"$price"},
                 totalQty:{"$sum":"$qty"} }},
           ],{explain: true}
           );
 result
```



ใน stages ที่ 0 ให้ตรวจสอบดูใน queryPlanner > winningPlan > inputStage > มีการเลือกใช้ index ที่เราสร้างขึ้นหรือไม่ หาก mongodb ทำการแสกนจาก index แล้วยังช้าอยู่ อาจจะต้องพิจารณาเรื่องการปรับจูน index หรือ performance หลายๆด้านอีกที

* ในการใช้ options explain หากใช้พารามิเตอร์ $project ต้องเป็นเงื่อนไขที่ไม่ซับซ้อน แต่ควรเอา $project ออกเพื่อช่วยให้สามารถวิเคราะห์ภาพรวมของลำดับการทำงานของ mongodb

---

4-4 การใช้งาน Pipeline Expression กับ String

สำหรับข้อมูลแบบ String นั้น เราอาจมีการเก็บในรูปแบบที่หลากหลาย เช่น การเก็บที่อยู่เป็นแบบ array หรือ embleded หรือ เก็บเป็นฟิลด์แยก ชื่อกับนามสกุลออกจากกัน แต่เมื่อต้องนำมาแสดงผลอาจจะมีการรวมฟิลด์กัน จึงต้องใช้ฟังก์ชั่นในการรวมฟิลด์ ไม่ต้องไปเก็บเพิ่มให้เปลืองเนื้อที่เกินความจำเป็น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับการนำข้อมูลแบบ string มาแสดงผลแบบต่างๆ ร่วมกับพารามิเตอร์การจัดการ string เช่น $substr, $concat ,$toUpper, $toLower

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $project และการใช้งานร่วมกับพารามิเตอร์ string ต่างๆ เช่น $substr, $concat ,$toUpper, $toLower
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ string

```
db.collection.aggregate(
     {$project:{"displayName":{"$parameter":document,array}
   }
})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	displayName	string	สามารถตั้งชื่อฟิลด์ใหม่เพื่อสื่อความหมายใหม่ได้
2	$parameter	string	ชื่อพารามิเตอร์ในแบบฝึกหัดนี้จะเน้นพารามิเตอร์ที่เกี่ยวกับ string เท่านั้น เช่น $substr, $concat ,$toUpper, $toLower
3	document,array	document,array	รูปแบบการรับพารามิเตอร์ของ $parameter ซึ่งบางฟังก์ชั่นอาจจะรับเป็น document บางฟังก์ชั่นอาจรับเป็น array

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ $substr

substr มีให้ใช้ทุกๆภาษาและฐานข้อมูลกันเลยทีเดียว เป็นฟังก์ชั่นที่ใช้ตัดคำจากตำแหน่งที่ต้องการไปอีกกี่ตัวอักษร

```
db.collection.aggregate(
     {$project:{"displayName":
          {"$substr":["$field",startPosition,length]}
     }})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$field	string	ชื่อฟิลด์ที่ต้องการแสดง ซึ่งต้องระบุเครื่องหมาย $ นำหน้าชื่อฟิลด์เสมอ
2	startPosition	integer	ตำแหน่งเริ่มต้นที่ต้องการให้แสดงผล ค่าตำแหน่งเริ่มต้นที่ 0
3	length	integer	จำนวนตัวอักษรตั้งแต่ startPosition ไปอีกกี่ตัว

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ $concat
concat เป็นฟังก์ชั่นที่ใช้สำหรับเชื่อมคำระหว่าง string หลายๆตัวให้เป็นคำเดียวกัน ซึ่งสามารถระบุคำจากฟิลด์ที่มีอยู่หรือพิมพ์คำที่ต้องการเองได้

```
db.collection.aggregate(
     {$project:{"displayName":{"$concat":
["$field1","string","$fieldN"]}
  }})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$field	string	ชื่อฟิลด์ที่ต้องการแสดง ซึ่งต้องระบุเครื่องหมาย $ นำหน้าชื่อฟิลด์เสมอ
2	string	string	ข้อความที่ต้องการระบุเอง

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ $toUpper
toUpper เป็นฟังก์ชั่นที่ใช้สำหรับปรับ string ให้เป็นตัวอักษรพิมพ์ใหญ่ ใช้ได้สำหรับภาษาอังกฤษ

```
db.collection.aggregate(
     {$project:{"displayName":{"$toUpper":{"$field1"}}
  }})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$field	string	ชื่อฟิลด์ที่ต้องการแสดง ซึ่งต้องระบุเครื่องหมาย $ นำหน้าชื่อฟิลด์เสมอ

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ $toLower
toUpper เป็นฟังก์ชั่นที่ใช้สำหรับปรับ string ให้เป็นตัวอักษรพิมพ์เล็ก ใช้ได้สำหรับภาษาอังกฤษ

```
db.collection.aggregate(
     {$project:{"displayName":{"$toLower":{"$field1"}}
  }})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$field	string	ชื่อฟิลด์ที่ต้องการแสดง ซึ่งต้องระบุเครื่องหมาย $ นำหน้าชื่อฟิลด์เสมอ

วิธีใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ string

1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $project กับ string ทั้ง 4 ฟังก์ชั่น ดังนี้

```
 db.customers.aggregate({$project:{"address":{"$substr":
["$address",5,10]}}})

  db.customers.aggregate({$project:{"fullname":{"$concat":
["$firstName"," ","$lastName"]}}})

  db.customers.aggregate({$project:{"FIRSTNAME":
{"$toUpper":"$firstName"}}})

  db.customers.aggregate({$project:{"firstname":
{"$toLower":"$firstName"}}})
```

