

Chapter 6

Repository Pattern for Microservice

บรรยายโดย ผศ.ดร.ธราวดิษฐ์ ริติจรูณโรจน์ และอาจารย์สัญชัย น้อยจันทร์

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง



Outline

- แนวทางการออกแบบระบบด้วย Repository Pattern
- แนวคิดของฐานข้อมูลแบบ NoSQL และการใช้งานฐานข้อมูล MongoDB
- การโปรแกรม Spring Boot สำหรับเชื่อมต่อฐานข้อมูล MongoDB แบบ Repository Pattern

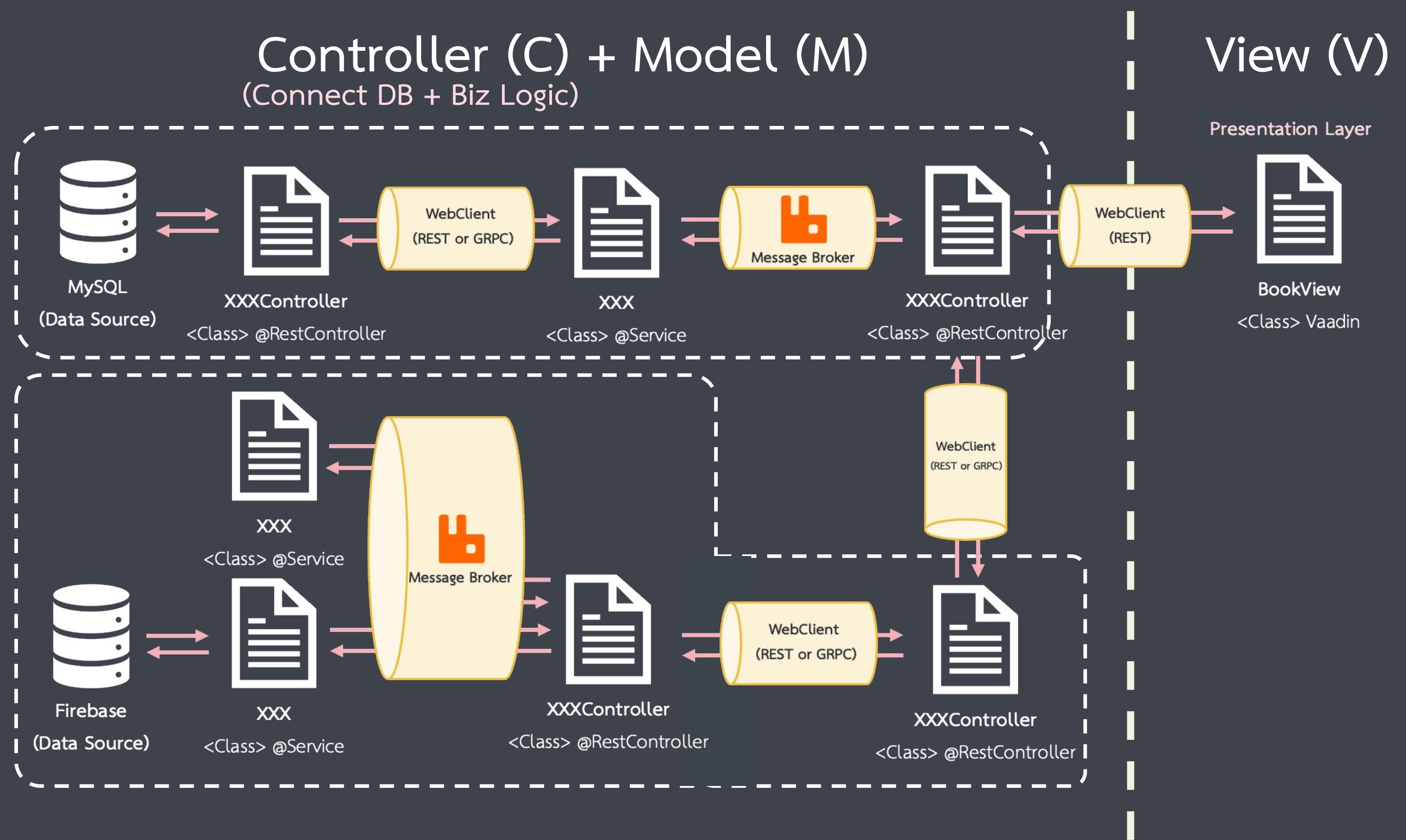


Outline

- แนวทางการออกแบบระบบด้วย Repository Pattern
- แนวคิดของฐานข้อมูลแบบ NoSQL และการใช้งานฐานข้อมูล MongoDB
- การโปรแกรม Spring Boot สำหรับเชื่อมต่อฐานข้อมูล MongoDB แบบ Repository Pattern



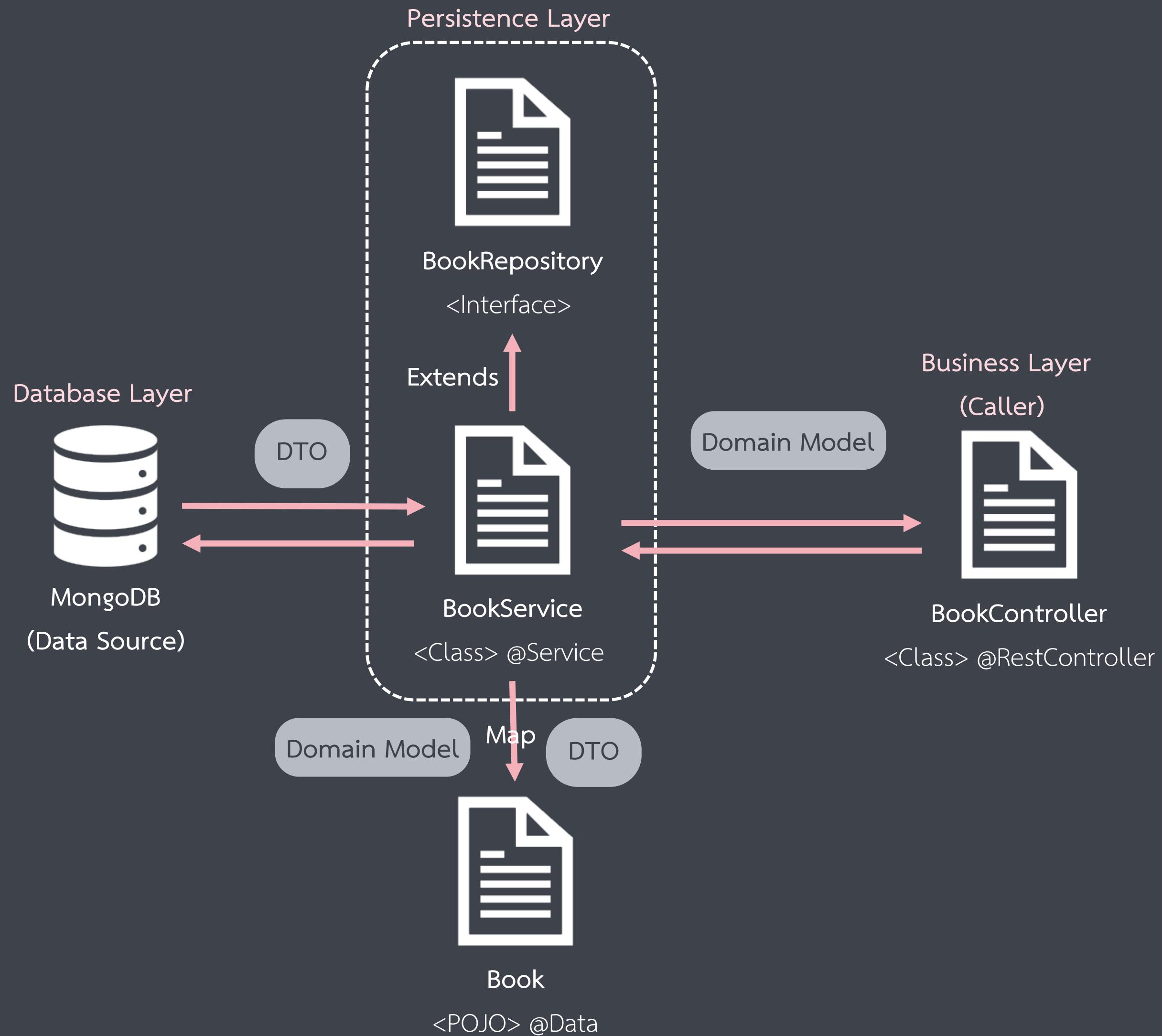
ปัญหาและข้อสังเกต



จากแผนภาพนักศึกษาจะพบว่าในส่วนของ REST API SERVICE ที่นักศึกษาเคยสร้างไว้อาจจะมีความคลุมเครือระหว่างงาน (1) การจัดการข้อมูล (2) การเชื่อมต่อ Data Source และ (3) Business Logic หรือถ้ากล่าวในมุมมองของ MVC Design Pattern จะพบว่า REST API SERVICE ของนักศึกษาอาจจะรวมงานของส่วน Model และ Controller เข้าไว้ด้วยกัน สิ่งนี้ส่งผลให้แต่ละส่วนเกิด Tightly Coupled ภายใน และถ้ามีการปรับ Data Source หรือปรับ Business Logic หรือรูปแบบการสื่อสารก็จะกระทบกันทั้งหมด ทำให้ยากต่อการพัฒนาและการปรับปรุงในอนาคต

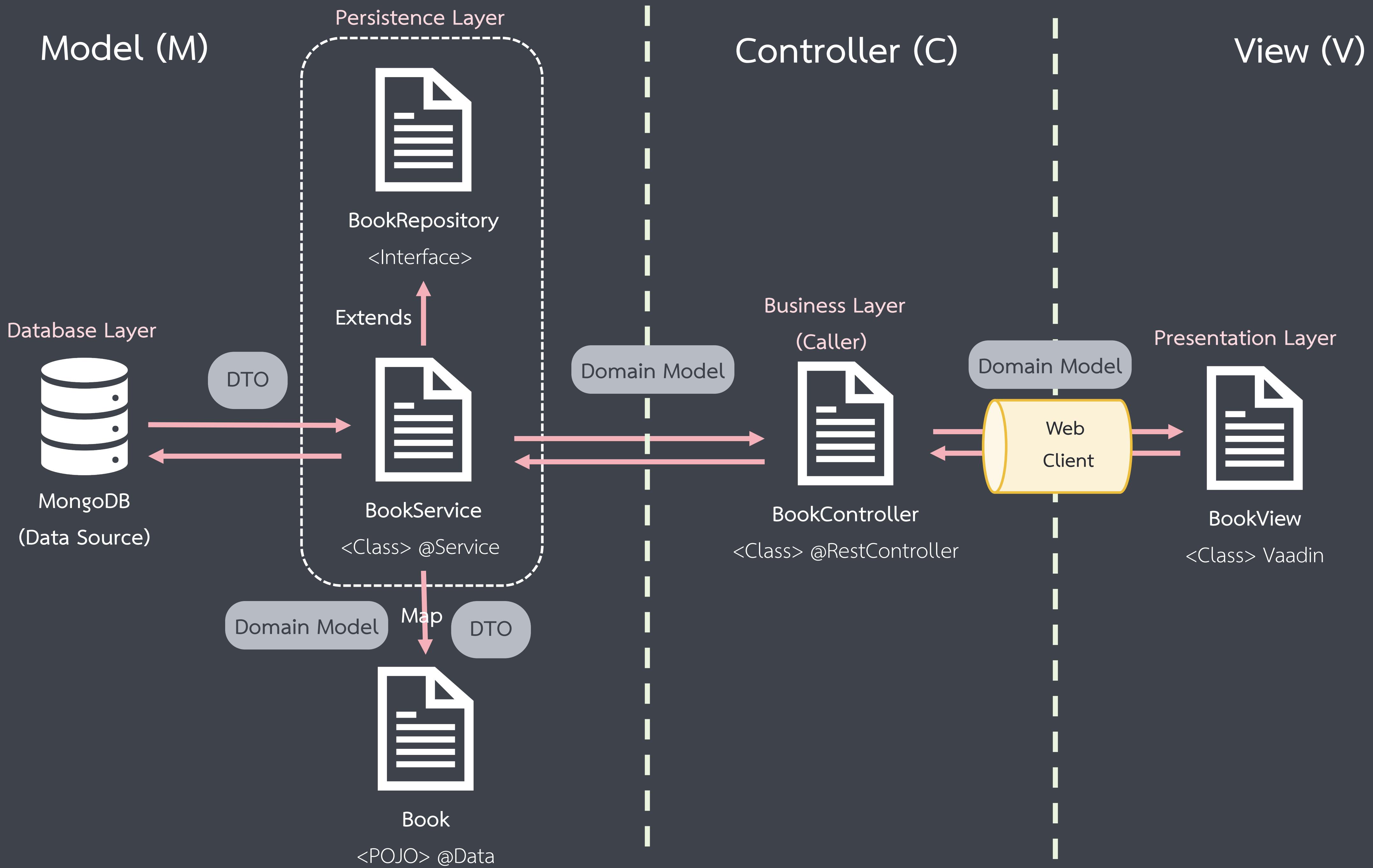


Repository Pattern

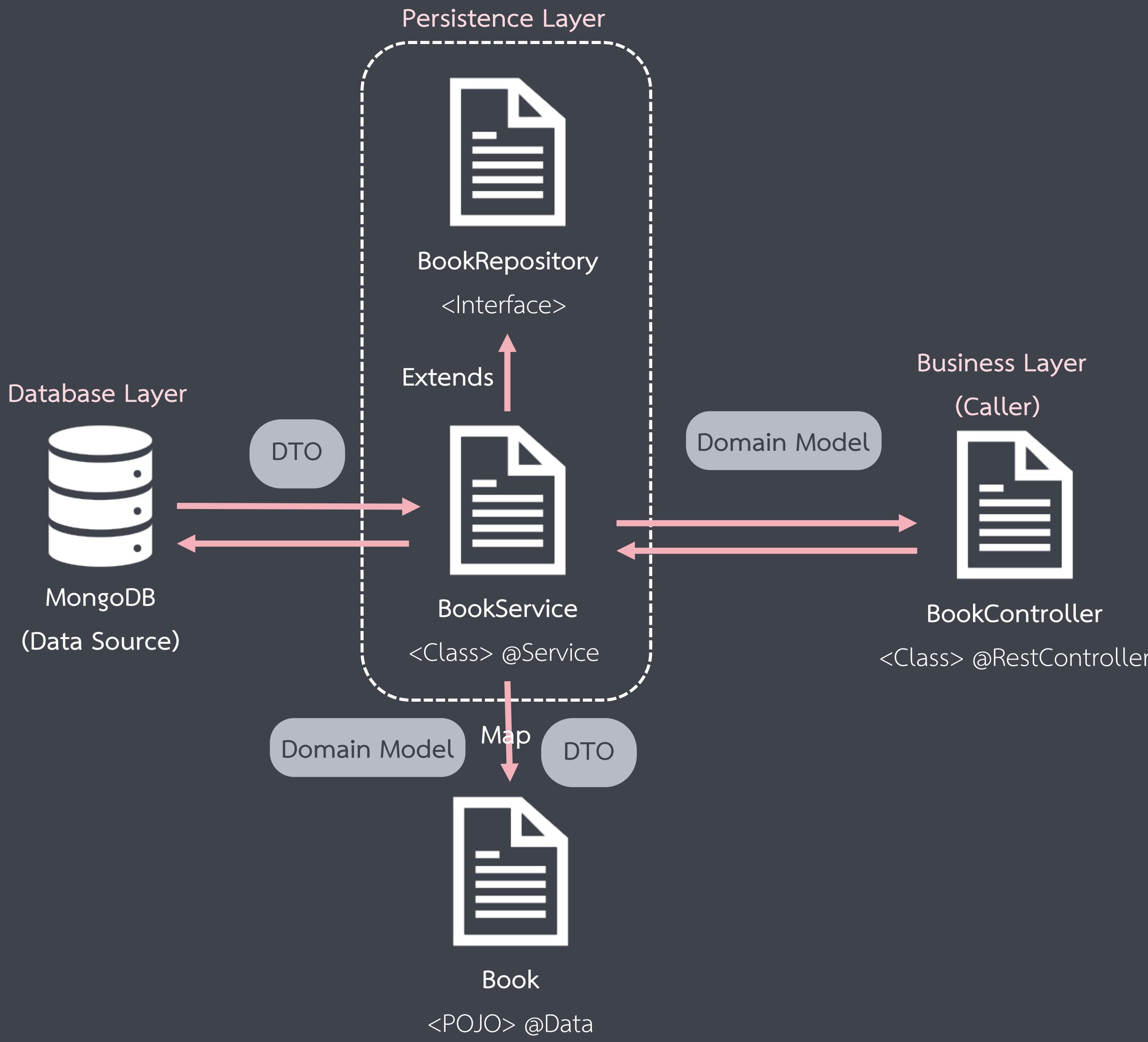


เป็นรูปแบบการออกแบบและพัฒนาโปรแกรมที่มุ่งเน้นในการแยกส่วนระหว่างการเข้าถึงข้อมูล (Data Source) กับ Business Logic ออกจากกัน โดยมี Repository Interface เป็นตัวกลางและตัวช่วยแปลงข้อมูล (Convert) ให้อยู่ในรูปแบบที่พร้อมใช้งาน ซึ่งถือว่าอยู่ใน Persistence Layer ที่อยู่ระหว่าง Business Layer กับ Database Layer นอกจากนี้ Repository Pattern ยังช่วย Decouple ระหว่างส่วนออกจากกันได้ด้วย อาทิเช่น

- “Business layer ไม่ต้องรู้ว่าข้อมูลมาจาก Data Source ได ส่งผลให้เกิดความสะดวกต่อการปรับเปลี่ยน Data Source และยังไม่กระทบกับ Business Layer”
- “แบ่งแยก Logic ที่ซับซ้อนสำหรับการเข้าถึง Data Source ออกจาก Business Layer ทำให้ลด Dependencies ระหว่างโค้ดที่จะเกิดขึ้นลง”



Repository Pattern



Repository Pattern จะประกอบด้วย 5 องค์ประกอบหลัก ได้แก่

- **Data Source**

คือ ที่จัดเก็บข้อมูล, แหล่งที่มาของข้อมูล, ฐานข้อมูล

- **Repository**

คือ อินเตอร์เฟสสำหรับร่างเมธอดที่ใช้ตั้งคำตามว่า (1) Business Layer ต้องการข้อมูลอะไรจาก Data Source และ (2) ให้คืนค่าข้อมูลอะไรและอยู่ในรูปแบบใด

- **Service หรือ RepositoryImpl**

คือ ทำหน้าที่เป็นตัวกลางในการติดต่อและเข้าถึงระหว่าง Business Layer กับ Database Layer ในรูปแบบ Service API นอกจากนี้ นักศึกษาสามารถทำ Caching Data ให้กับระบบได้คลาสนี้

- **Plain Old Java Object (POJO)**

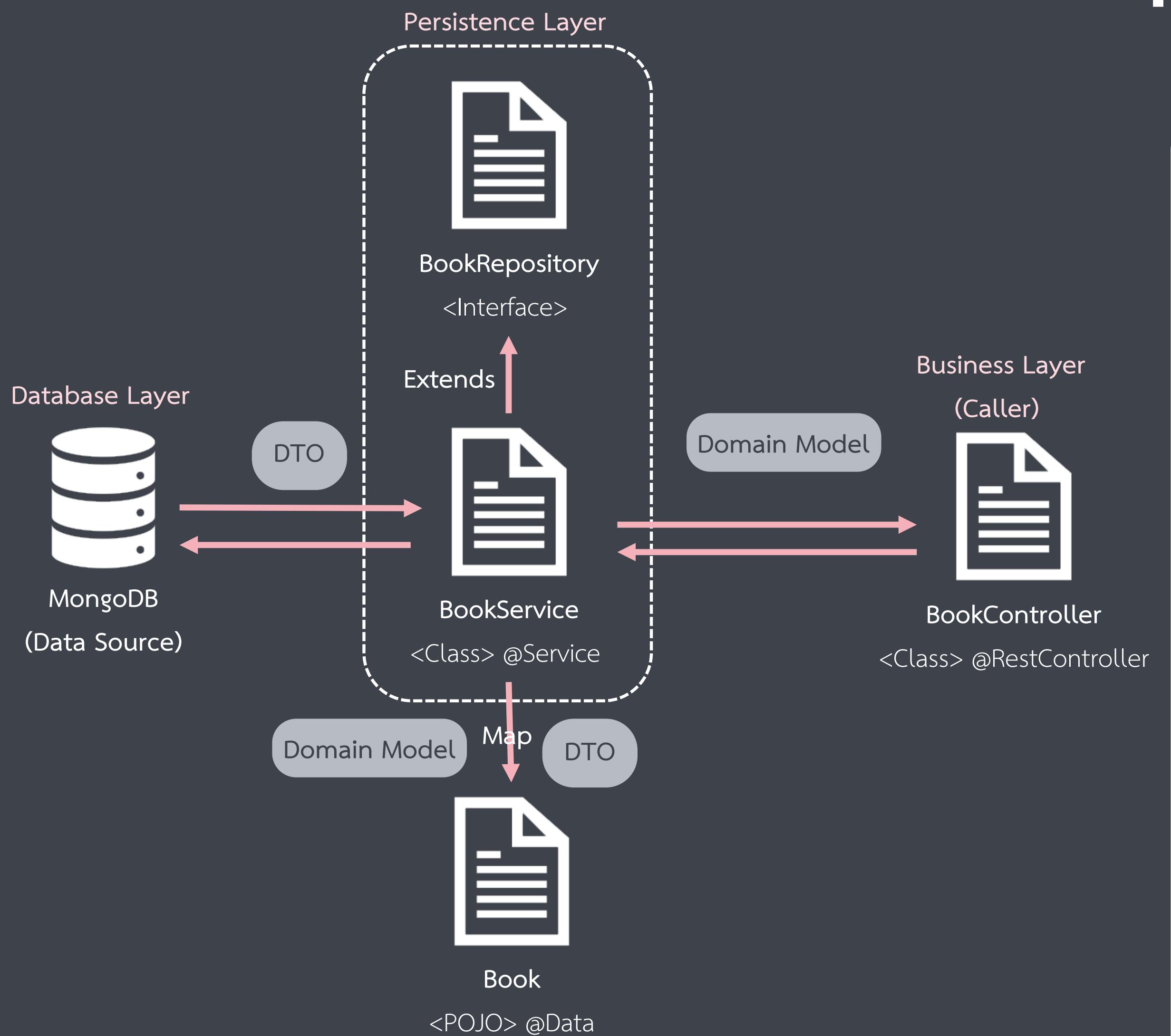
คือ คลาสรูรูมดาที่ไม่มีความซับซ้อน แต่ใน Repository Pattern จะมีหน้าที่แมป Domain Model เข้ากับ Data Transfer Object (DTO) กล่าวคือนำมาร่วมในการแปลงข้อมูล

- **Controller**

คือ คลาสที่เป็นจัดเก็บ Business Logic ต่าง ๆ เอาไว้ และช่วยจัดการต่าง ๆ



Repository Pattern

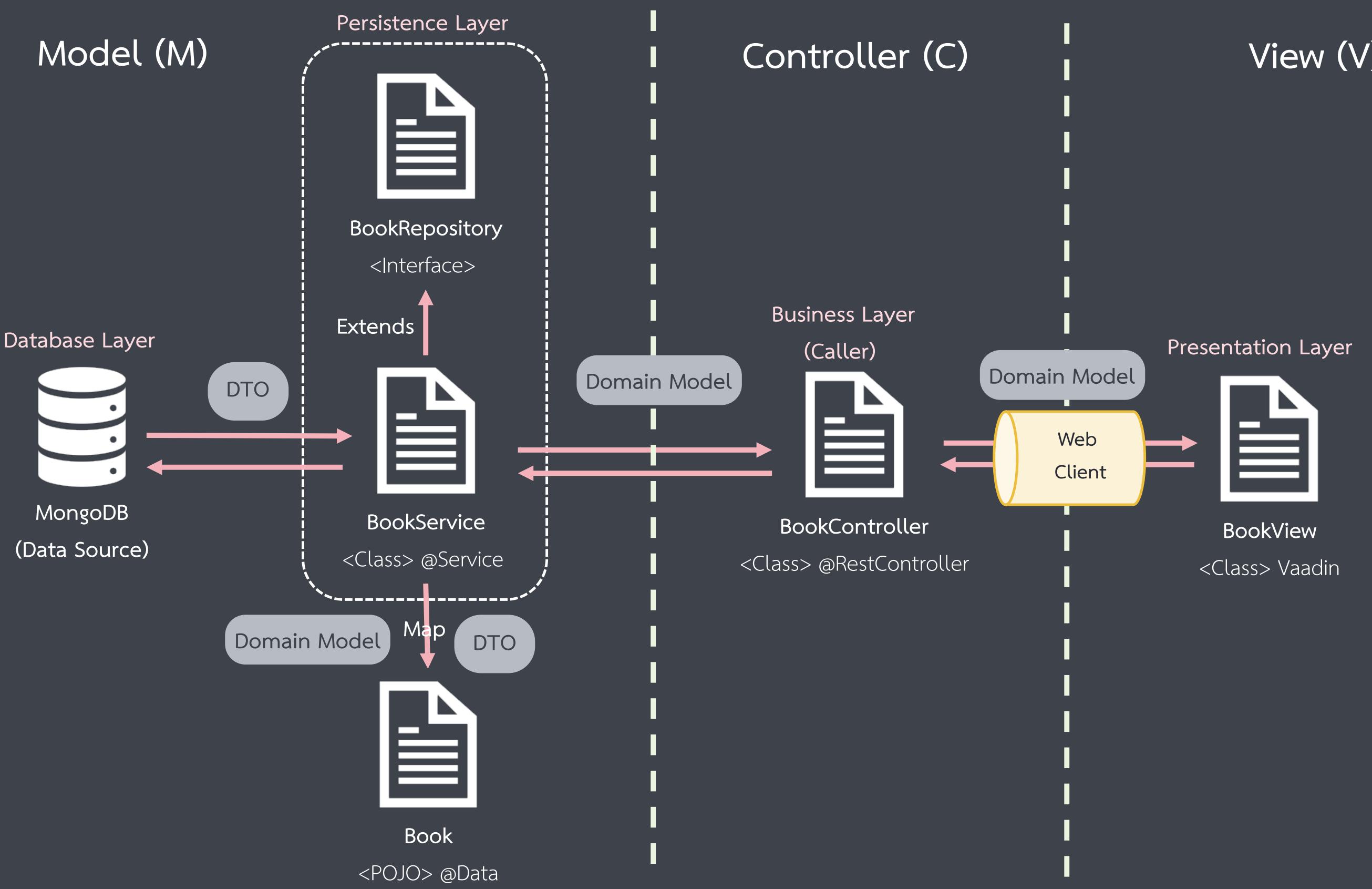


การนำ Repository Pattern ไปใช้งานจะทำให้เกิดความ
สะดวกในประเด็นต่อไปนี้

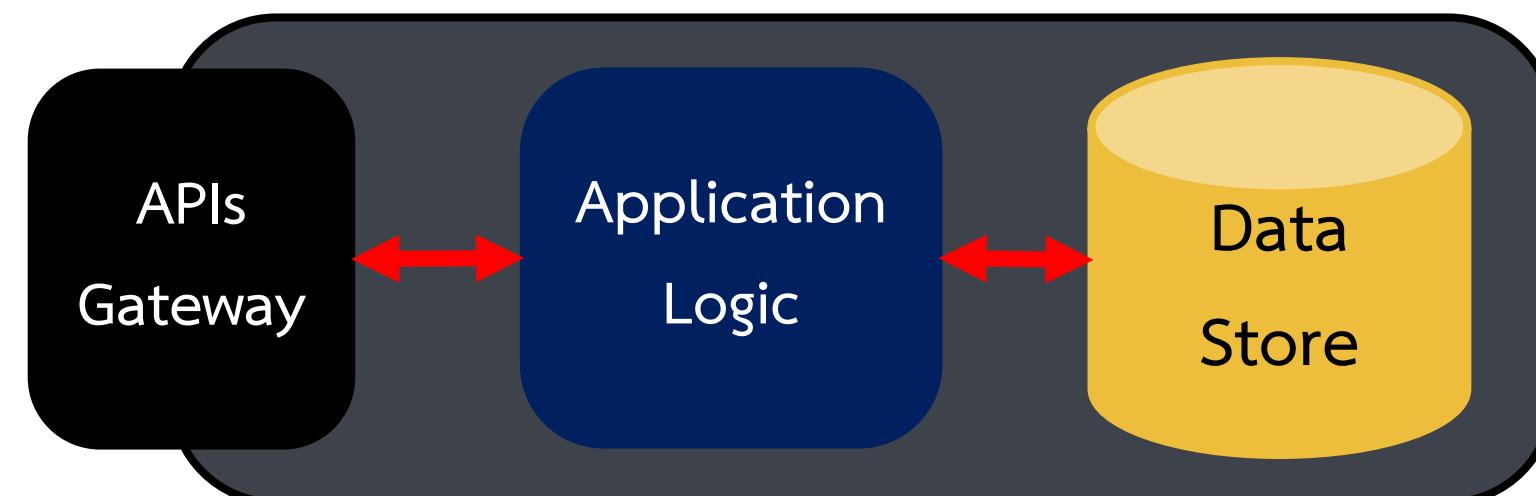
- **Reduce Duplicate Code and Programming Error**
ลดความซ้ำซ้อนของโค้ดในส่วนต่อ Data Source ลง
- **Easy to Maintain and Test**
มี Repository เป็นตัวกลาง ทำให้ Business Logic Layer และ Data Source Layer มีความสัมพันธ์กันแบบหลวມ ๆ
- **Caching Data**
มี Repository เป็นตัวกลาง จึงสะดวกต่อการทำ Caching data ส่งผล
ทำให้ลดจำนวนในการเข้าถึงแหล่งข้อมูล



ข้อควรระวัง



นักศึกษาต้องเข้าใจว่า View ของ Service มีไว้เพื่อรับข้อมูลเข้า Service จาก Service อื่น หรือ ก็คือ API หรือเป็นไฟล์ @RestController นั่นเอง ขณะที่ View ของ GUI เป็นส่วนประสานงานระหว่างตัวระบบ (หรือ Service) กับผู้ใช้งาน ดังนั้น “View ของ Service” ไม่ใช View ของ GUI” แต่ทั้งสอง View เป็นช่องทางสื่อสารระหว่าง Service กับ