4-5 การใช้งาน Pipeline Expression กับ Date

สำหรับข้อมูลแบบ Date ก็นิยมใช้ในการนำมาวิเคราะห์ข้อมูลต่อในหลายๆเรื่อง เช่น ทำการจัดกลุ่มตามช่วงวันเวลา ดูรายงานการขาย รายวัน รายสัปดาห์ รายเดือน รายปี ดูช่วงเวลาที่ขายดี ขายไม่ดี หรือในการทำปฏิบัติงานในองค์กรก็จะมีการนำเวลามาคำนวณหาเวลาที่ใช้ในการทำงานเพื่อเทียบกับ SLA (Service Level Agreement) เป็นเวลาที่ในองค์กรได้กำหนดไว้ในแต่ละงานว่าใช้ระยะเวลามาตรฐานไม่เกินเท่าไร เพื่อตรวจสอบดูว่าพนักงานใช้เวลาในการทำงานเกินมาตรฐานหรือไม่ ในการเก็บข้อมูล Date แนะนำว่าไม่ว่าฐานข้อมูลใดก็ตามส่วนใหญ่ก็จะรองรับข้อมูลประเภท Datetime อยู่แล้ว บางระบบอาจเก็บเป็น string หรือ ตัวเลข หากทำเช่นนั้นโอกาสที่ฟิลด์นั้นจะมี *dirty data สูง ซึ่งในความเป็นจริง
* dirty data คือข้อมูลที่ไม่ถูกชนิดตามที่ต้องเป็น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับ Date เพื่อนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $project และการใช้งานร่วมกับ Date
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ date

```
db.collection.aggregate(
     {$project:{"displayName":{"$parameter":"$field"}
  }
})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$parameter	string	ชื่อพารามิเตอร์ที่เกี่ยวกับข้อมูลชนิด Date ที่ต้องการใช้
	• $year • $month • $dayOfMonth • $hour • $minute • $second • $millisecond • $dayOfYear • $dayOfWeek • $week	number	• ชื่อพารามิเตอร์ที่ใช้แสดงค่าปี • ชื่อพารามิเตอร์ที่ใช้แสดงค่าเดือน • ชื่อพารามิเตอร์ที่ใช้แสดงค่าวันของเดือน • ชื่อพารามิเตอร์ที่ใช้แสดงค่าชั่วโมง • ชื่อพารามิเตอร์ที่ใช้แสดงค่านาที • ชื่อพารามิเตอร์ที่ใช้แสดงค่าวินาที • ชื่อพารามิเตอร์ที่ใช้แสดงค่าหนึ่งในพันวินาที • ชื่อพารามิเตอร์ที่ใช้แสดงค่าวันที่ของปี 1-365 หรือ 366 • ชื่อพารามิเตอร์ที่ใช้แสดงค่าวันที่ของสัปดาห์ 1-7 • ชื่อพารามิเตอร์ที่ใช้แสดงค่าสัปดาห์ที่ 1-52 หรือ 53

วิธีคำสั่ง aggregation ร่วมกับพารามิเตอร์ $project กับ date

1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $project กับ date เพื่อดูค่า $week, $dayOfWeek, $dayOfMonth,$dayOfYear ดังนี้

```
db.saleBooks.aggregate({
   $project: {
        "saleWeek": {
            "$week": "$saleDate"
        },
        "saleDayOfWeek": {
            "$dayOfWeek": "$saleDate"
        },
        "saleDayOfMonth": {
            "$dayOfMonth": "$saleDate"
        },
        "saleDayOfYear": {
            "$dayOfYear": "$saleDate"
        }
   }
 })
```

Key	Value
(1) ObjectId("573ec7e5bec88ff1af511e84")	{ 5 fields }
_id	ObjectId("573ec7e5bec88ff1af511e84")
saleWeek	38
saleDayOfWeek	2
saleDayOfMonth	24
saleDayOfYear	267

2. ในหลายๆระบบจะมีมุมมองในการวิเคราะห์และอยากรู้ว่าวันที่ขายหรือวันที่ได้ปฏิบัติงานเป็นระยะเวลากี่วันจนถึงปัจจุบัน ทำการคำนวณได้ ดังนี้

```
db.saleBooks.aggregate({
   $project: {
        totalDay: {
             $divide: [{
                  $subtract: [new Date(), '$saleDate']
             }, 24 * 1000 * 60 * 60]
        }
   }
 })
```

Key	Value
(1) ObjectId("573ec7e5bec88ff1af511e84")	{ 2 fields }
_id	ObjectId("573ec7e5bec88ff1af511e84")
totalDay	3243.59571835648
(2) ObjectId("573ec7e5bec88ff1af511e85")	{ 2 fields }
_id	ObjectId("573ec7e5bec88ff1af511e85")
totalDay	3934.39479475694
(3) ObjectId("573ec7e5bec88ff1af511e86")	{ 2 fields }
(4) ObjectId("573ec7e5bec88ff1af511e87")	{ 2 fields }

4-6 การใช้งาน Pipeline Expression กับการเปรียบเทียบ

สำหรับการเปรียบเทียบค่าภายในหนึ่งฟิลดีหรือระหว่างฟิลด์หนึ่งกับอีกฟิลด์หนึ่ง ก็มีความนิยมนำค่าเหล่านั้นไปใช้งานต่อในแอพพลิเคชั่นอื่นๆ หรือเพื่อแปลความหมายอื่นๆ เช่น นำข้อมูลยอดขายมาเปรียบเทียบกับต้นทุนที่ใช้ไป เพื่อแปลผลว่ารายการนั้น กำไร เท่าทุน หรือขาดทุน เป็นต้น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับการเปรียบเทียบค่า เพื่อนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $cmp และ $strcasecmp
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $cmp และ $strcasecmp

```
db.collection.aggregate(
     {$project:{"displayName":{"$parameter":
     ["$field1","$field2"]}
  }
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	$parameter	string	ชื่อพารามิเตอร์ที่เกี่ยวกับฟังก์ชั่นเปรียบเทียบค่า
	• $cmp • $strcasecmp	boolean	• ชื่อพารามิเตอร์ที่ใช้ในการเปรียบเทียบค่า มากกว่า (1) เท่ากับ (0) น้อยกว่า (-1) ระหว่างฟิลด์หนึ่งกับอีกฟิลด์หนึ่ง • ชื่อพารามิเตอร์ที่ใช้ในการเปรียบเทียบค่าตัวอักษร - ฟิลด์แรกมีค่ามากกว่าฟิลด์ที่สอง (1) - ฟิลด์แรกมีค่าเท่ากับฟิลด์ที่สอง (0) - ฟิลด์แรกมีค่าน้อยกว่าฟิลด์ที่สอง (-1)

วิธีใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $cmp และ $strcasecmp

1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $project กับ $cmp และ $strcasecmp ดังนี้

```
db.saleBooks.aggregate({$project:{
          isEqual:{$cmp:["$qty","$price"]}
     }
})

db.saleBooks.aggregate({$project:{
          isEqual:{$cmp:["$qty","$qty"]}
     }
})

db.saleBooks.aggregate({$project:{
          isEqual:{$cmp:["$price","$qty"]}
     }
})

db.books.aggregate({$project:{
          title:{"$strcasecmp":["$category","javascript"]}
     }
})
```

4-7 การใช้งาน Unwind Expression

สำหรับฟังก์ชั่น Unwind มีประโยชน์มากสำหรับจัดรูปแบบของฟิลด์ที่เก็บข้อมูลแบบ array ให้อยู่ในรูปแบบข้อมูลมิติเดียว เพื่อให้สามารถนำมาวิเคราะห์และจัดกลุ่มข้อมูลในรูปแบบ aggregate ได้ง่ายขึ้น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับฟังก์ชั่น $unwind เพื่อนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $unwind
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $unwind