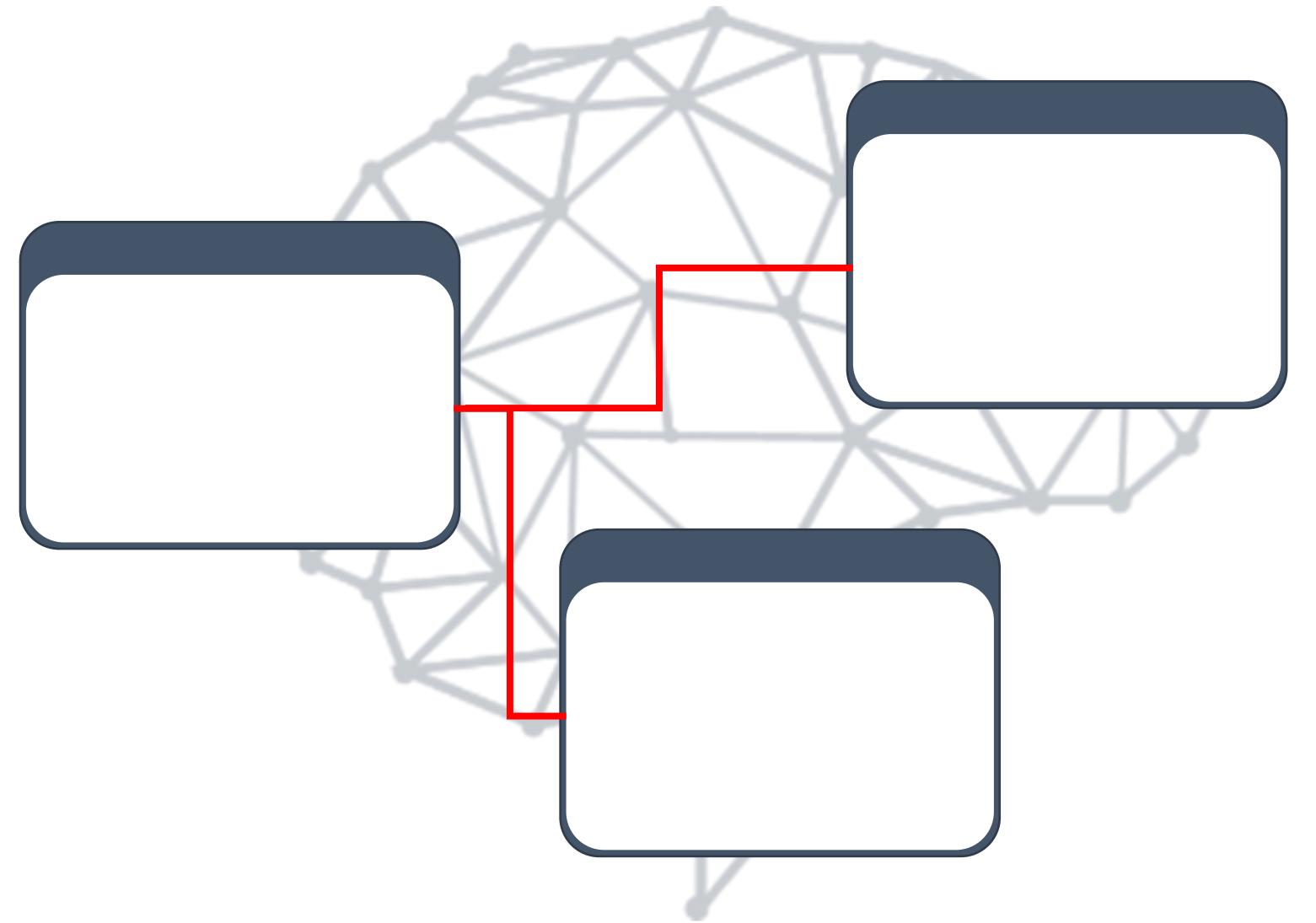


Outline

- แนวทางการออกแบบระบบด้วย Repository Pattern
- แนวคิดของฐานข้อมูลแบบ NoSQL และการใช้งานฐานข้อมูล MongoDB
- การโปรแกรม Spring Boot สำหรับเชื่อมต่อฐานข้อมูล MongoDB แบบ Repository Pattern



แนวคิดการออกแบบฐานข้อมูล RDMS



Relational Database

สำหรับ RDMS ผู้พัฒนาสามารถออกแบบ **โครงสร้างข้อมูลให้อยู่ในรูปแบบหรือมาตรฐาน** ที่ต้องการได้โดยปราศจากการพิจารณารูปแบบการเรียกใช้งานข้อมูล (Query) อีกทั้งยังสามารถ **ขยายหรือเพิ่มเติมรูปแบบการเรียกใช้งานข้อมูลได้ในภายหลัง**



การเรียกใช้งานข้อมูล (Query)





แนวคิดการออกแบบฐานข้อมูล NoSQL



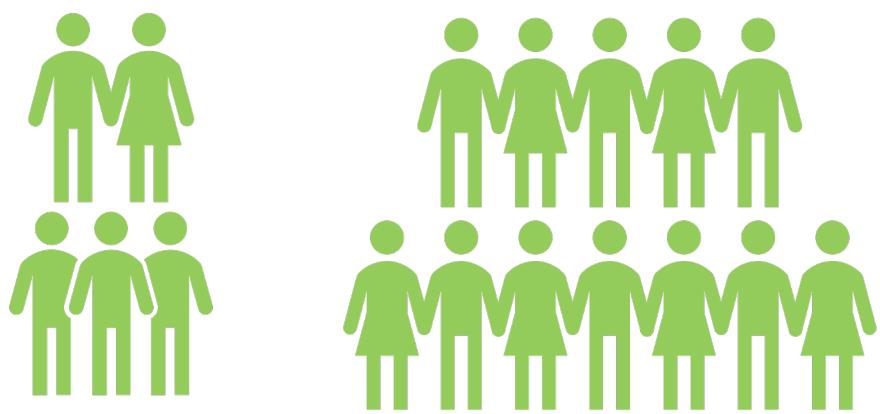
NoSQL Database

สำหรับ NoSQL ผู้พัฒนา **ไม่ควรเริ่มออกแบบจากโครงสร้างข้อมูลหรือรูปแบบก่อน** แต่ควรพิจารณาคำถูกและคำตอบที่ต้องการก่อน (Query Question) นอกจากนี้ ควรทำความเข้าใจปัญหาทางธุรกิจและการทำงานส่วนที่สำคัญของระบบของเราว่ามีการเรียกใช้ข้อมูลแบบใด “**Most well designed applications require only one table**”



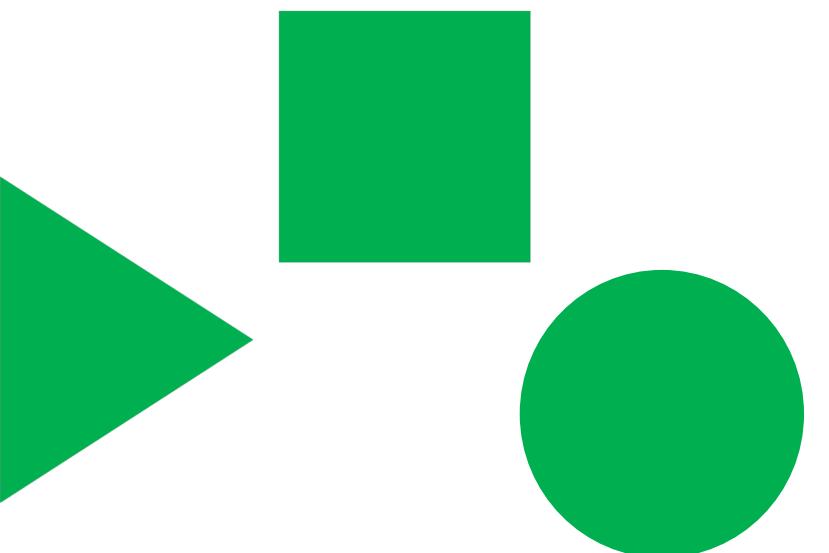
แนวคิดการออกแบบฐานข้อมูล NoSQL

สำหรับ NoSQL ผู้พัฒนาควรเริ่มจาก **พิจารณาคำถามและคำตอบที่ต้องการก่อน (Query Question)** เพื่อใช้ในการออกแบบฐานข้อมูลให้เหมาะสม ซึ่งในทางปฏิบัติมี 3 ปัจจัยหลักที่ควรพิจารณา ก่อน ได้แก่



Data Size

ต้องทราบรู้จำนวนของข้อมูลที่จะจัดเก็บต่อหนึ่งครั้ง สิ่งนี้สามารถช่วยให้ออกแบบการแบ่งข้อมูลได้มีประสิทธิภาพสูงขึ้น



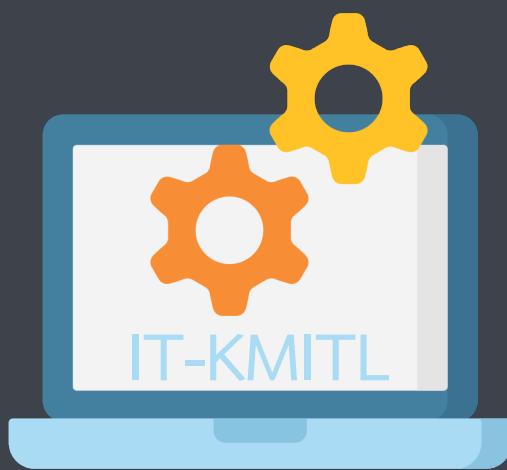
Data Shape

ในฐานข้อมูลประเภท RDMS บางคำสั่ง Query จะมีการเปลี่ยนรูปร่างหรือสร้างตารางช่วงรวมมาเพื่อประมวลผลบาง Query สิ่งนี้ส่งผลทำให้ใช้เวลาในการประมวลผลที่สูงขึ้นทำให้ประสิทธิภาพด้านเวลาและการเปลี่ยนแปลงขนาดไม่ดี



Data Velocity

เนื่องจากฐานข้อมูล NoSQL รองรับการประมวลแบบกระจาย (Distribution) ซึ่งหมายความว่ามีหลายเครื่องผู้ให้บริการสำหรับการจัดเก็บข้อมูล ซึ่งเมื่อผู้พัฒนาทราบเรื่อง Data Size และ สามารถเลือก I/O หรือ อุปกรณ์ Hardware ที่เหมาะสมได้



แนวคิดการออกแบบฐานข้อมูล NoSQL

CountryID	CName
1	Thailand
2	Japan

yearID	Year
1	1990
2	2000

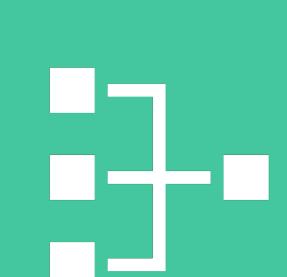
CustID	CustName	CustLastName
1	Sara	Smith
2	Alex	Potter

ID	CountryID	YearID	CustomerID	Amount
1	1	1	1	350
2	2	1	2	450

ความสัมพันธ์ถูกสะท้อนด้วย FK

Process

ID	CountryID	YearID	CustName	CustLastName	Amount
1	Thailand	1990	Sara	Smith	350
2	Japan	2000	Alex	Potter	450



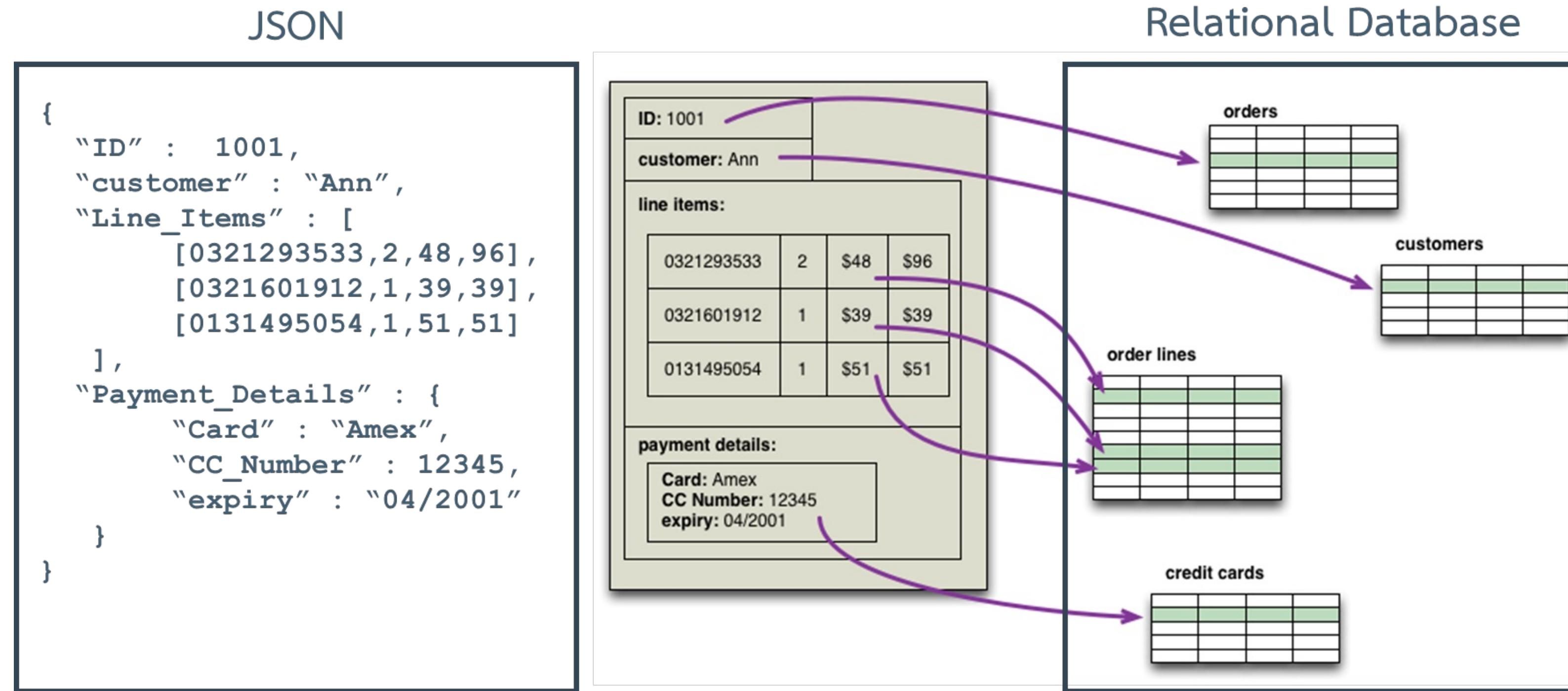
Keep related data together

การจัดเก็บความสัมพันธ์ร่วมกับข้อมูล

ตลอด 20 ปีที่ผ่านมาการจัดการและเชื่อมต่อข้อมูล (เรียกว่า ความสัมพันธ์ภายใน locality of reference) เป็นปัจจัยที่สำคัญมากที่ส่งผลกระทบต่อเวลาในการตอบสนองของฐานข้อมูล ดังนั้น จึงมีแนวคิดที่ว่า **ถ้าสามารถจัดเก็บความสัมพันธ์ภายในและข้อมูลไว้ที่เดียวกันได้จะสามารถลดเวลาในการประมวลผลลงได้**



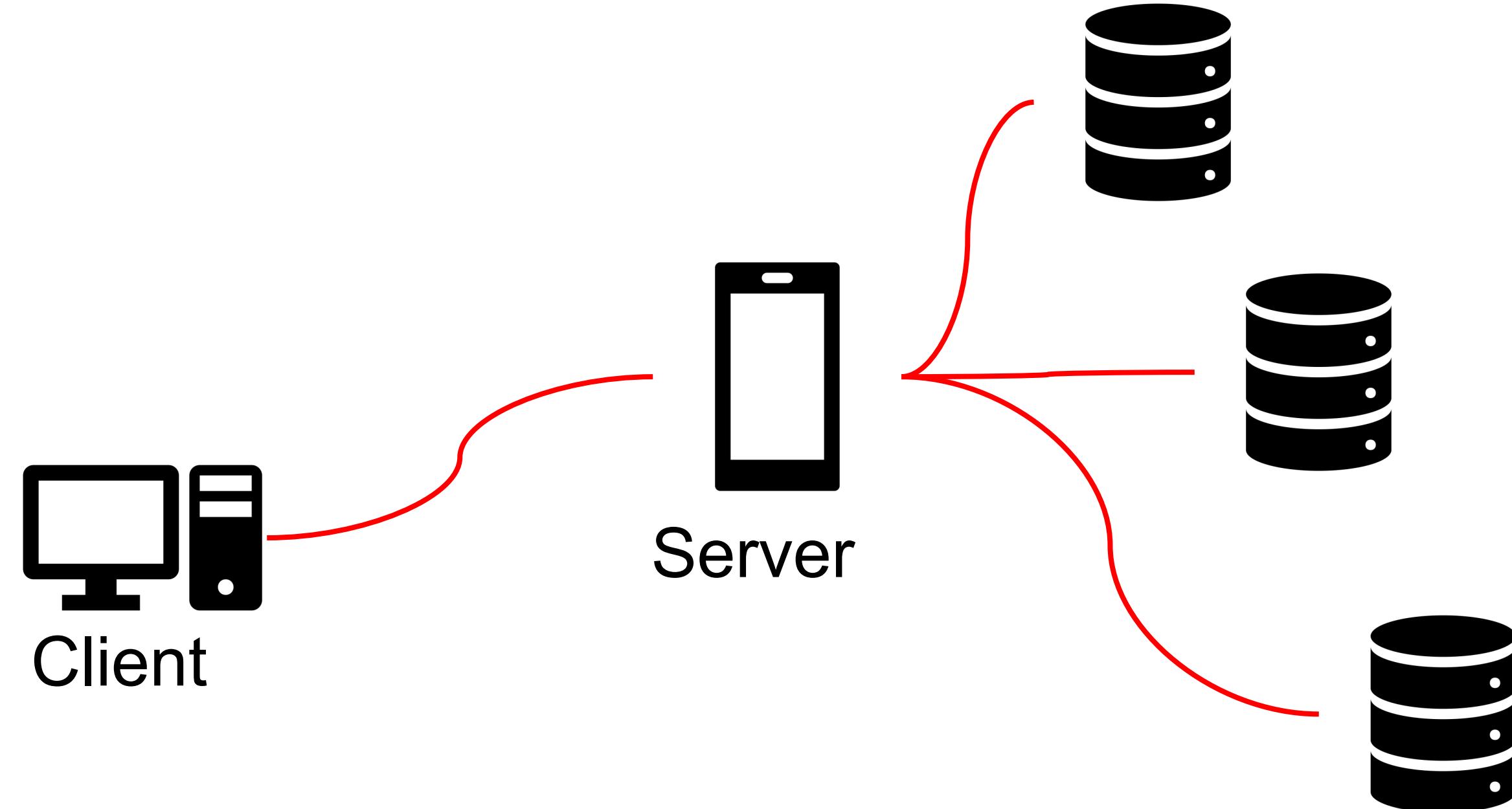
แนวคิดการออกแบบฐานข้อมูล NoSQL



ซึ่งสอดคล้องกับลักษณะของฐานข้อมูลชนิด NoSQL ขณะที่ฐานข้อมูลชนิด RDBS ความสัมพันธ์ต่าง ๆ และข้อมูลถูกกระจายเก็บในตารางต่าง ๆ จึงทำให้เสียเวลาในการประมวลผล ถึงแม้ว่าการออกแบบฐานข้อมูลที่ดีควรจะออกแบบให้มีตารางเพียงตารางเดียว แต่อย่างไรก็ตาม ก็อาจจะมีมากกว่าหนึ่งตารางได้ตามความเหมาะสม เช่น มี Query หลากหลายแบบ, ข้อมูลที่ถูกแบ่งเก็บเป็นช่วงเวลาที่มีขนาดใหญ่มาก ๆ เป็นต้น



แนวคิดการออกแบบฐานข้อมูล NoSQL



ปริมาณคำตาม (Query) ที่มากและซับซ้อนเป็นส่วนหนึ่งที่ส่งผลกระทบต่อความสามารถและประสิทธิภาพของฐานข้อมูล เพื่อที่จะหลีกเลี่ยงการเกิด “Hot Spots” เวลาออกแบบฐานข้อมูลควรคำนึงถึงการ **กระจายปริมาณการสือสารและปริมาณข้อมูลให้อยู่ในหลายแหล่งข้อมูลหรือหลายฐานข้อมูล** ที่แตกต่างกัน ดังนั้น การกระจายคำตามเพื่อเข้าถึงข้อมูลจากหลายแหล่งข้อมูล (Data Source) หรือหลายฐานข้อมูล ซึ่งแหล่งข้อมูลเหล่านั้นอาจจะอยู่บนเครื่องผู้ให้บริการเดียวกันหรือคนละเครื่องก็ได้



ฐานข้อมูล NoSQL ชนิดเอกสาร

ฐานข้อมูลที่มีแนวคิดจากการออกแบบเพื่อใช้จัดเก็บเอกสารที่มีโครงสร้างและรายละเอียดแตกต่างกัน ซึ่งอาจจะกล่าวได้ว่าในเอกสารใด ๆ ข้อมูล (Data) หรือสารสนเทศ (Information) จะถูกเข้ารหัส (Encapsulation) เอาไว้ในรูปแบบใดรูปแบบหนึ่ง เช่น XML, JSON, BSON, PHP Array, Python Dictionary, Ruby Hash และ อื่น ๆ



Relational Database	MongoDB
Table หรือ View	Collection
Row	Document
Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard



MongoDB

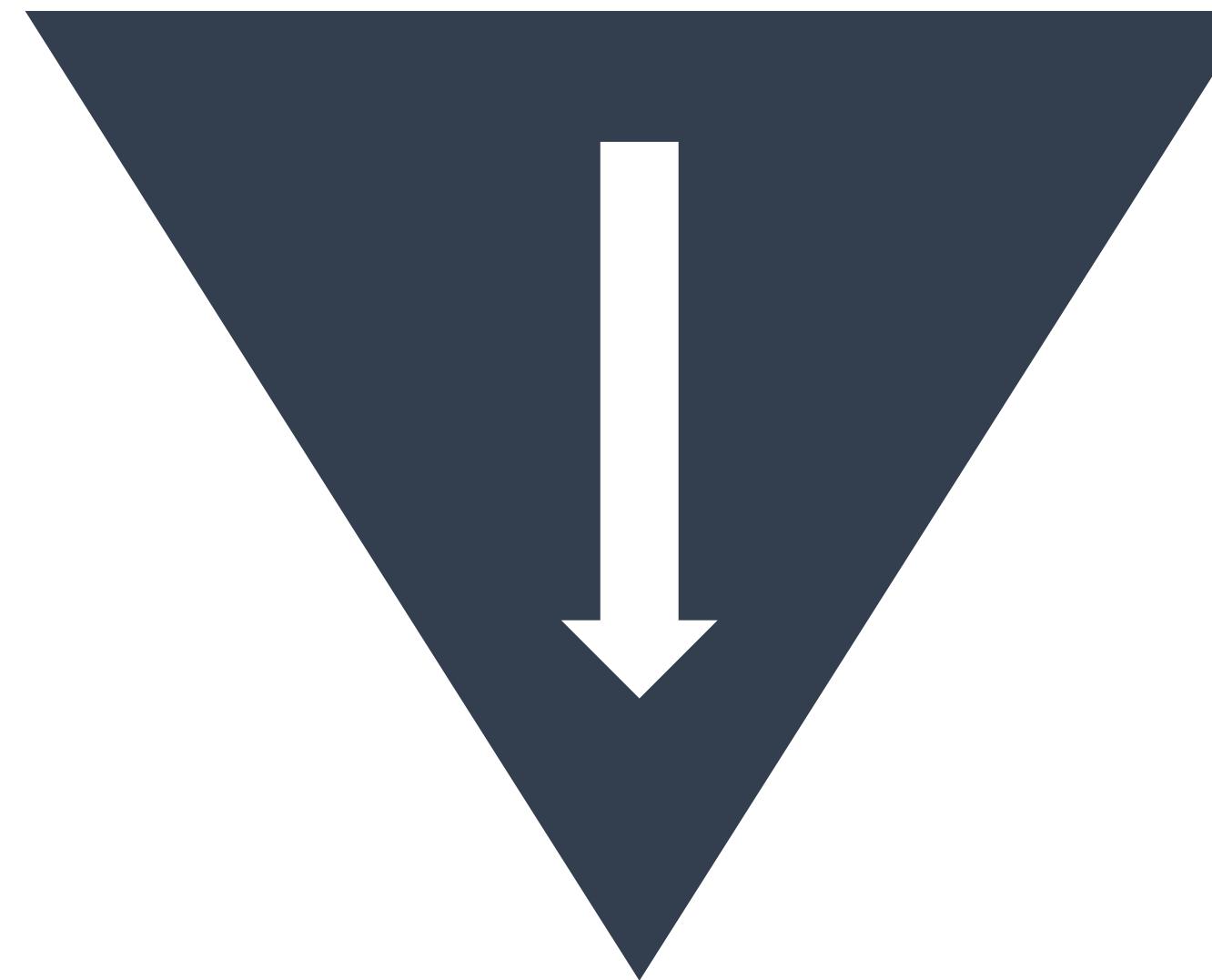
เป็นฐานข้อมูลประเภท NoSQL ชนิดเอกสาร (Document) ที่มีโครงสร้างยืดหยุ่น (Flexible Schema) นั่นคือ **Collection** ของ MongoDB **ไม่จำเป็นที่โครงสร้างของแต่ละ Document ต้องเหมือนกัน** ขณะที่ฐานข้อมูล RDMS ต้องกำหนดให้แต่ละ แถว (row) มีโครงสร้างสอดคล้องกับที่กำหนดไว้ข้างต้น

ซึ่งในแต่ละ collection ใน Document ได ๆ ไม่จำเป็นที่ (1) คุณลักษณะ (Field), (2) ชนิดข้อมูล และ (3) จำนวนคุณลักษณะ จะต้องเหมือนหรือเท่ากัน นอกจากนี้ การปรับเปลี่ยนโครงสร้างในแต่ละ Document อาทิเช่น การเพิ่ม หรือลบคุณลักษณะใหม่ หรือเปลี่ยนชนิดข้อมูลสามารถทำได้

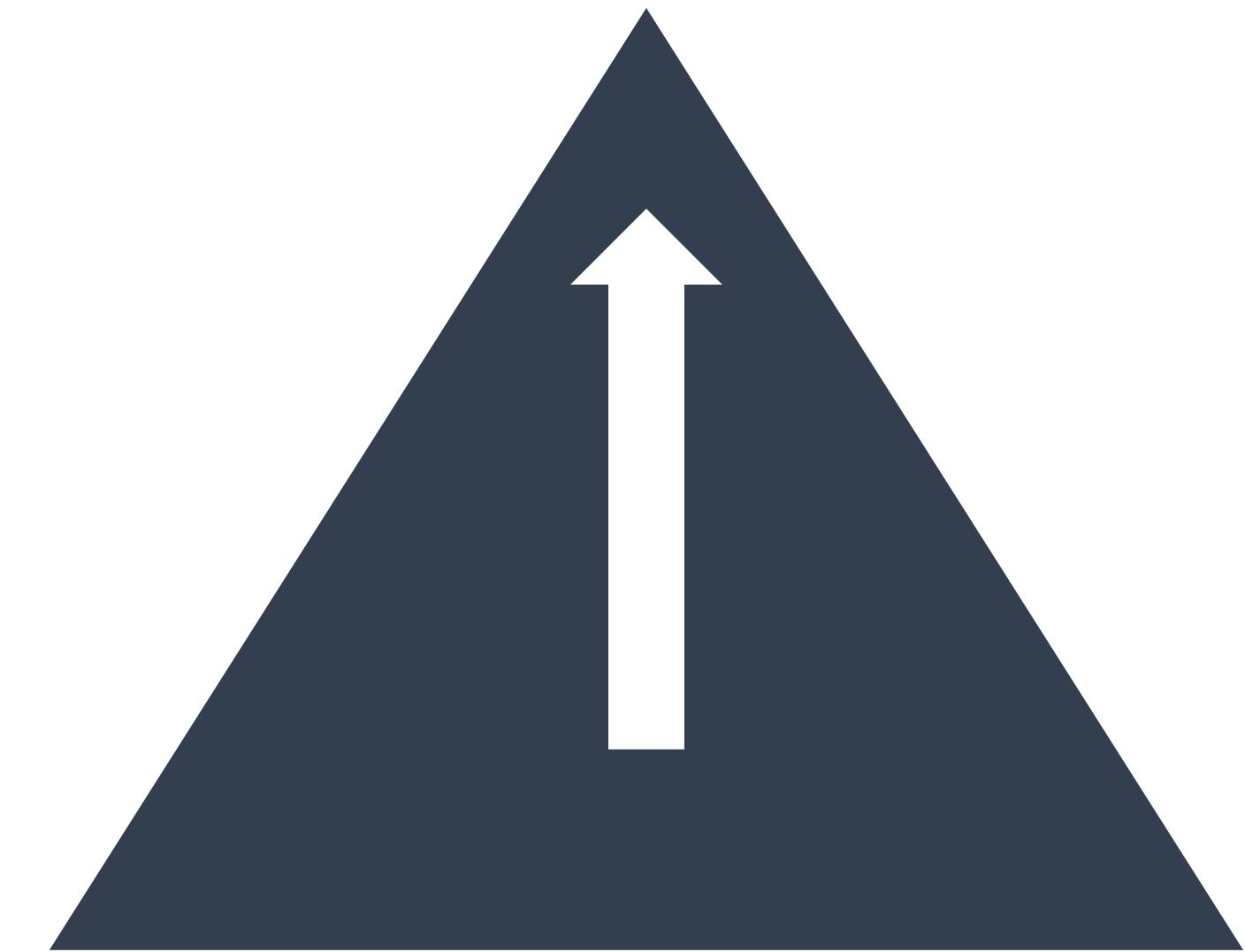
```
“Car” : [  
    {  
        “_id” : 1,  
        “owner” : 3,  
        “color” : “Blue”,  
        “brand” : “Toyota”  
    }, {  
        “_id” : 5,  
        “owner” : 3,  
        “color” : 234,  
        “brand” : “Honda”,  
        “MP3” : False  
    }]  
]
```



แนวทางการออกแบบฐานข้อมูล MongoDB



การ Normalization โดยอาศัยหลักการ
Document Reference



การ Denormalization โดยอาศัยหลักการ
Embedded Document



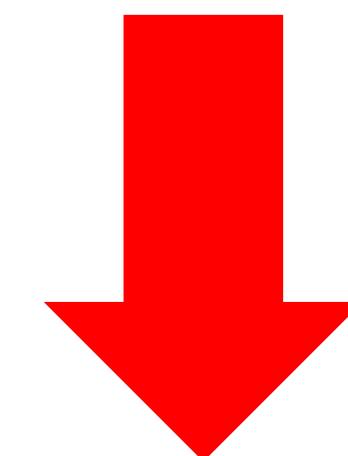
แนวทางการออกแบบฐานข้อมูล MongoDB

- หลักเลี้ยงแนวคิดหรือวิธีการออกแบบที่สำคัญความสัมพันธ์
- คาดการณ์เหตุการณ์ที่จะเกิดขึ้นเมื่อโครงสร้างของฐานข้อมูลถูกปรับเปลี่ยนในภายหลัง
- พิจารณาขนาดของข้อมูล คีย์ และไฟล์เอกสาร
- รูปแบบการสอบถาม (Query)



แนวทางการออกแบบฐานข้อมูล MongoDB

ID	CountryID	YearID	CustName	CustLastName	Amount
1	Thailand	1990	Sara	Smith	350
2	Japan	2	Alex	Potter	450



Normalization

CountryID	CName	yearID	Year	CustID	CustName	CustLastName
1	Thailand	1	1990	1	Sara	Smith
2	Japan	2	2000	2	Alex	Potter

ID	CountryID	YearID	CustomerID	Amount
1	1	1	1	350
2	2	1	2	450

CountryID	CName	yearID	Year	CustID	CustName	CustLastName
1	Thailand	1	1990	1	Sara	Smith
2	Japan	2	2000	2	Alex	Potter

ID	CountryID	YearID	CustomerID	Amount
1	1	1	1	350
2	2	1	2	450



Denormalization

ID	CountryID	YearID	CustName	CustLastName	Amount
1	Thailand	1990	Sara	Smith	350
2	Japan	2	Alex	Potter	450



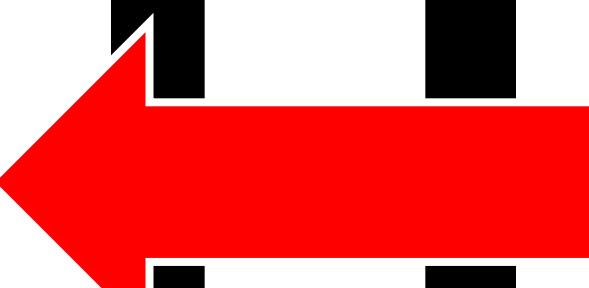
แนวทางการออกแบบฐานข้อมูล MongoDB

```
Customer :{  
    "_id" : 3,  
    "name" : "Peter",  
    "address" : "Bangkok, Thailand"  
}  
  
Car : [  
    {"_id" : 1,  
    "owner" : 3,  
    "color" : "Blue",  
    "brand" : "Toyota"},  
    {"_id" : 5,  
    "owner" : 3,  
    "color" : "Black",  
    "brand" : "Honda"}  
]
```

Denormalization



Normalization



```
Customer :{  
    "_id" : 3,  
    "name" : "Peter",  
    "address" : "Bangkok, Thailand"  
    "Car" : [ {  
        "_id" : 1,  
        "owner" : 3,  
        "color" : "Blue",  
        "brand" : "Toyota"  
    } , {  
        "_id" : 5,  
        "owner" : 3,  
        "color" : "Black",  
        "brand" : "Honda"  
    } ]  
}
```