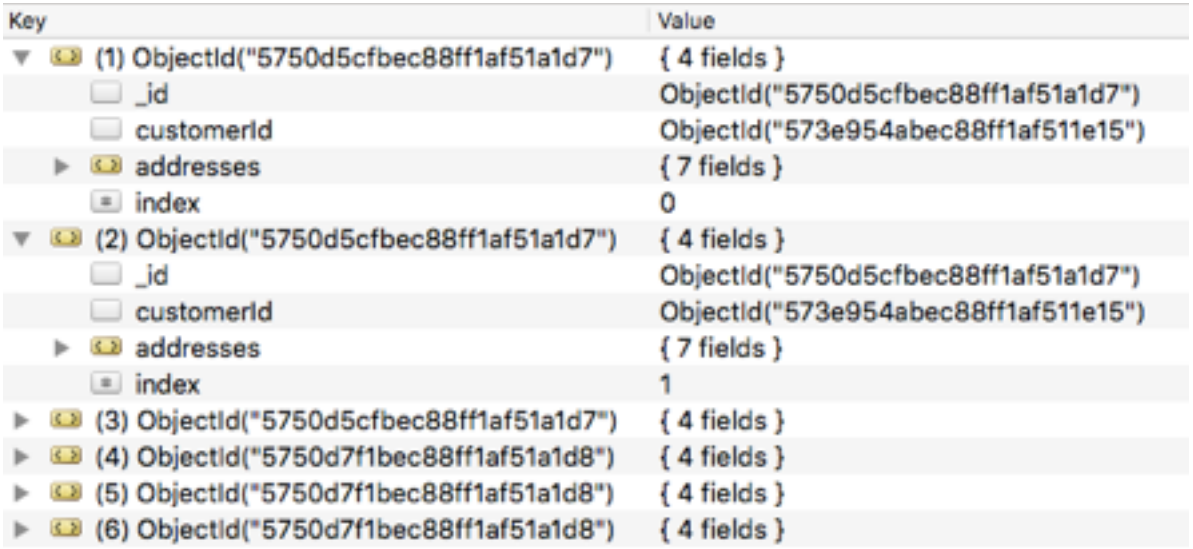
```
db.collection.aggregate(
 {
  $unwind:
    {
      path: <field path>,
      includeArrayIndex: <string>,
      preserveNullAndEmptyArrays: <boolean>
    }
}
})
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	`path`	string	ชื่อฟิลด์ที่เก็บค่า array ที่ต้องการแปลงรูปแบบออกจาก array ให้อยู่ในเอกสารเดียว
2	`includeArrayIndex`	string	ชื่อฟิลด์ใหม่ที่ต้องการแสดงเลข index ของ array สำหรับชื่อฟิลด์ไม่สามารถขึ้นต้นด้วย $
3	`preserveNullAndEmptyArrays`	boolean	ไม่แสดงฟิลด์นั้นถ้าหากมีค่าเป็น nulll หรือ empty

วิธีใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $unwind

1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $unwind ดังนี้

```
db.customerAddresses.aggregate(
{"$unwind" : {"path":"$addresses",
     "includeArrayIndex":"index",
     "preserveNullAndEmptyArrays":true}
})
```



2. จากชุดคำสั่งด้านบนจะได้ผลลัพธ์จากเดิมที่มี 2 รายการเป็น 6 รายการ ข้อมูลแบบเดิมออกแบบไว้เก็บข้อมูลลูกค้าเป็นรายการ ซึ่งลูกค้าหนึ่งคนสามารถมีที่อยู่ได้หลายๆที่อยู่จึงเก็บไว้ในรูปแบบ array

---

4-8 การใช้งาน Sort Expression

สำหรับฟังก์ชั่น Sort ใช้สำหรับเรียงลำดับข้อมูลตามฟิลด์ที่เราต้องการ โดยปกติจะเรียงตาม index ที่ตรงกับเงื่อนไขของ query นั้น หากไม่เข้าเงื่อนไข inde ก็จะเรียงตาม _id หากไม่มีการระบุ

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับ $sort เพื่อนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $sort
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $sort

```
{ $sort: { <field1>: <sort order>,
 <field2>: <sort order> ... } }
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	field	string	ชื่อฟิลด์ที่ต้องการทำการ sort
2	<sort order>	number	ลำดับในการ sort กำหนดค่าเป็นตัวเลข -1 เรียงจากมากไปน้อย 1 เรียงจากน้อยไปมาก

วิธีใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $sort

1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $sort ดังนี้

```
db.saleBooks.aggregate(
     { $match: { "customerId": { $eq: "1615101111399" } } }
     ,{ $group: {
          "_id": {"book":"$bookId","customer":"$customerId"}
          ,"totalPrice":{$sum:"$price"},"totalQty":
{"$sum":"$qty"} }
     }
     ,{$project:{ "totalPrice": {"$multiply":
["$totalPrice","$totalQty"]} }}
     ,{$sort:{"totalPrice":-1}}
     );
```

Key	Value
(1) { 2 fields }	{ 2 fields }
_id	{ 2 fields }
totalPrice	32169045174.0
(2) { 2 fields }	{ 2 fields }
_id	{ 2 fields }
totalPrice	31190428752.0
(3) { 2 fields }	{ 2 fields }
(4) { 2 fields }	{ 2 fields }
(5) { 2 fields }	{ 2 fields }
(6) { 2 fields }	{ 2 fields }
(7) { 2 fields }	{ 2 fields }
(8) { 2 fields }	{ 2 fields }
(9) { 2 fields }	{ 2 fields }
(10) { 2 fields }	{ 2 fields }

4-9 การใช้งาน Limit and Skip Expression

สำหรับฟังก์ชั่น Limit ใช้สำหรับจำกัดจำนวนรายการที่ต้องการแสดงผล การแสดงผลข้อมูลจำนวนมากมีผลต่อการโอนย้ายข้อมูลจากระบบเครือข่ายกับการเก็บไว้ในหน่วยความจำของเครื่องที่ทำการแสดงผล ในกรณีที่ไม่มีความจำเป็นต้องดูข้อมูลทั้งหมด หรือต้องการโหลดข้อมูลจำนวนมาก การใช้ limit และ skip มีส่วนช่วยรักษาเสถียรภาพของฐานข้อมูลได้

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับ $limit และ $skip เพื่อนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ aggregation ฟังก์ชั่นร่วมกับพารามิเตอร์ $limit และ $skip
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $limit

```
{ $limit: <positive integer> }
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	<positive integer>	integer	จำนวนรายการที่ต้องการแสดงผล

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $skip

```
{ $skip: <positive integer> }
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	<positive integer>	integer	จำนวนเอกสารที่ต้องการข้ามไป ไม่แสดง เช่น หากระบุ $skip เท่ากับ 5 ผลลัพธ์ที่ได้คือจะแสดงเอกสารตั้งแต่ตำแหน่งที่ 6 เป็นต้นไป