การออกแบบ Collection



Embedded Document



Document Reference



การอوكແບບ Collection



Embedded Document

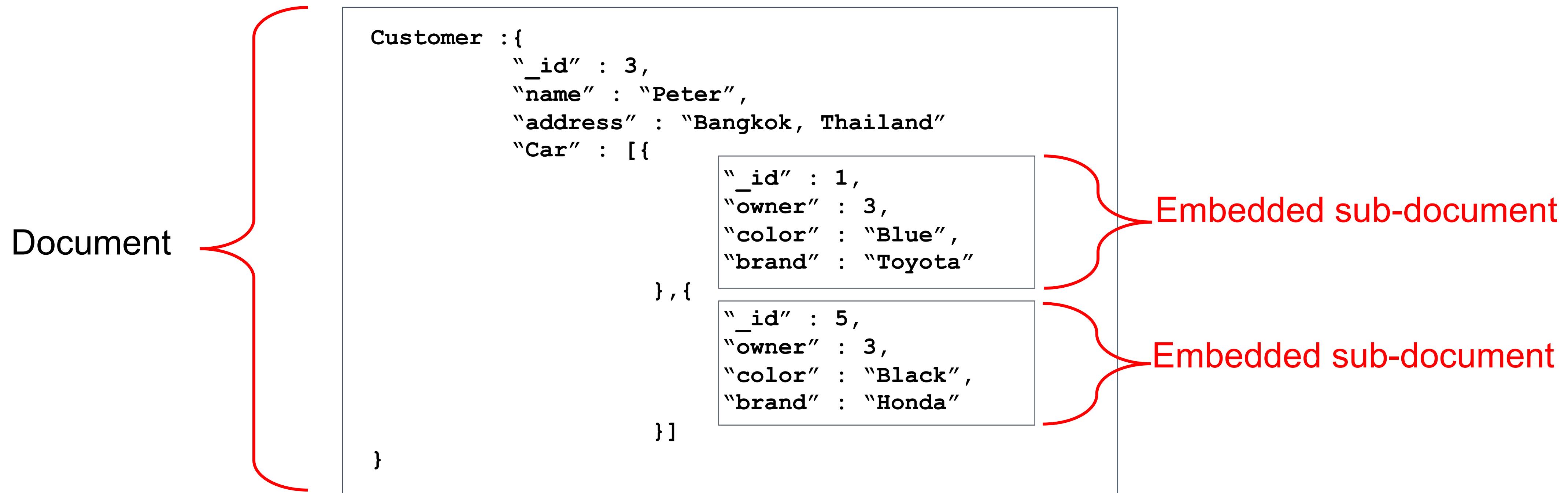


Document Reference



Embedded Document

Embedded / Nested Document เป็นหนึ่งในวิธีสำหรับการออกแบบ ๆ จำลองข้อมูล (Data Model Design) ที่อยู่บนพื้นฐานของ Denormalization ซึ่งอาศัยการแทรก / ฝังตัว object หรือ document ลงใน document อีกอัน



ซึ่งนิยมใช้สำหรับการออกแบบ Collection จากความสัมพันธ์แบบ one-to-one หรือ one-to-many



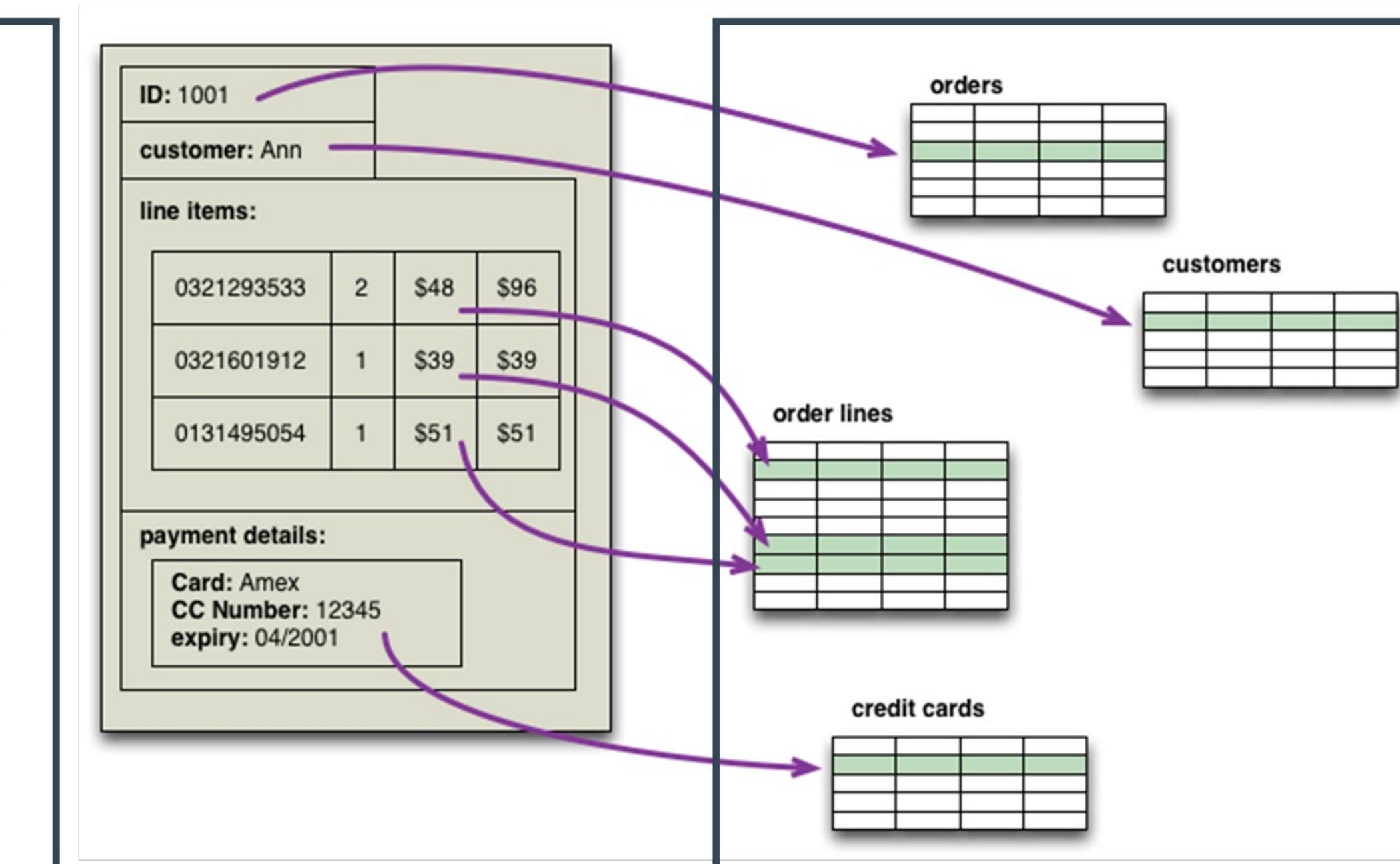
Embedded Document

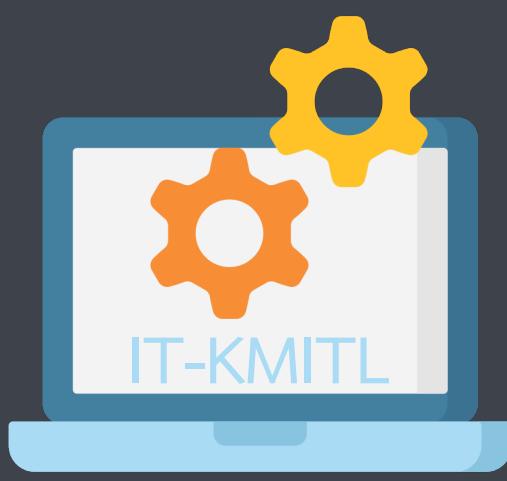
การ Embedded Document เป็นวิธีการที่ทำให้การอ่านข้อมูลมีประสิทธิภาพดีขึ้น เนื่องจากข้อมูลที่เกี่ยวข้องกันได้รวมกลุ่มกันเป็นหนึ่งเดียวกัน ส่งผลให้ประสิทธิภาพในการขอรับบริการได้รับการตอบสนองที่รวดเร็ว และข้อมูลที่สัมพันธ์กันสามารถปรับปรุงรูปแบบความสัมพันธ์ได้ โดยไม่ต้องคำนึงถึงโครงสร้าง (Schema)

MongoDB

```
{  
    "ID" : 1001,  
    "customer" : "Ann",  
    "Line_Items" : [  
        [0321293533,2,48,96],  
        [0321601912,1,39,39],  
        [0131495054,1,51,51]  
    ],  
    "Payment_Details" : {  
        "Card" : "Amex",  
        "CC_Number" : 12345,  
        "expiry" : "04/2001"  
    }  
}
```

Relational Database





ออกแบบ Collection ความสัมพันธ์แบบ one-to-one

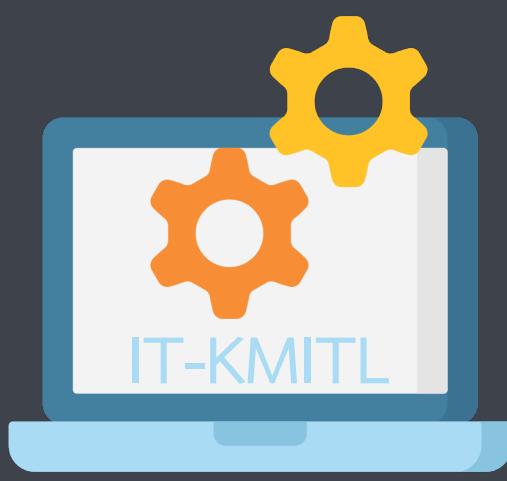
People

_id	name
9	Sara Wille

Address

people_id	street	city	zip
9	Rama III Rd.	Bangkok	10120

```
{  
  _id: 9,  
  name: "Sara Wille"  
}  
  
{  
  people_id: 9,  
  street: "Rama III Rd.",  
  city: "Bangkok",  
  zip: "10120"  
}
```



ออกแบบ Collection ความสัมพันธ์แบบ one-to-one

People

_id	name	street	city	zip
9	Sara Wille	Rama III Rd.	Bangkok	10120

```
{  
  "_id": 9,  
  "name": "Sara Wille" ,  
  "street": "Rama III Rd." ,  
  "city": "Bangkok" ,  
  "zip": "10120"  
}
```



ออกแบบ Collection ความสัมพันธ์แบบ one-to-one

People

_id	name	Address		
		street	city	Zip
9	Sara Wille	Rama III Rd.	Bangkok	10120

เนื่องจากกลุ่มคุณลักษณะดังกล่าวเป็นคุณลักษณะของที่อยู่ (Address) ซึ่งประกอบด้วยคุณลักษณะย่อย ๆ โดย **แต่ละ collection** อาจจะมีไม่เหมือนกัน นอกจากนี้ ยังเป็นหลักการออกแบบโดยอาศัยเทคนิค Embedded Document นั่นคือ การนำห้องโครงสร้างเดิมซ่อนทับหรือใส่เข้าไปร่วมกับอีกโครงสร้างหนึ่ง

```
{  
    _id: 9,  
    name: "Sara Wille" ,  
    Address : {  
        street: "Rama III Rd." ,  
        city: "Bangkok" ,  
        zip: "10120"  
    }  
}
```



ออกแบบ Collection ความสัมพันธ์แบบ one-to-many

People

_id	name
9	Ron Weasley

Address

people_id	street	city	zip
9	Rama III Rd.	Bangkok	10120
9	Rama IX Rd.	Bangkok	10150

```
{
  _id: 9,
  name: "Ron Weasley"
}, {
  people_id: 9,
  street: "Rama III Rd.",
  city: "Bangkok",
  zip: "10120"
}, {
  people_id: 9,
  street: "Rama IX Rd.",
  city: "Bangkok",
  zip: "10150"
}
```



ออกแบบ Collection ความสัมพันธ์แบบ one-to-many

People

_id	name	address		
		street	city	zip
9	Ron Weasley	Rama III Rd.	Bangkok	10120
		Rama IX Rd.	Bangkok	10150

```
People : {  
    _id: 9,  
    name: "Ron Weasley" ,  
    address : [ {  
        street: "Rama III Rd.",  
        city: "Bangkok",  
        zip: "10120"  
    }, {  
        street: "Rama IX Rd.",  
        city: "Bangkok",  
        zip: "10150"  
    } ]  
}
```



การอ่านแบบ Collection



Embedded Document



Document Reference



หลักการ Document Reference

Book

```
{  
    title: "The Basic of SQL",  
    author: {  
        "name" : "Hermione Granger", "address" : "Bangkok, Thailand"  
    }  
    published_date: ISODate("2010-09-24"),  
    pages: 121,  
}, {  
    title: "Java Programming",  
    author: {  
        "name" : "Hermione Granger", "address" : "Bangkok, Thailand"  
    }  
    published_date: ISODate("2015-09-02"),  
    pages: 320,  
}
```

Document Reference เป็นหนึ่งในวิธีสำหรับการออกแบบ ๆ จำลองข้อมูล (Data Model Design) ที่อยู่บนพื้นฐานของ Normalization ซึ่งอาศัยการอ้างอิงจาก object หรือ document อันหนึ่งไปสู่อีกอันหนึ่ง เพื่อ **หลีกเลี่ยงการซ้ำซ้อนของข้อมูลที่มากเกินไป**

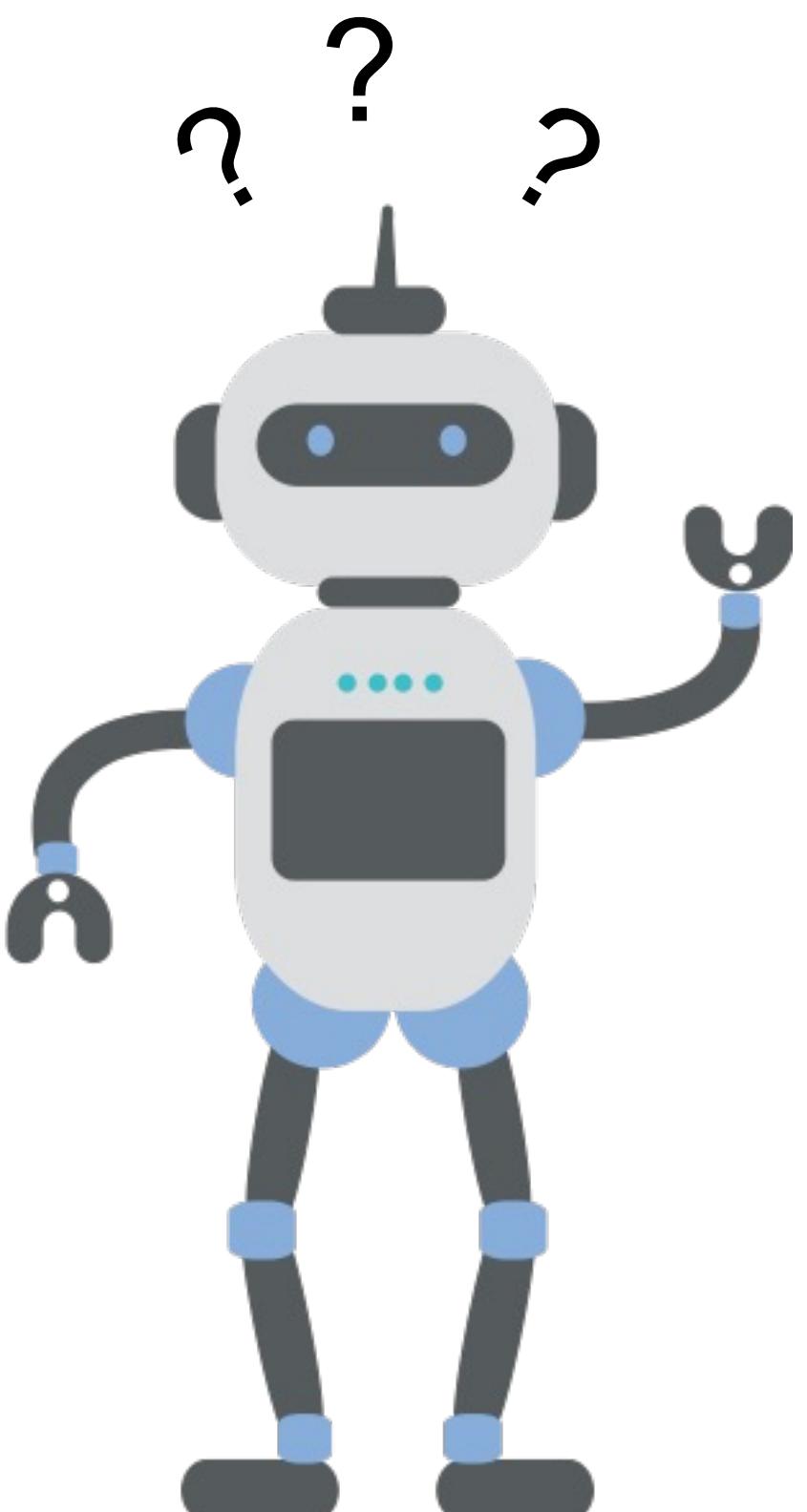
อย่างไรก็ตาม ในรูปแบบด้านซ้ายหมายความว่าจำนวนหนังสือต่อผู้แต่งน้อย ขณะที่ในรูปแบบด้านขวาหมายความว่าจำนวนหนังสือต่อผู้แต่งมาก เนื่องจากในรูปแบบด้านซ้าย **ขนาดของ array จะเพิ่มสูงมาก** ถ้าจำนวนหนังสือต่อผู้แต่งมาก