วิธีการใช้คำสั่ง aggregation ร่วมกับพารามิเตอร์ $skip และ $limit ร่วมกัน

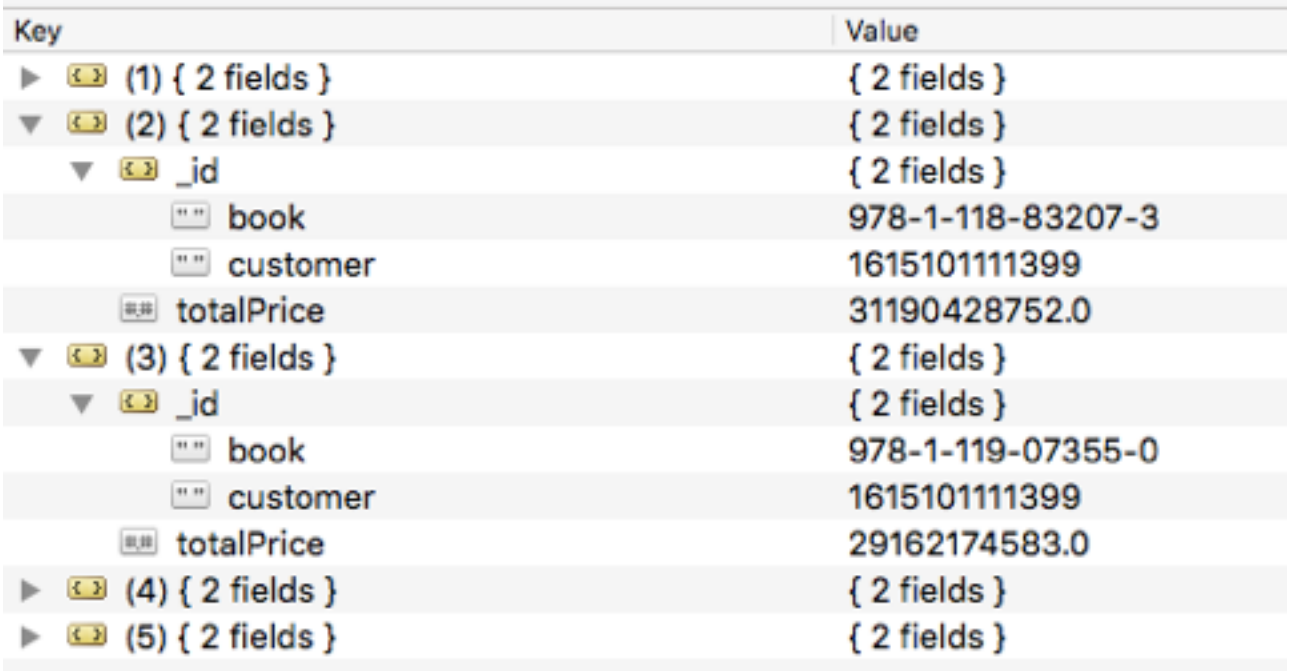
1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $limit ดังนี้

```
db.saleBooks.aggregate(
     { $match: { customerId: { $eq: "1615101111399" } } }
     ,{ $group: { _id:
                    {"book":"$bookId","customer":"$customerId"},
          totalPrice:{$sum:"$price"},totalQty:{"$sum":"$qty"} }}
     ,{$project:{ "totalPrice": {"$multiply":
          ["$totalPrice","$totalQty"]}  }}
     ,{$sort:{"totalPrice":-1}}
     ,{$limit:5}
     );
```

2. เพิ่มพารามิเตอร์ $skip ดังนี้

```
db.saleBooks.aggregate(
     { $match: { customerId: { $eq: "1615101111399" } } }
     ,{ $group: { _id:
                    {"book":"$bookId","customer":"$customerId"},
          totalPrice:{$sum:"$price"},totalQty:{"$sum":"$qty"} }}
     ,{$project:{ "totalPrice": {"$multiply":
                              ["$totalPrice","$totalQty"]}  }}
     ,{$sort:{"totalPrice":-1}}
     ,{$limit:5}
     ,{$skip:2}
     );
```

เมื่อเพิ่มพารามิเตอร์ $skip เข้าไปแล้วลองรันคำสั่งอีกครั้งจะพบว่า ได้ผลลัพธ์ เมื่อเทียบกับคำสั่งแรก ตั้งแต่รายการที่ 3 เป็นต้นไป



4-10 การใช้งาน $lookup กับ aggregation เพื่อทำการ join

สำหรับฟังก์ชั่น Join เป็นคุณสมบัติที่ mongodb เพิ่มเข้ามาในเวอร์ชั่น 3.2 เดิมทีนั้น mongodb นั้นเน้นจุดยืนให้ออกแบบฐานข้อมูลแบบ Denormalize เป็นหลัก แต่สุดท้ายแล้วก็ต้องปรับเปลี่ยนเพิ่มฟังก์ชั่นในการ join เข้ามาจนได้ เพราะในโลกของข้อมูลขนาดใหญ่การนำข้อมูลมาวิเคราะห์ ยังมีความจำเป็นต้องเข้าใจความสัมพันธ์ของข้อมูลและการเชื่อมโยงกันของข้อมูลเพื่อลดการจัดเก็บข้อมูลมากเกินความจำเป็น

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง $lookup เพื่อทำการ join และนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ $lookup
- ทดสอบการใช้งาน

รูปแบบคำสั่ง aggregation ร่วมกับพารามิเตอร์ $lookup

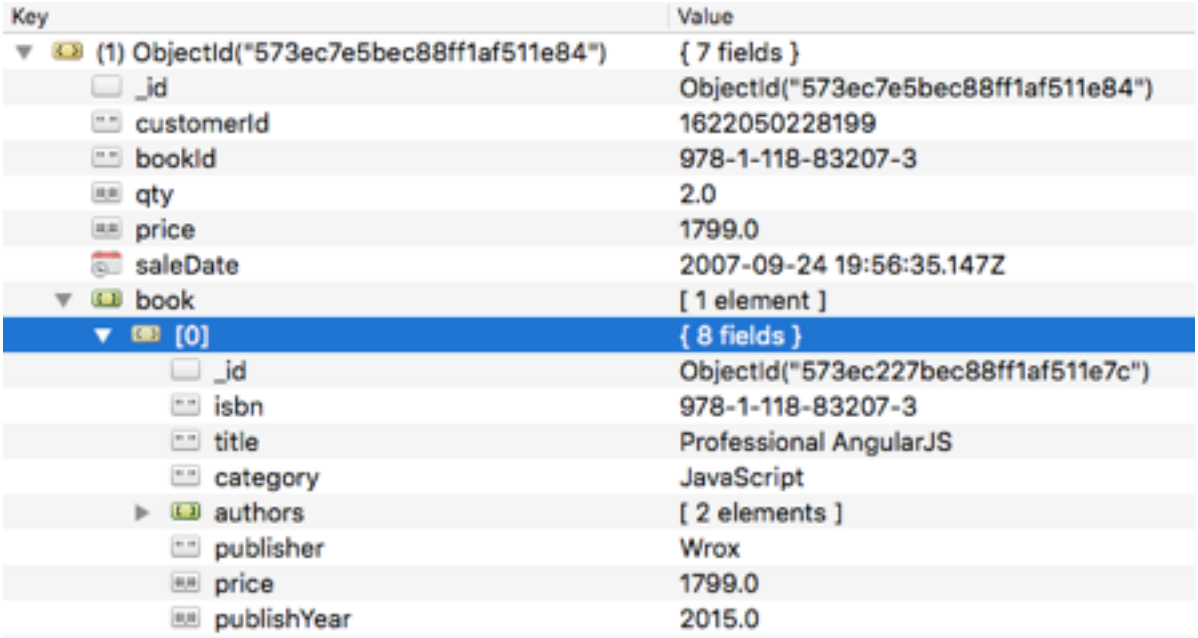
```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from"
collection>,
       as: <output array field>
     }
}
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	from	string	ชื่อ collection ที่ต้องการทำการ join ข้อมูลภายในฟิลด์ ส่วนใหญ่แล้วจะเป็น colleciton ที่ออกแบบไว้เป็น master data
2	localField	string	ชื่อฟิลด์จาก collection ที่เรียกใช้ใน aggregation
3	foreignField	document	ชือฟิลด์จาก collection from
4	as	string	ชื่อฟิลด์ที่ต้องการให้แสดงข้อมูลจาก collection from

วิธีการใช้ aggregation ร่วมกับพารามิเตอร์ $lookup

1. ป้อนคำสั่งเพื่อใช้ aggregation ร่วมกับพารามิเตอร์ $lookup ดังนี้