Author

```
{  
  "_id" : 2,  
  "name" : "Hermione Granger",  
  "address" : "Bangkok, Thailand",  
  "books" : [21923, 99683]  
}
```

Book

```
{  
  "_id" : 21923  
  title: "The Basic of SQL",  
  published_date: ISODate("2010-09-24"),  
  pages: 121,  
}  
{  
  "_id" : 99683  
  title: "Java Programming",  
  published_date: ISODate("2015-09-02"),  
  pages: 320,  
}
```



Author

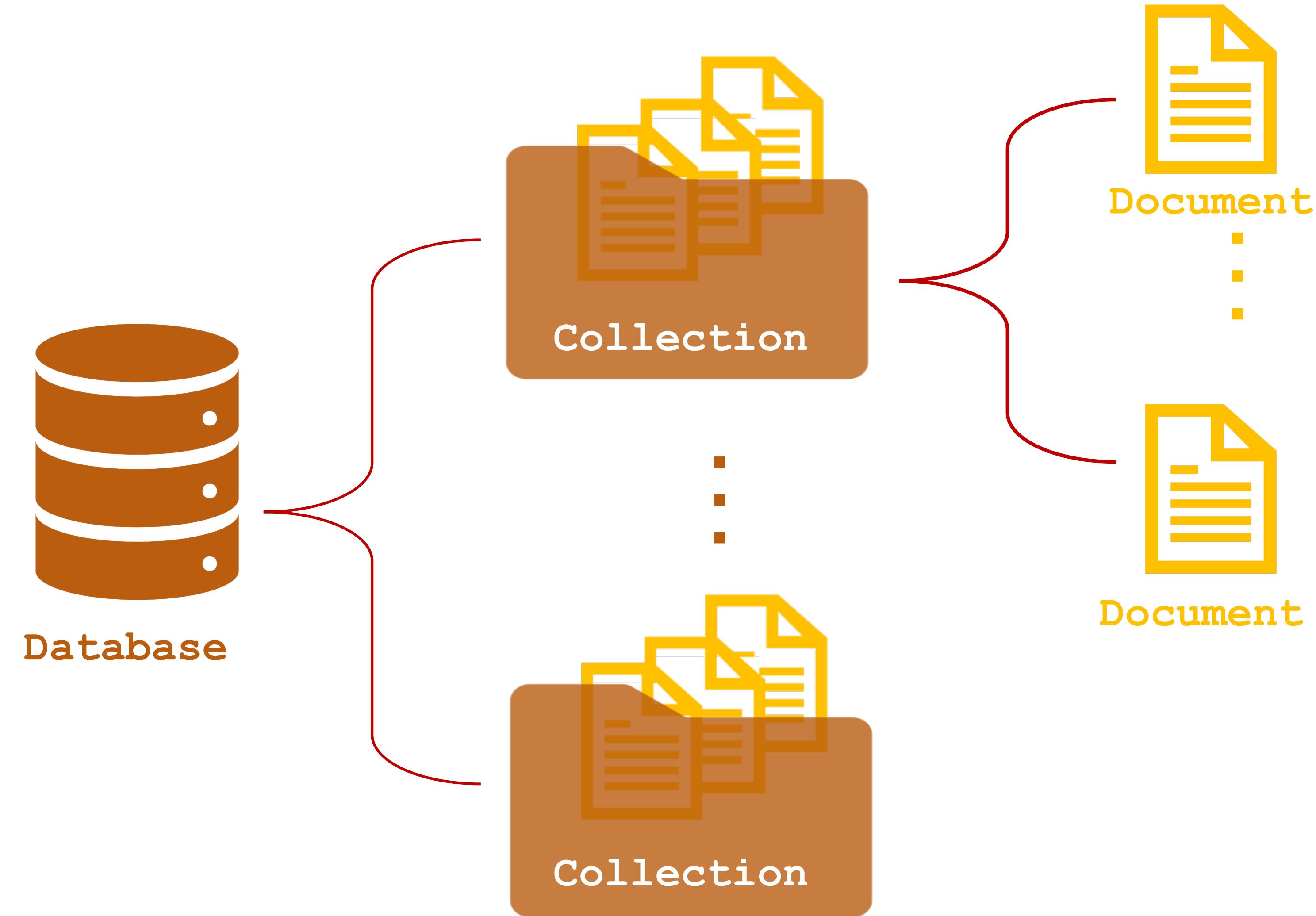
```
{  
  "_id" : 29875,  
  "name" : "Hermione Granger",  
  "address" : "Bangkok, Thailand",  
}
```

Book

```
{  
  "_id" : 21923  
  title: "The Basic of SQL",  
  published_date: ISODate("2010-09-24"),  
  pages: 121,  
  author_id : 29875  
}  
{  
  "_id" : 99683  
  title: "Java Programming",  
  published_date: ISODate("2015-09-02"),  
  pages: 320,  
  author_id : 29875  
}
```

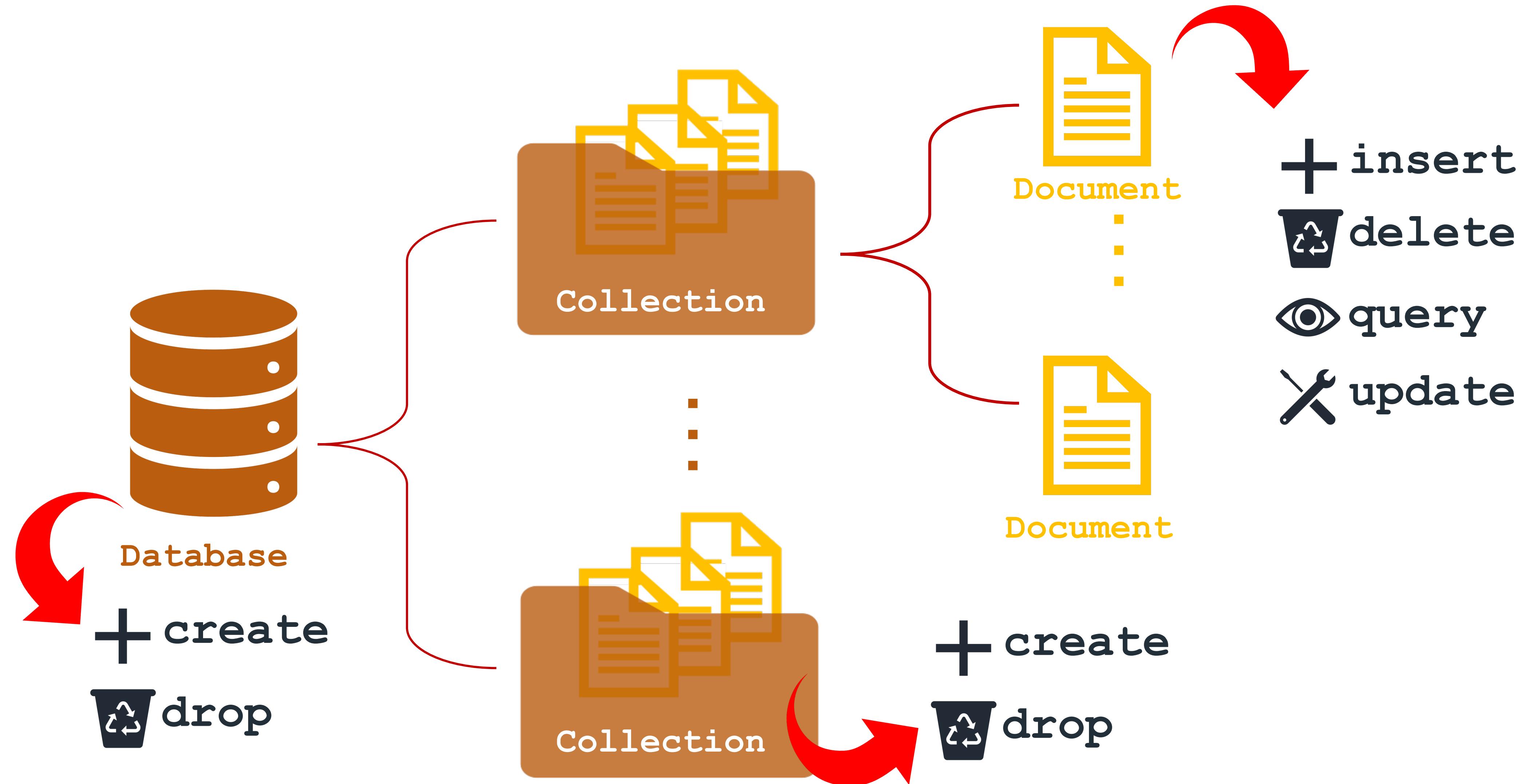


โครงสร้าง MongoDB





โครงสร้าง MongoDB





รายละเอียดเพิ่มเติม

วีดีโอที่ 1: การออกแบบฐานข้อมูลแบบ NoSQL และ Mapping RDBMS to NoSQL (30 นาที)

- <https://www.youtube.com/watch?v=k7cF1ZP63A4...>

วีดีโอ 2 : แนะนำคำสั่ง Query ของ MongoDB (60 นาที)

- <https://www.youtube.com/watch?v=H1lbJqGf3iE...>

วีดีโอ 3 : แนะนำคำสั่ง Query ของ MongoDB (55 นาที)

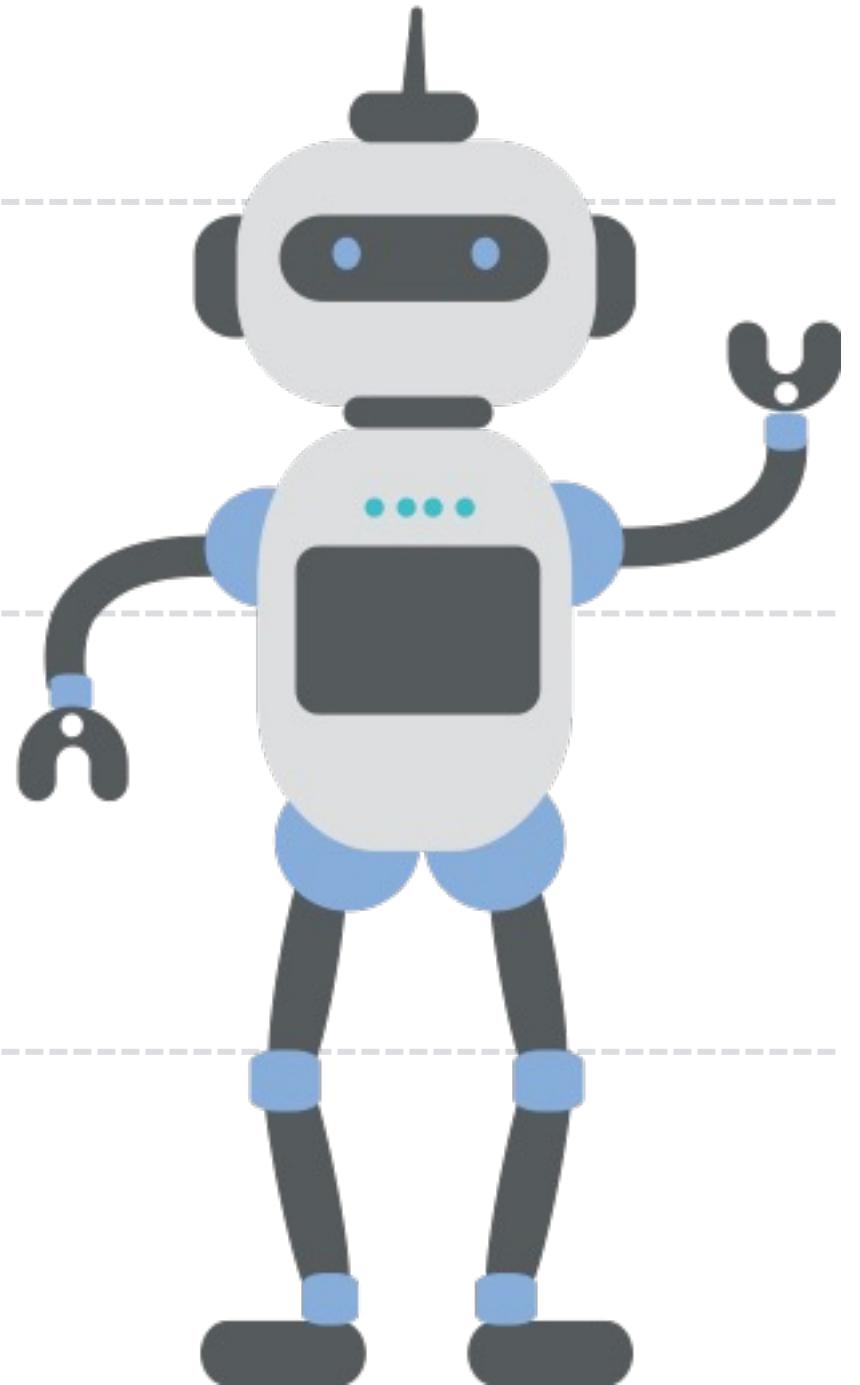
- <https://www.youtube.com/watch?v=7sMwxxyioEE...>

วีดีโอ 4 : แนะนำคำสั่ง Query ของ MongoDB (60 นาที)

- <https://www.youtube.com/watch?v=UP-PHmkEgfc...>

วีดีโอที่ 5 : วิธีการ import ข้อมูล json เข้า MongoDB (4 นาที)

- <https://www.youtube.com/watch?v=oVzcDJByFow...>





วิธีการติดตั้ง MongoDB

วิธีการติดตั้งสำหรับ Windows

- MongoDB Community Server สามารถดาวน์โหลดได้จาก <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>
- สามารถดูวิธีติดตั้ง <https://www.youtube.com/watch?v=f8-0etb-nas>
- หลังจาก install เสร็จแล้วให้ใช้คำสั่ง mongo ในการเข้าไปเขียนคำสั่งใน cmd

วิธีการติดตั้งสำหรับ MAC

- MongoDB Community Server สามารถดาวน์โหลดได้จาก <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>
- ที่อยู่สำหรับเพิ่ม Path สำหรับบางเครื่องบน MAC "export PATH=\$PATH:/usr/local/opt/mongodb-community/bin"
- หลังจาก install เสร็จแล้วให้ใช้คำสั่ง mongo ในการเข้าไปเขียนคำสั่งใน terminal