```
db.saleBooks.aggregate([
   {$lookup:{from:"books"
       ,localField:"bookId"
       ,foreignField:"isbn"
       ,as:"book"
       }}
   ])
```



Key	Value
(1) ObjectId("573ec7e5bec88ff1af511e84")	{ 7 fields }
_id	ObjectId("573ec7e5bec88ff1af511e84")
customerId	1622050228199
bookId	978-1-118-83207-3
qty	2.0
price	1799.0
saleDate	2007-09-24 19:56:35.147Z
book	[1 element]
[0]	{ 8 fields }
_id	ObjectId("573ec227bec88ff1af511e7c")
isbn	978-1-118-83207-3
title	Professional AngularJS
category	JavaScript
authors	[2 elements]
publisher	Wrox
price	1799.0
publishYear	2015.0

2. เมื่อทำการเขียนคำสั่งสำหรับทำการ join ข้อมูลจาก collection books จะถูกนำมาแสดงในฟิลด์ book ที่เรากำหนดไว้ในรูปแบบ array

---

4-11 การใช้งาน Map Reduce

สำหรับฟังก์ชั่น map reduce มีจุดประสงค์คล้ายคลึงกับ aggregate แต่มีความยืดหยุ่นสูง สามารถเขียนฟังก์ชั่นเพื่อแปลงผลที่ต้องการได้เอง เหมาะกับงานที่มีการกระจายตัวของข้อมูลที่มีการขยายระบบโดยการทำ Sharding

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่งในการใช้ mapreduce เพื่อนำไปประยุกต์ใช้งานได้

แนวทางปฏิบัติ

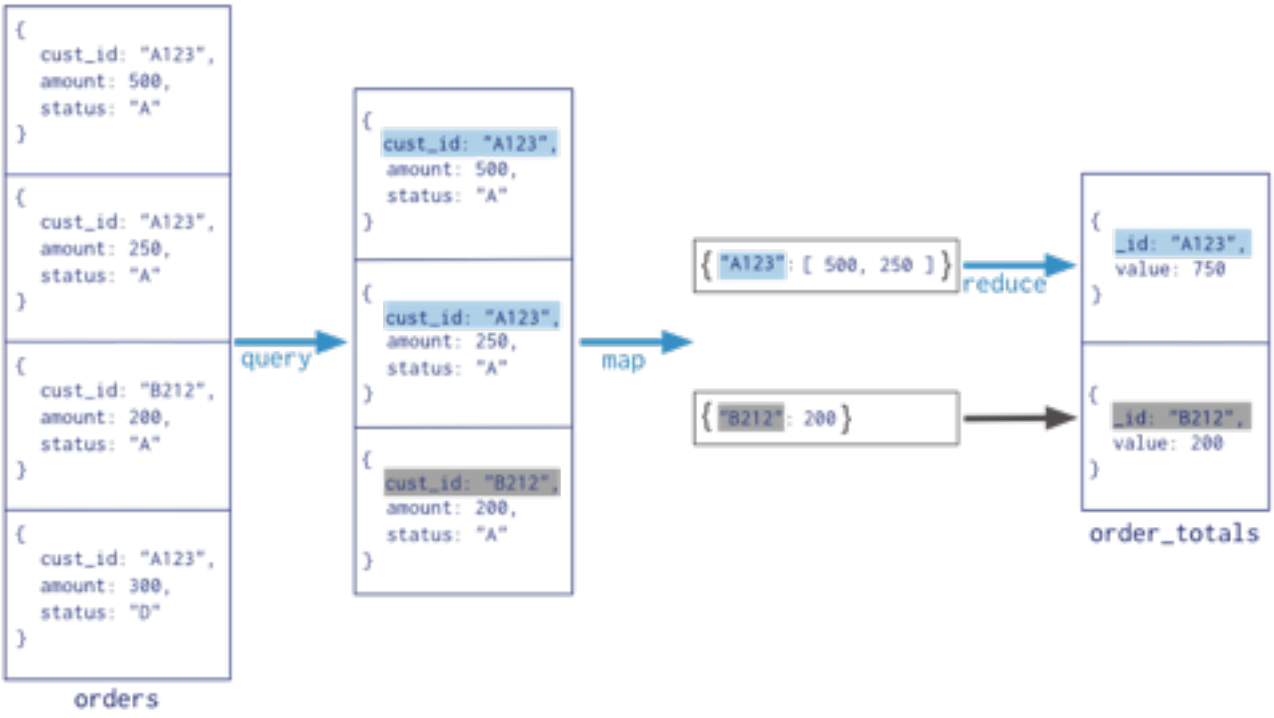
- ศึกษารูปแบบคำสั่งในการใช้ mapreduce
- ทดสอบการใช้งาน

รูปแบบคำสั่ง mapreduce

```
db.collection.mapReduce(
                         <map>,
                         <reduce>,
                         {
                           out: <collection>,
                           query: <document>,
                           sort: <document>,
                           limit: <number>,
                           finalize: <function>,
                           scope: <document>,
                           jsMode: <boolean>,
                           verbose: <boolean>,
                           bypassDocumentValidation: <boolean>
                         }
                       )
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	`map`	function	ฟังก์ชั่นที่ใช้ในการจัดกลุ่มข้อมูลที่ตรงตามเงื่อนไข
2	`reduce`	function	ฟังก์ชั่นที่ใช้ในการลดรูปของกลุ่มข้อมูลให้เหลือจำนวนรายการที่ไม่ซ้ำกันตามเงื่อนไขที่ทำการจัดกลุ่ม เช่น มีการ map ข้อมูลชื่อสินค้า สินค้าแต่ละตัวมียอดขายในแต่ละวันหลายรายการ หากต้องการผลรวมของยอดขายของสินค้าแต่ละวันก็ทำการเขียนฟังก์ชั่น เพื่อหาผลรวม และจะเหลือรายการสินค้านั้นเพียงรายการเดียวพร้อมผลลัพธ์
3	`options`	document	พารามิเตอร์เพิ่มเติมที่ต้องการใช้ใน mapreduce
3.1	`bypassDocumentValidation`	boolean	ทำการตรวจสอบความถูกต้องของรูปแบบเอกสารในกรณีที่มีการ insert
3.2	`out`	string or document	ระบุผลลัพธ์ของ Map Reduce ใน collection สำหรับการระบุแบบ document จะมีรายละเอียดปลีกบ่อยอีกพอสมควรใช้สำหรับกรณีที่มีการทำ sharding
3.3	`query`	document	เงื่อนไขในการกรองข้อมูล

```
Collection
   ↓
db.orders.mapReduce(
          map      ──►  function() { emit( this.cust_id, this.amount ); },
          reduce   ──►  function(key, values) { return Array.sum( values ) },
                        {
          query    ──►     query: { status: "A" },
          output   ──►     out: "order_totals"
                        }
                    )
```



#	พารามิเตอร์	ชนิด	รายละเอียด
3.4	`sort`	document	เงื่อนไขในการเรียงลำดับข้อมูล
3.5	`limit`	number	กำหนดจำนวนรายการที่ต้องการแสดง
3.6	`finalize`	function	ฟังก์ชั่นภายใน finalize จะได้รับค่า key และ values ที่ได้ผ่านการทำ reduce แล้ว หากต้องการปรับเปลี่ยนค่าอะไรให้เขียนภายในขั้นตอนสุดท้าย
3.7	`scope`	document	การกำหนดตัวแปรแบบ Global variable ที่สามารถใช้ได้ทั้งในส่วนของ map , reduce และ finalize เช่น scope: {"title":"จังหวัด"}

#	พารามิเตอร์	ชนิด	รายละเอียด
3.8	jsMode	boolean	ค่าเริ่มต้นคือ false ในการใช้ mapreduce ในเฟสของการ Map MongoDb จะทำการแปลงเป็นรูปแบบ BSON ก่อน และจะแปลงกลับเป็น JavaScript เมื่อเฟส reduce ซึ่งข้อดีคือสามารถประมวลกับข้อมูลขนาดใหญ่ได้ แต่จะมีความช้ากว่าหากกำหนดออบชั่นเป็น true เพราะมีการเก็บลง disk ชั่วคราว ส่วนการกำหนด jsMode เป็น true จะไม่แปลงเป็น BSON ประมวลผลได้เร็วกว่าแต่มีข้อจำกัดคือ map คีย์ที่ไม่ซ้ำกันได้น้อยกว่า 500,000 รายการ
3.9	verbose	boolean	กำหนดให้แสดงเวลาที่ใช้ในการประมวลผล ค่าเริ่มต้นกำหนดเป็น true

ลำดับการทำงานของฟังก์ชั่น mapreduce

1. query กรองข้อมูลตามเงื่อนไข
2. map ทำการจัดกลุ่มข้อมูลตาม เงื่อนไขการ map
3. reduce ลดจำนวนรายการและแปลงผลลัพธ์การคำนวณต่างไปเก็บไว้ใน collectioin ใหม่ที่ระบุ ใน output

วิธีการใช้คำสั่ง mapreduce

1. ป้อนคำสั่งเพื่อใช้ mapreduce กับ collection place เพื่อดูว่าในแต่ละจังหวัดมีกี่ตำบล ป้อนคำสั่งดังนี้

```
db.places.mapReduce(
      function() {
            var key = this.province.substring(2);
            emit(key,1);
      },

      function(key, values) {
            return   Array.sum(values);
      },
      {
            out: "province_total"
      }
)
```

Key	Value
(1)	{ 8 fields }
result	province_total
timeMillis	384.0
counts	{ 4 fields }
input	7768
emit	7768
reduce	687
output	77
ok	1.0
_o	{ 4 fields }
_keys	[4 elements]
[0]	result
[1]	timeMillis
[2]	counts
[3]	ok
_db	{ 2 fields }
_coll	{ 4 fields }

2. เมื่อทำการประมวลผล ผลลัพ์จะถูกนำไปเก็บไว้ใน result ที่เรากำหนดไว้ในคำสั่งคือ province_total เราสามารถดูผลลัพธ์ได้จาก collection province_total ดังนี้

```
db.province_total.find()
```

Key	Value
(1) กระบี่	{ 2 fields }
_id	กระบี่
value	112.0
(2) กาญจนบุรี	{ 2 fields }
_id	กาญจนบุรี
value	95.0
(3) กาฬสินธุ์	{ 2 fields }
(4) กำแพงเพชร	{ 2 fields }
(5) ขอนแก่น	{ 2 fields }
(6) จันทบุรี	{ 2 fields }

บทที่ 5 Security และการออกแบบฐานข้อมูล

mongodb จะยอมให้เข้าถึงฐานข้อมูลเพื่อบริหารจัดการทุกอย่างได้ตั้งแต่เริ่มต้น ทำให้มีข้อดีตรงที่ผู้ที่อยากศึกษาในช่วงเริ่มต้นสามารถใช้งานได้ทุกอย่างไม่ติดขัดเรื่องสิทธิการใช้งาน แต่ก็เป็นข้อเสียที่ตามทีหลังเมื่อนำระบบขึ้นผู้ใช้งานหลงลืมที่จะกำหนดค่าสิทธิการใช้งานและเพิ่มความปลอดภัยต่างๆให้กับระบบ ทำให้มีข่าวออกมาช่วงหนึ่งว่าข้อมูลที่เก็บใน mongodb เกิดการรั่วไหลจำนวนมาก การหลงลืมหรือไม่ตั้งใจกำหนดสิทธิความปลอดภัยให้ฐานข้อมูลก็เป็นสาเหตุหนึ่งเช่นกัน ดังนั้นต้องกำหนดสิทธิและความปลอดภัยตั้งแต่ช่วงทดสอบก่อนนำระบบขึ้นใช้งานจริง

5-1 สร้าง User

เมื่อเริ่มต้นใช้งาน mongodb จะสามารถใช้งานได้โดยไม่จำเป็นต้องมี user ใดเลย ซึ่งต่างจากฐานข้อมูลอื่นๆที่ต้องสร้าง user ขึ้นมาก่อน ดังนั้นก่อนนำขึ้นใช้งานจริง ต้องไม่ลืมที่จะสร้าง user เพื่อเพิ่มความปลอดภัยให้กับระบบ

วัตถุประสงค์
เข้าใจวิธีการใช้คำสั่ง ในการสร้าง user และประยุกต์ใช้งานได้

แนวทางปฏิบัติ

- ศึกษารูปแบบคำสั่งในการใช้ db.createUser
- ทดสอบการใช้งาน

รูปแบบคำสั่ง db.createUser

```
db.createUser(user, writeConcern)
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	`user`	document	ข้อมูลที่จำเป็นของผู้ใช้ เช่น userName,password,role
2	`writeConcern`	document	ในส่วนนี้จะไม่ได้กล่าวถึง

รูปแบบข้อมูล user