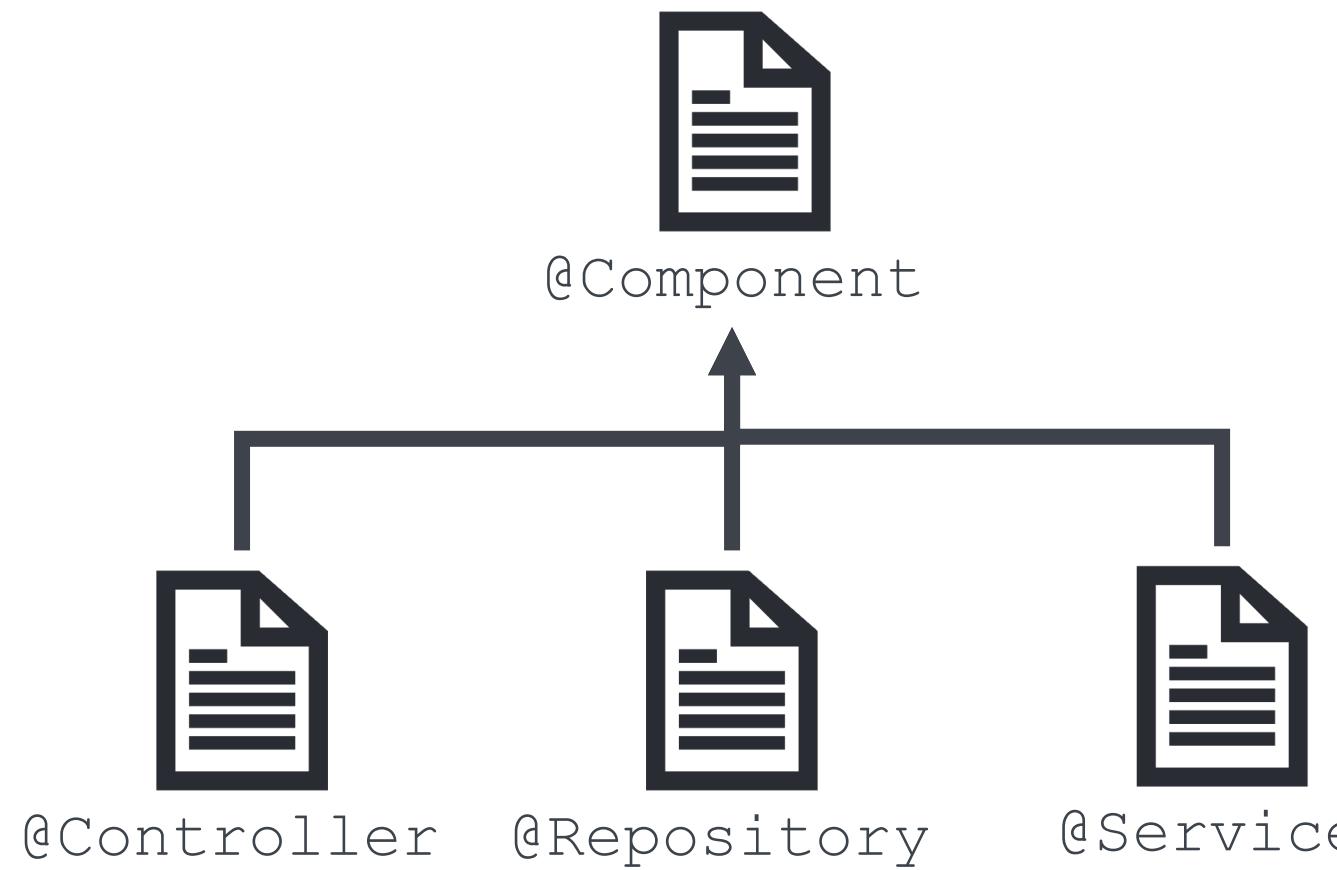


Outline

- แนวทางการออกแบบระบบด้วย Repository Pattern
- แนวคิดของฐานข้อมูลแบบ NoSQL และการใช้งานฐานข้อมูล MongoDB
- การโปรแกรม Spring Boot สำหรับเชื่อมต่อฐานข้อมูล MongoDB แบบ Repository Pattern



Spring Bean



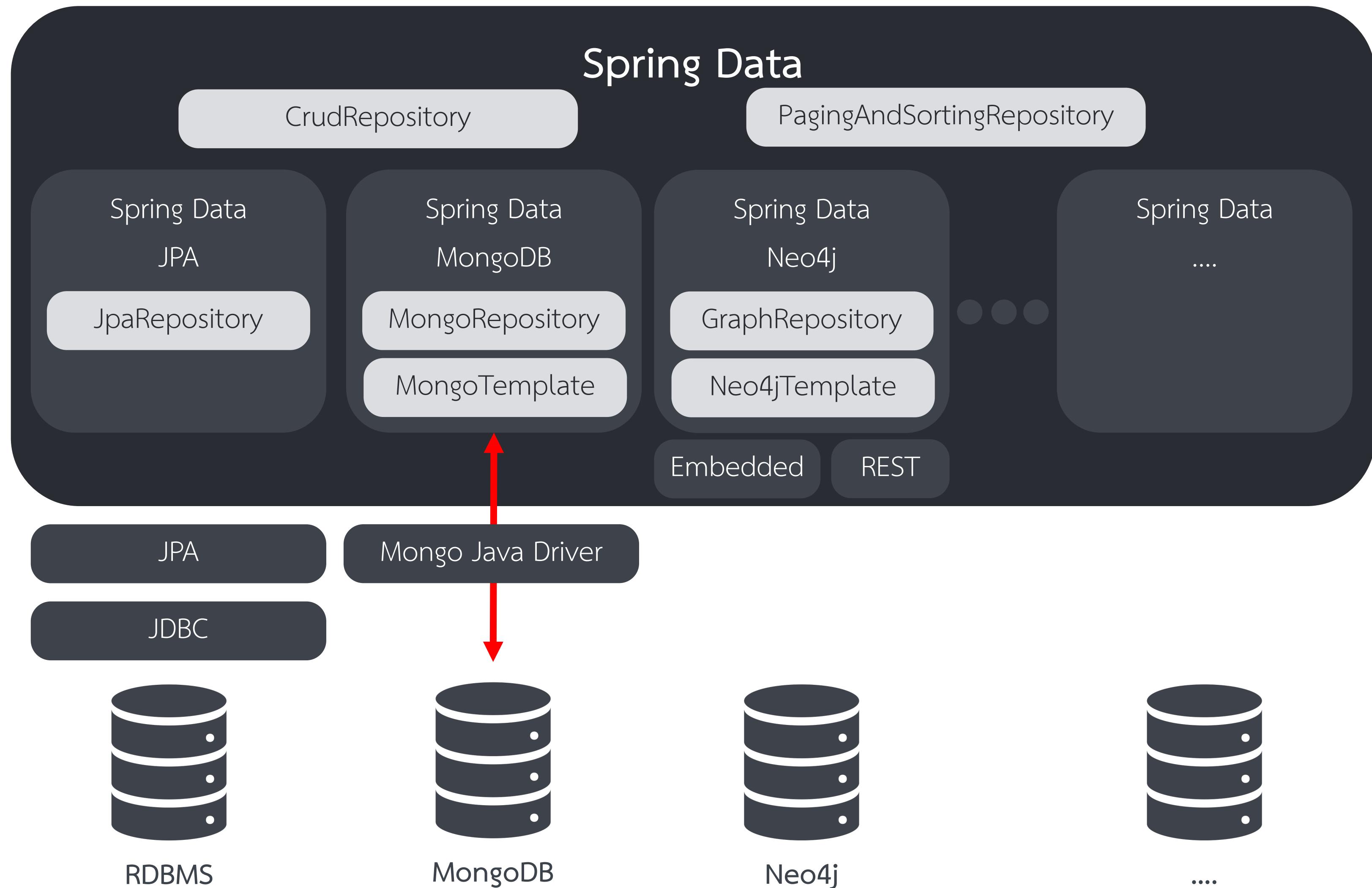
@Autowired เป็น Annotation ที่ใช้บ่งบอก Spring Framework ว่าให้หา Object ที่เกี่ยวข้องหรือสร้างใหม่มากำหนดให้ ซึ่งหมายความว่าเราสามารถใช้งานได้ ถึงแม้ว่าเราจะยังไม่ได้สร้าง Object ดังกล่าว ซึ่งการทำงานแบบจะเรียกว่า “Inject object” หรือ “Dependency Injection”

คลาส Java ใน Spring Framework ที่รองรับคุณสมบัติ Dependency Injection (หมายความว่า “คลาสดังกล่าวสามารถ Inject Object มาใช้งานได้ เพียงประกาศ object ดังกล่าวไว้ก่อนเท่านั้น”) จะถูกเรียกว่า "Spring Bean" ประกอบด้วย 4 ชนิด ได้แก่ (1) ทั่วไป (2) ส่วนติดต่อผู้ใช้งาน (3) จัดการข้อมูลและ (4) แนวคิดทางธุรกิจ เราสามารถกำหนดได้ด้วย 4 Annotation ระดับคลาสดังนี้

Annotation	คุณสมบัติ
@Component	การระบุว่า Object ของคลาสนี้เป็น Spring Bean แบบทั่วไป
@Controller	การระบุว่า Object ของคลาสนี้เป็น Spring Bean ที่มีหน้าที่จัดการส่วน Presentational Layer หรือ View
@Repository	การระบุว่า Object ของคลาสนี้เป็น Spring Bean ที่มีหน้าที่จัดการส่วน Persistence Layer หรือ กίng ๆ Model
@Service	การระบุว่า Object ของคลาสนี้เป็น Spring Bean ที่มีหน้าที่จัดการส่วน Business Layer หรือ Controller



Spring Data



จุดประสงค์ของ Spring Data เพื่อต้องลดจำนวน Boilerplate Code ลงในส่วนของการเชื่อมต่อฐานข้อมูลลง



อินเตอร์เฟส CrudRepository

CrudRepository เป็นอินเตอร์เฟสระดับสูงที่ได้รับการออกแบบมาสำหรับการ Create-Read-Update-Delete (CRUD) กับ

เมธอด	คำอธิบาย
count ()	คืนค่าเป็นจำนวน Document ทั้งหมด
delete (T entity)	ลบ Document ที่สอดคล้องกับเงื่อนไขที่กำหนด
deleteAll ()	ลบทุก Document
deleteAll (Iterable<? extends T> entities)	ลบทุก Document ที่สอดคล้องกับเงื่อนไขที่กำหนด
deleteAllById (Iterable<? extends ID> ids)	ลบทุก Document ที่มี ID สอดคล้องกับที่กำหนด
deleteById (ID id)	ลบ Document ที่มี ID สอดคล้องกับที่กำหนด
existsById (ID id)	ตรวจสอบว่าพบ Document ที่มี ID สอดคล้องกับที่กำหนดหรือไม่
findAll ()	ค้นหาและคืนค่าข้อมูลทุก Document
findAllById (Iterable<ID> ids)	ค้นหาและคืนค่าข้อมูลทุก Document ที่มี ID สอดคล้องกับที่กำหนด
findById (ID id)	ค้นหาและคืนค่าข้อมูล Document ที่มี ID สอดคล้องกับที่กำหนด
save (S entity)	เพิ่ม Document ใหม่หรือปรับปรุง Document ลงใน Collection ที่กำหนด
saveAll (Iterable<S> entities)	เพิ่มหลาย Document ใหม่หรือปรับปรุง Document ลงใน Collection ที่กำหนด



อินเตอร์เฟส PagingAndSortingRepository

PagingAndSortingRepository เป็นอินเตอร์เฟสระดับสูงเช่นกัน ที่ช่วยอำนวยความสะดวกในการเข้าถึงหรือเชื่อมต่อข้อมูล ซึ่งประกอบไปด้วยเมธอดที่น่าสนใจดังนี้

เมธอด	คำอธิบาย
findAll (Example<S> example)	ค้นหาและคืนค่าข้อมูลทุก Document ที่สอดคล้องกับเงื่อนไขที่กำหนด
findAll (Sort sort)	ค้นหาและคืนค่าข้อมูลทุก Document โดยจัดเรียงข้อมูลตามที่กำหนด



อินเตอร์เฟส MongoRepository

MongoRepository เป็นอินเตอร์เฟสสำหรับ Repository เนื่องทางที่หมายรวมกันฐานข้อมูล MongoDB และยังช่วยให้เราสื่อสารกับ Spring Data JPA ที่ทำหน้าสร้างคำสั่งคิวรีข้อมูลใน MongoDB ให้อัตโนมัติจากส่วนหัวของเมธอดที่ประกาศไว้ในอินเตอร์เฟส

เมธอด	คำอธิบาย
findAll()	ค้นหาและคืนค่าข้อมูลทุก Document
findAll(Example<S> example)	ค้นหาและคืนค่าข้อมูลทุก Document ที่สอดคล้องกับเงื่อนไขที่กำหนด
findAll(Example<S> example, Sort sort)	ค้นหาและคืนค่าข้อมูลทุก Document ที่สอดคล้องกับเงื่อนไขที่กำหนด โดยจัดเรียงข้อมูลตามที่กำหนด
findAll(Sort sort)	ค้นหาและคืนค่าข้อมูลทุก Document โดยจัดเรียงข้อมูลตามที่กำหนด
insert(Iterable<S> entities)	เพิ่มหลาย Document ใหม่ลงใน Collection ที่กำหนด
insert(S entity)	เพิ่ม Document ใหม่รายการเดียวลงใน Collection ที่กำหนด
saveAll(Iterable<S> entities)	เพิ่มหลาย Document ใหม่หรือปรับปรุง Document ลงใน Collection ที่กำหนด

โดยคำสั่ง insert() และ save() มีความแตกต่างกันดังนี้

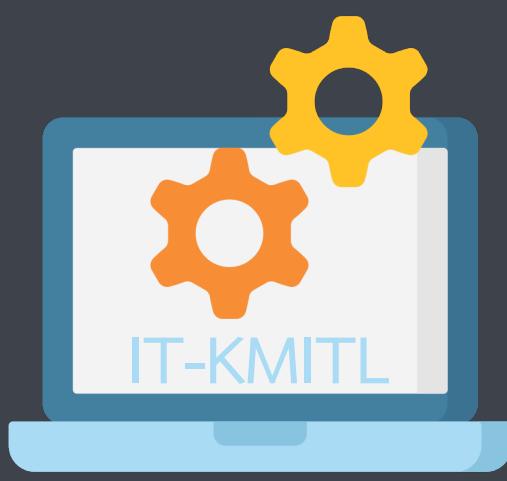
- คำสั่ง insert() จะเพิ่ม Document ใหม่ลงใน Collection ถ้าไม่พบ key ดังกล่าว แต่จะเกิดข้อผิดพลาดขึ้นถ้าพบ key
- คำสั่ง save() จะเพิ่ม Document ใหม่ลงใน Collection ถ้าไม่พบ key ดังกล่าว แต่จะแทนที่ Document เก่าด้วย Document ใหม่ถ้าพบ key