```
{ user: "<name>",
  pwd: "<cleartext password>",
  customData: { <any information> },
  roles: [
    { role: "<role>", db: "<database>" } | "<role>",
    ...
  ]
}
```

#	พารามิเตอร์	ชนิด	รายละเอียด
1	user	string	ชื่อผู้ใช้งาน
2	pwd	string	รหัสผ่าน
3	customData	document	ข้อมูลอะไรก็ได้ที่ต้องการเก็บเพิ่มเติม เช่น รหัสพนักงาน,เบอร์,โทรศัพท์ ,อีเมล์ เป็นต้น
4	roles	array	สามารถระบุเป็นค่าว่างได้ โดยค่าที่ระบุเป็น array string ซึ่งสามารถระบุเป็นค่าของ build-in roles และ user-defined role

วิธีการใช้คำสั่ง db.createUser

1. ป้อนคำสั่ง db.createUser เพื่อสร้าง user ดังนี้

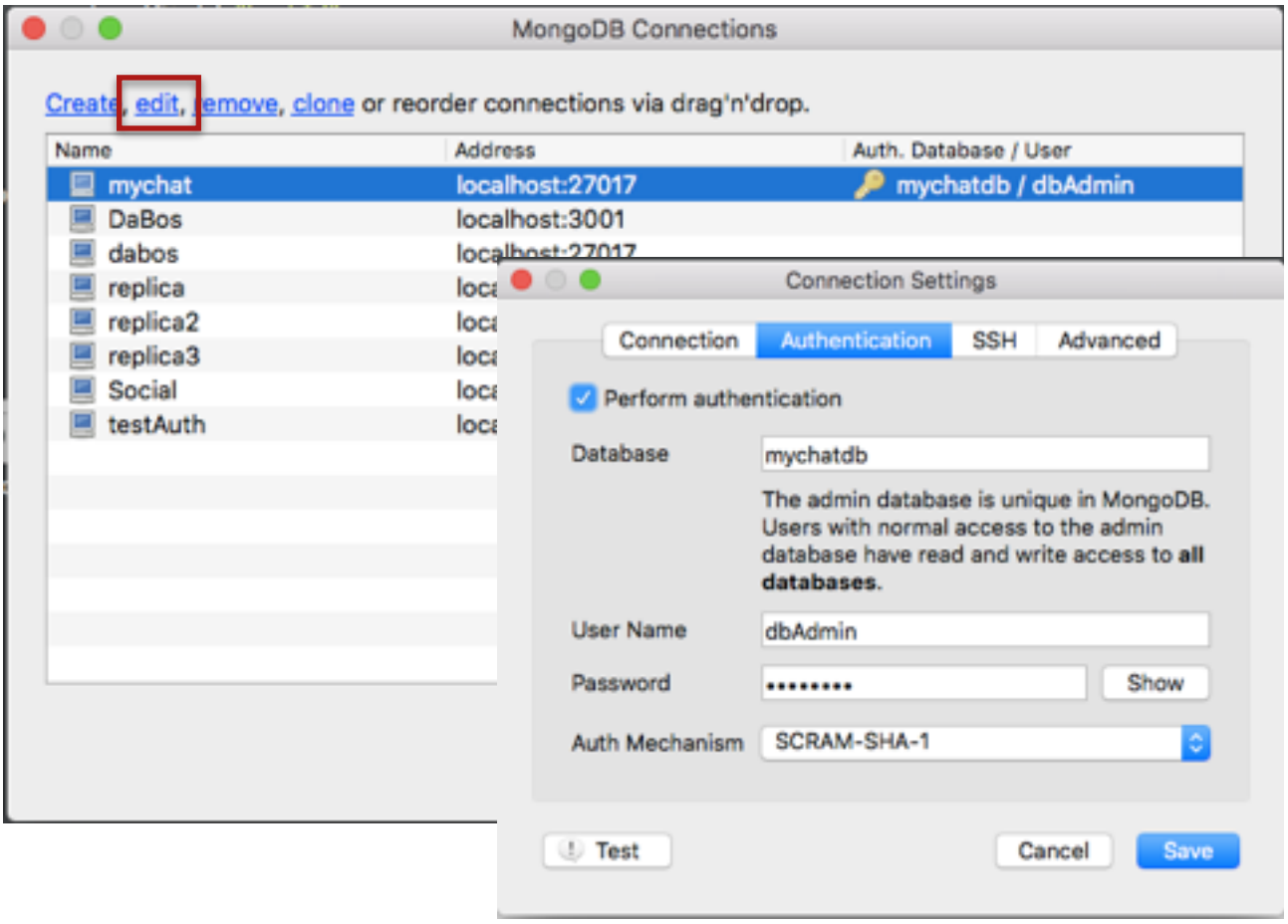
```
db.createUser(
   {
     user: "dbAdmin",
     pwd: "p@ssw0rd",
     roles: [ "readWrite", "dbAdmin" ]
   }
)
```

```
Successfully added user: { "user" : "dbAdmin", "roles" :
[ "readWrite", "dbAdmin" ] }
```

2. เข้าไปที่หน้า terminal แล้วทำการ restart mongoldb ดังนี้

```
> ./mongod --dbpath "<Database Path>" --auth
```

3. เมื่อกลับมาใช้ robomongo จะไม่สามารถเข้าถึงฐานข้อมูลได้ ต้องทำการกำหนด user และ password ใน configuration



สำหรับ role ต่างๆที่ mongodb สร้างไว้ให้ศึกษาเพิ่มเติมได้ที่
https://docs.mongodb.com/manual/reference/built-in-roles/#built-in-roles

Application Design

Normalize และ Denormalize

เราควรออกแบบฐานข้อมูลให้เป็นแบบ Normalize หรือ Denormalize ดี ?

ในการใช้งานจริงอาจจะพบเห็นการออกแบบทั้งสองแบบ ซึ่งมีข้อดีข้อเสียต่างกัน จะให้ทั้งระบบเป็นแบบใดแบบหนึ่งเท่านั้นก็อาจจะทำได้ยากเพราะส่วนใหญ่ทุกระบบทุกองค์กรล้วนต้องทำงานภายใต้ข้อจำกัดต่างๆ จึงต้องเลือกข้อดีข้อเสียต่างกันไป

Normalise
สำหรับผู้ที่ได้เรียนวิชาการออกแบบฐานข้อมูล และผู้คุลกคลีอยู่กับฐานข้อมูล คงได้ศึกษาและออกแบบมาจนคุ้นเคยแล้วสำหรับการออกแบบฐานข้อมูล การทำ Normalize จุดประสงค์เพื่อทำการแยกตารางที่มีความสัมพันธ์ในระดับย่อยๆ ในลักษณะ 1 to Many ออกไปเก็บไว้อีกตารางหนึ่ง เพื่อให้บริหารจัดการให้มีความอิสระหรือขึ้นต่อกันได้ง่ายขึ้น ในฐานข้อมูลแบบเชิงสัมพันธ์เรามักนิยมเก็บรหัสของคีย์หลักในตารางแม่ไว้ในตารางย่อย การออกแบบในลักษณะนี้จะทำได้ค่อนข้างยากใน mongodb เวอร์ชั่นเก่าๆที่ยังไม่รองรับการ join สำหรับผู้ใช้ mongodb เวอร์ชั่นเก่าๆนิยมออกแบบในลักษณะ embleded ให้เก็บความสัมพันธ์ใน document อันเดียว

Relational Database

Customer

Customer	CustomerID
Jones	1
Lucas	2
Stevens	3

Product

Product	ProductID
TV	1
Mobile	2
Tablet	3

Order

OrderID	CustomerID	OrderDate
201608-001	1	2016-08-01
201608-002	2	2016-08-02
201608-003	3	2016-08-03
201608-004	2	2016-08-04

OrderDetails

OrderDetailsID	OrderID	ProductID	UnitPrice	Qty	TotalPrice
1	201608-001	1	9000	1	
2	201608-001	2	15000	1	
3	201608-001	3	20000	1	
4	201608-002	2	15000	2	

NOSQL Relation

Product-1

Product	ProductID	ProductAttributeID
TV	1	1
Mobile	2	2
Tablet	3	3

ProductAttribute

ProductAttributeID	Model	Height	Weight
1	LCD 30 “	30 Inch	1.45 kg

Denormalize

การทำ Denormalise คือการไม่ต้องแยกเก็บข้อมูลออกเป็นหลายตาราง เก็บไว้ตารางเดียวเลย ก็จะทำให้ค้นหาข้อมูลได้เร็วขึ้น ข้อเสียที่ตามมาคือการใช้เนื้อที่ในการเก็บข้อมูลมากขึ้น เหมาะแก่การนำไปใช้กับข้อมูลที่ใช้ในการออกรายงานที่ไม่ต้องเสียเวลาประมวลผลหนักๆกับข้อมูลเยอะๆ เช่น ข้อมูลสรุปยอดขายรายปี รายเดือน รายสัปดาห์ใน แยกตามรายการสินค้า ในอดีตที่ผ่านมา หากระบบไม่ได้มีการจัดการโครงสร้างแบบ Datawarehouse การเก็บข้อมูลที่เน้นอ่านอย่างเดียวไม่มีการเปลี่ยนแปลงแล้ว การทำ Denormalize เหมาะมากในการเพิ่มประสิทธิภาพความเร็วของระบบ

Relational Denormalise

Order-1-1

OrderID	OrderDate	CustomerID	CustomerName	No	ProductName	Qty	TotalPrice
201608-001	2016-08-01	1	Jones	1	TV	1	9000
201608-001	2016-08-01	1	Jones	2	Mobile	1	15000
201608-001	2016-08-01	1	Jones	3	Tablet	1	20000

NOSQL embledded

Order-1

OrderID	OrderDate	Customer	OrderItems
201608-001	2016-08-01	{id:1,name:”Jones”}	[{no:1,productName:”TV”,qty:1,totalPrice: 9000}, {no:2,productName:”Mobile”,qty:1,totalPrice: 15000}, {no:3,productName:”Tablet”,qty:1,totalPrice: 20000}]

Denormalise ของ nosql หากเปรียบเทียบกับ relational database แล้วใช้พื้นที่น้อยกว่าเพราะไม่จำเป็นต้องเก็บข้อมูลซ้ำๆกัน

เราควรออกแบบฐานข้อมูลแบบไหนให้เหมาะกับสถานการณ์

Relation -> Normalize

#	ข้อดี	ข้อเสีย
1	ใช้เนื้อที่น้อยในการเก็บข้อมูลที่มีความสัมพันธ์กันเพราะเก็บแค่คีย์หลัก	ต้องเข้าใจความสัมพันธ์ของฐานข้อมูลที่ออกแบบไว้ เพื่อให้สามารถดูแลและแก้ไขได้ถูกต้อง หากมีการเปลี่ยนแปลงใดๆ อาจส่งผลต่อ application ที่นำไปใช้งาน
2	ไม่ต้องคอยตามเปลี่ยนข้อมูลต่างๆของคีย์ที่อ้างถึง เช่น เมื่อตาราง ProductAttribute เปลี่ยนชื่อ หรือค่าฟิลด์อื่นก็ไม่จำเป็นต้องเปลี่ยนที่ตารางทุกค่า Product เว้นแต่ว่าต้องการเปลี่ยนคีย์ใหม่ก็เปลี่ยน คีย์ที่อ้างถึงกันเท่านั้น	

การออกแบบฐานข้อมูลแบบ Relational เหมาะกับข้อมูลที่มีความสัมพันธ์กันโดยที่ข้อมูลในอีกตารางมีรายละเอียดเยอะหลายฟิลด์ หรือหลายรายการความสัมพันธ์แบบ 1-m ทำให้ช่วยลดการเก็บข้อมูลเกินความจำเป็นและการดูแลรักษาได้เป็นระเบียบ

Embedded -> Denormalise

#	ข้อดี	ข้อเสีย
1	ดึงข้อมูลได้รวดเร็วและง่ายขึ้น เพราะไม่ต้องสนใจความสัมพันธ์จากตารางอื่น	เก็บข้อมูลเยอะ ซ้ำซ้อนกับตารางอื่นๆ

การออกแบบฐานข้อมูลแบบ Embleded นั้นมีข้อดีคือสามารถดึงข้อมูลออกมาใช้ได้ง่ายและเร็ว เพราะไม่ต้องคำนึงถึงความสัมพันธ์ข้อมูล แต่ต้องใช้ให้ถูกประเภท การออกแบบข้อมูลแบบ Embleded ควรเป็นข้อมูลประเภท snapshot หรือ immutable จะเหมาะมาก คือเป็นข้อมูลที่สร้างขึ้นมาแล้วจะไม่มีการเปลี่ยนแปลงอีก เช่น ข้อมูลสั่งซื้อสินค้า ณ วันหนึ่งเมื่อสั่งซื้อสินค้าแล้วก็ควรมีการเก็บชื่อสินค้าไว้ ชื่อลูกค้าและคีย์ไว้ เมื่อเวลาผ่านไปลูกค้าอาจจะเปลี่ยนชื่อ ถ้าเราเก็บแค่คีย์และทำเป็นแบบ Relation ข้อมูลในอดีตเราก็จะผิดทันที การเก็บข้อมูลแบบ embleded ไม่เหมาะกับข้อมูลที่มีความสัมพันธ์ที่เปลี่ยนที่ตารางหลักแล้วต้องเปลี่ยนกับตารางที่สัมพันธ์กันด้วย เพราะจะดูแลยากคอยอัพเดตตารางอื่นตลอด ไม่เหมาะกับความสัมพันธ์กับข้อมูลที่มีความสัมพันธ์หลายรายการเช่นกันเพราะจะทำ index ได้ยาก