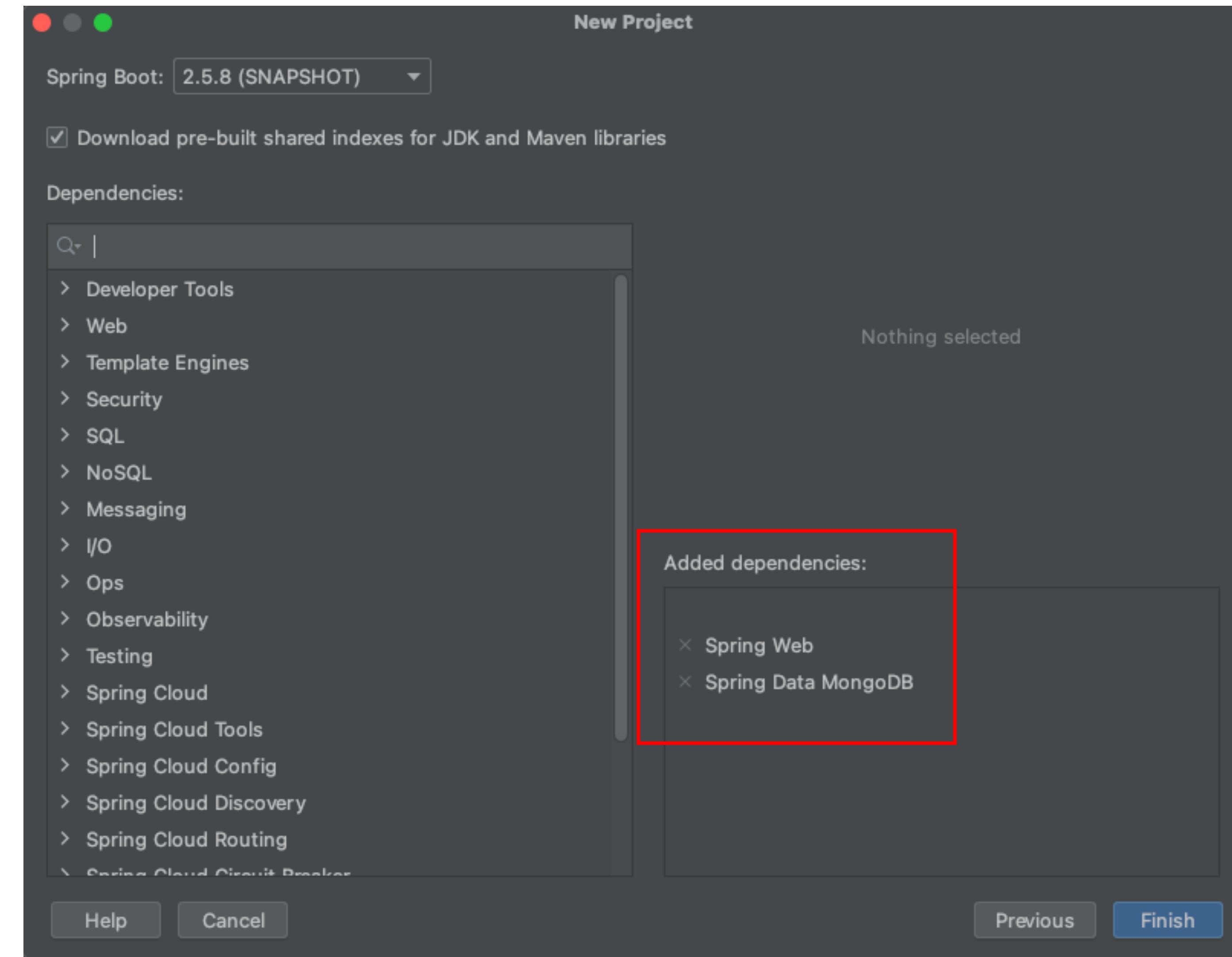
Annotation ที่เกี่ยวกับ Spring Data

Annotation ที่น่าสนใจและเกี่ยวข้องกับการเชื่อมต่อฐานข้อมูลประกอบด้วย

Annotation	คุณสมบัติ
@Document(collection = "")	ใช้สำหรับกำหนดว่าให้ Object ที่สร้างมาจากคลาสนั้นเป็น Domain Object สำหรับ Repository นี้
@Id	ใช้กำหนดว่าแอ็ตทริบิว้นนั้นเป็น Key
@Query(...)	<p>ใช้กำหนดเงื่อนไขและรูปแบบของข้อมูลได้ โดยมีรายละเอียดดังนี้</p> <ul style="list-style-type: none">sort ใช้สำหรับการจัดเรียงข้อมูลตามแอ็ตทริบิวท์กำหนดไว้ อาทิ เช่น<div style="border: 1px solid black; padding: 5px;"><pre>@Query(sort = "{ weight : -1, height : 1 }")</pre></div>fields ใช้กำหนดแอ็ตทริบิวท์ต้องการ อาทิ เช่น<div style="border: 1px solid black; padding: 5px;"><pre>@Query(fields = "{ '_id':0, 'price':1 }")</pre></div>count ใช้นับจำนวน document ที่สอดคล้อง อาทิ เช่น<div style="border: 1px solid black; padding: 5px;"><pre>@Query(count = true)</pre></div>value ใช้กำหนดเงื่อนไขของการค้นหาข้อมูล<div style="border: 1px solid black; padding: 5px;"><pre>@Query(value="{ 'name' : ?0, 'category' : ?1 }") @Query(value="{ name:'?0' }", sort = "{price : -1}", fields = "{ 'name':1, 'price'=1 }") @Query(value="{ name:'?0' }", fields = "{ '_id':0, 'price':1 }") @Query(value="{ price: {\$gt:?0} }", count = true)</pre></div>
@Data	ใช้กำหนดว่าคลาสนี้เป็น Plain Old Java Object (POJO) และรวมการทำงานของ @Setter, @Getter, @EqualsAndHashCode @RequiredArgsConstructor, @ToString เข้าด้วยกัน



ขั้นที่ 1 การจัดการ Dependencies





ขั้นที่ 2 กำหนดค่า application.properties

```
// กำหนดที่อยู่ของ Database  
spring.data.mongodb.host=localhost  
  
// กำหนด Port ที่ใช้เชื่อมต่อกับฐานข้อมูล  
spring.data.mongodb.port=27017  
  
// กำหนดชื่อฐานข้อมูลที่จะติดต่อไป  
spring.data.mongodb.database=SOA
```



ขั้นที่ 3 สร้าง POJO (หรือ Entity Class)

```
@Data // เพื่อมงบอกว่าคลาสนี้เป็น Plain Old Java Object (POJO)
@Document("Book") // เพื่อกำหนด collection name ที่คลาสนี้จะเป็น Data Model
public class Book {
    @Id // ใช้มงบอกว่าแอ็ตทริบิวต์เป็นคีย์ และสร้างให้รองรับ ObjectId ของ MongoDB
    private String _id;
    private String name;
    private String category;
    private int price;

    public Book() {}
    public Book(String _id, String name, String category, int price) {
        this._id = _id;
        this.name = name;
        this.category = category;
        this.price = price;
    }
}

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
```



ขั้นที่ 4 สร้าง Repository แบบ Interface

```
@Repository //ใช้ระบุว่า Object ของคลาสนี้เป็น Spring Bean มีหน้าที่จัดการส่วน Persistence Layer
public interface BookRepository extends MongoRepository<Book, String> {
    (POJO)

    @Query(value = "{name: '?0'}")
    public Book findByName(String name);

    @Query(value = "{name: '?0'}", fields = "{ '_id':0, 'price':1 }")
    public Book findByName2(String name);

    @Query(value = "{price: {$gt: ?0}}", count = true)
    public Integer countMoreThanX(int num);

    public Book updateBook(Book book);
}
```



ขั้นที่ 5 สร้าง Service (หรือ RepositoryImpl)

```
@Service  
public class BookService {  
    @Autowired  
    private BookRepository repository;  
  
    public BookService(BookRepository repository) {  
        this.repository = repository;  
    }  
    public List<Book> retrieveBooks() {  
        return repository.findAll();  
    }  
    public Book retrieveBookByName(String name) {  
        return repository.findByName(name);  
    }  
    public Book retrieveBookByName2(String name) {  
        return repository.findByName2(name);  
    } // ต่อหน้าถัดไป
```



ขั้นที่ 5 สร้าง Service (หรือ RepositoryImpl)

```
public Book createBook(Book book) {  
    return repository.save(book);  
}  
public boolean deleteBook(Book book) {  
    try { repository.delete(book); return true; }  
    catch (Exception e) { return false; }  
}  
public int countBook() {  
    return (int)repository.count();  
}  
public int countBookPriceMoreThanX(int x) {  
    return repository.countMoreThanX(x);  
}  
public Book updateBook(Book book) {  
    return repository.save(book);  
}  
}
```



ขั้นที่ 6 สร้าง RestController

```
@RestController  
public class BookController {  
    @Autowired  
    private BookService bookService ;  
  
    // หน้าถัดไป  
}
```



ขั้นที่ 6 สร้าง RestController (Query)

```
@RequestMapping(value = "/books", method = RequestMethod.GET)
public ResponseEntity<?> getBooks() {
    List<Book> books = bookService.retrieveBooks();
    return ResponseEntity.ok(books);
    // การสร้าง ResponseEntity (คือ Object ข้อมูลที่จะคืนค่าออกไป) และกำหนด status เป็น 200OK
}

@RequestMapping(value = "/book/{name}", method = RequestMethod.GET)
public ResponseEntity<?> getBook(@PathVariable("name") String name) {
    Book book = bookService.retrieveBookByName(name);
    return ResponseEntity.ok(book);
}

@RequestMapping(value = "/book2/{name}", method = RequestMethod.GET)
public ResponseEntity<?> getBook2(@PathVariable("name") String name) {
    Book n = bookService.retrieveBookByName2(name);
    return ResponseEntity.ok(n);
}
```



ขั้นที่ 6 สร้าง RestController (Insert)

```
@RequestMapping(value = "/newBook/{name}/{group}/{price}" , method = RequestMethod.GET)
public ResponseEntity<?> createBook(@PathVariable("name") String name,
                                         @PathVariable("group") String group,
                                         @PathVariable("price") int price) {

    Book n = bookService.createBook(new Book(null, name, group, price)) ;

    return ResponseEntity.ok(n);
}
```



ขั้นที่ 6 สร้าง RestController (Delete)

```
@RequestMapping(value = "/deleteBook/{name}", method = RequestMethod.GET)
public boolean deleteBook(@PathVariable("name") String name) {

    Book book = bookService.retrieveBookByName(name);

    boolean status = bookService.deleteBook(book);

    return status;
}
```



ขั้นที่ 6 สร้าง RestController (Count)

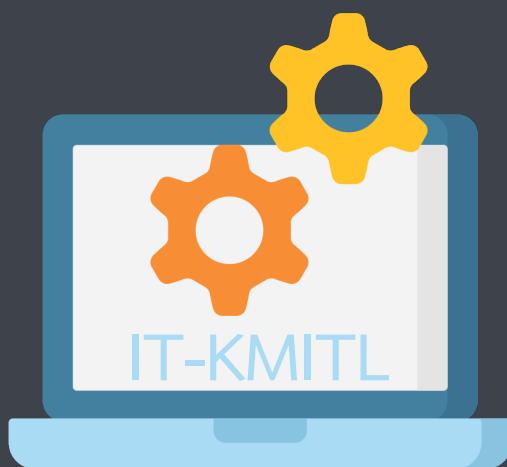
```
@RequestMapping(value = "/countBook", method = RequestMethod.GET)
public int countBook() {
    return bookService.countBook();
}

@RequestMapping(value = "/countBook/{x}", method = RequestMethod.GET)
public int countBookMoreThanX(@PathVariable("x") int x) {
    return bookService.countBookPriceMoreThanX(x);
}
```



ขั้นที่ 6 สร้าง RestController (Update)

```
@RequestMapping(value = "/updateBook/where/{nameOld}/{nameNew}/{group}/{price}" ,  
               method = RequestMethod.GET)  
public boolean updateBook(@PathVariable("nameOld") String nameOld,  
                         @PathVariable("nameNew") String nameNew,  
                         @PathVariable("group") String group,  
                         @PathVariable("price") int price) {  
  
    Book book = bookService.retrieveBookByName(nameOld) ;  
    if(book != null) {  
  
        bookService.updateBook(new Book(book.get_id() , nameNew, group, price)) ;  
        return true;  
  
    }else {  
        return false;  
    }  
}
```



ตัวอย่างข้อมูลใน MongoDB

MongoDB Compass - localhost:27017

SOA

Local

6 DBS 4 COLLECTIONS C

CREATE DATABASE

Database Name Storage Size Collections Indexes

HH	4.1KB	1	1	
ITShop	20.5KB	1	1	
SOA	36.9KB	1	1	

HOST
localhost:27017

CLUSTER
Standalone

EDITION
MongoDB 5.0.3 Community

Filter your data

HH



MongoDB Compass - localhost:27017/SOA

Local

6 DBS 4 COLLECTIONS C

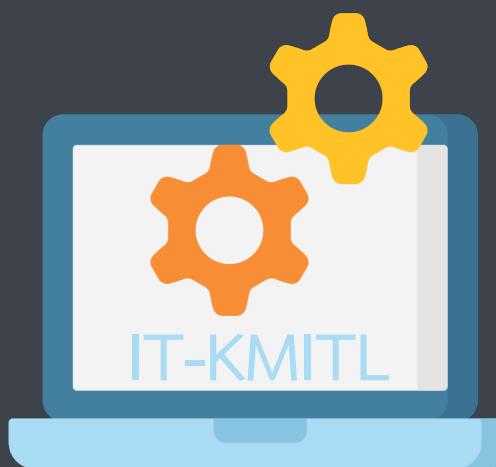
CREATE COLLECTION

Collections

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
Book	10	101.5 B	1.0 KB	1	36.9 KB	

HOST
localhost:27017

CLUSTER
Standalone



ตัวอย่างข้อมูลใน MongoDB

MongoDB Compass - localhost:27017/SOA

Local

HOST
localhost:27017

CLUSTER
Standalone

6 DBS 4 COLLECTIONS

Collections

CREATE COLLECTION

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
Book	10	101.5 B	1.0 KB	1	36.9 KB	

MongoDB Compass - localhost:27017/SOA.Book

Local

HOST
localhost:27017

CLUSTER
Standalone

EDITION
MongoDB 5.0.3 Community

6 DBS 4 COLLECTIONS

SOA.Book Documents

SOA.Book

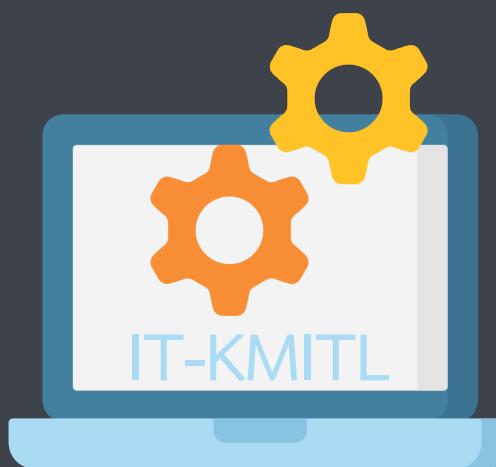
Documents Aggregations Schema Explain Plan Indexes Validation

FILTER { field: 'value' } OPTIONS FIND RESET C REFRESH

ADD DATA VIEW

Displaying documents 1 - 10 of 10

_id	ObjectId	name	category	price	_class
1	619213915c87c9f47bfa9775	"ThaiFood"	"Food"	199	"com.example"
2	619213915c87c9f47bfa9776	"A3"	"cook"	560	No field
3	619213915c87c9f47bfa9777	"ERP"	"Biz"	450	"com.example"
4	619213915c87c9f47bfa9778	"A5"	"art"	850	No field
5	619213915c87c9f47bfa9779	"A6"	"biz"	420	No field
6	619213915c87c9f47bfa977a	"Intro2Drawing"	"Art"	2099	"com.example"



ตัวอย่างการทำงาน

GET <http://localhost:8080/books> Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Headers 8 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
--	-----	-------	-------------	-----	-----------	---------

Body Cookies Headers (5) Test Results Status: 200 OK Time: 215 ms Size: 983 B Save Response

Pretty Raw Preview Visualize

```
[{"_id": "619213915c87c9f47bfa9775", "name": "ThaiFood", "category": "Food", "price": 199}, {"_id": "619213915c87c9f47bfa9776", "name": "A3", "category": "cook", "price": 560}, {"_id": "619213915c87c9f47bfa9777", "name": "ERP", "category": "Biz", "price": 450}, {"_id": "619213915c87c9f47bfa9778", "name": "A5", "category": "art", "price": 850}, {"_id": "619213915c87c9f47bfa9779", "name": "A6", "category": "biz", "price": 420}, {"_id": "619213915c87c9f47bfa977a", "name": "Intro2Drawing", "category": "Art", "price": 2099}, {"_id": "619213915c87c9f47bfa977b", "name": "JapanFood", "category": "cook", "price": 890}, {"_id": "619213915c87c9f47bfa977c", "name": "X-Men", "category": "Cartoon", "price": 1090}, {"_id": "619213915c87c9f47bfa977d", "name": "PowerPubGirl", "category": "Cartoon", "price": 199}, {"_id": "61921e222fcb4c213e9a7e3f", "name": "MathForIT", "category": "Edu", "price": 250}]
```



ตัวอย่างการทำงาน

GET ▼ http://localhost:8080/book/X-Men Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results 🌐 Status: 200 OK Time: 94 ms Size: 247 B Save Response ▼

Pretty Raw Preview Visualize GET

```
{"_id": "619213915c87c9f47bfa977c", "name": "X-Men", "category": "Cartoon", "price": 1090}
```



ตัวอย่างการทำงาน

GET <http://localhost:8080/updateBook/where/DrawingArt/Intro2Drawing/Art/2099> Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 526 ms Size: 168 B Save Response

Pretty Raw Preview Visualize JSON

1 true