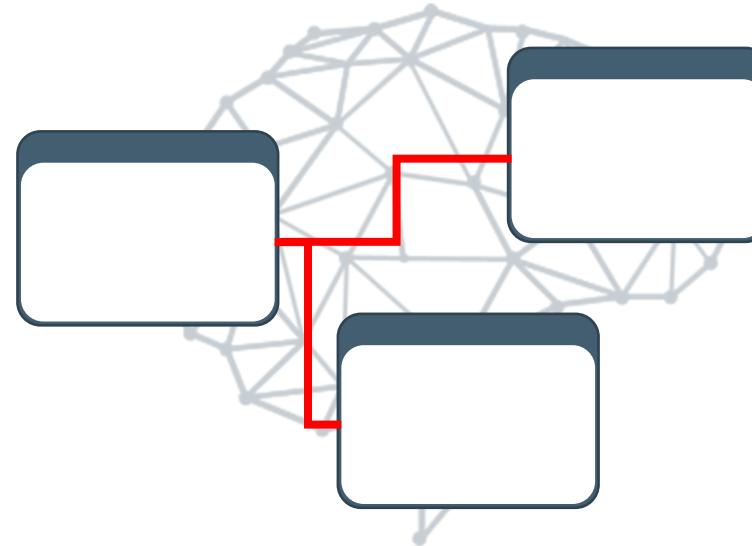




Chapter 12: Data Modeling for MongoDB

By Asst.Dr.Taravichet Titijaroonroj

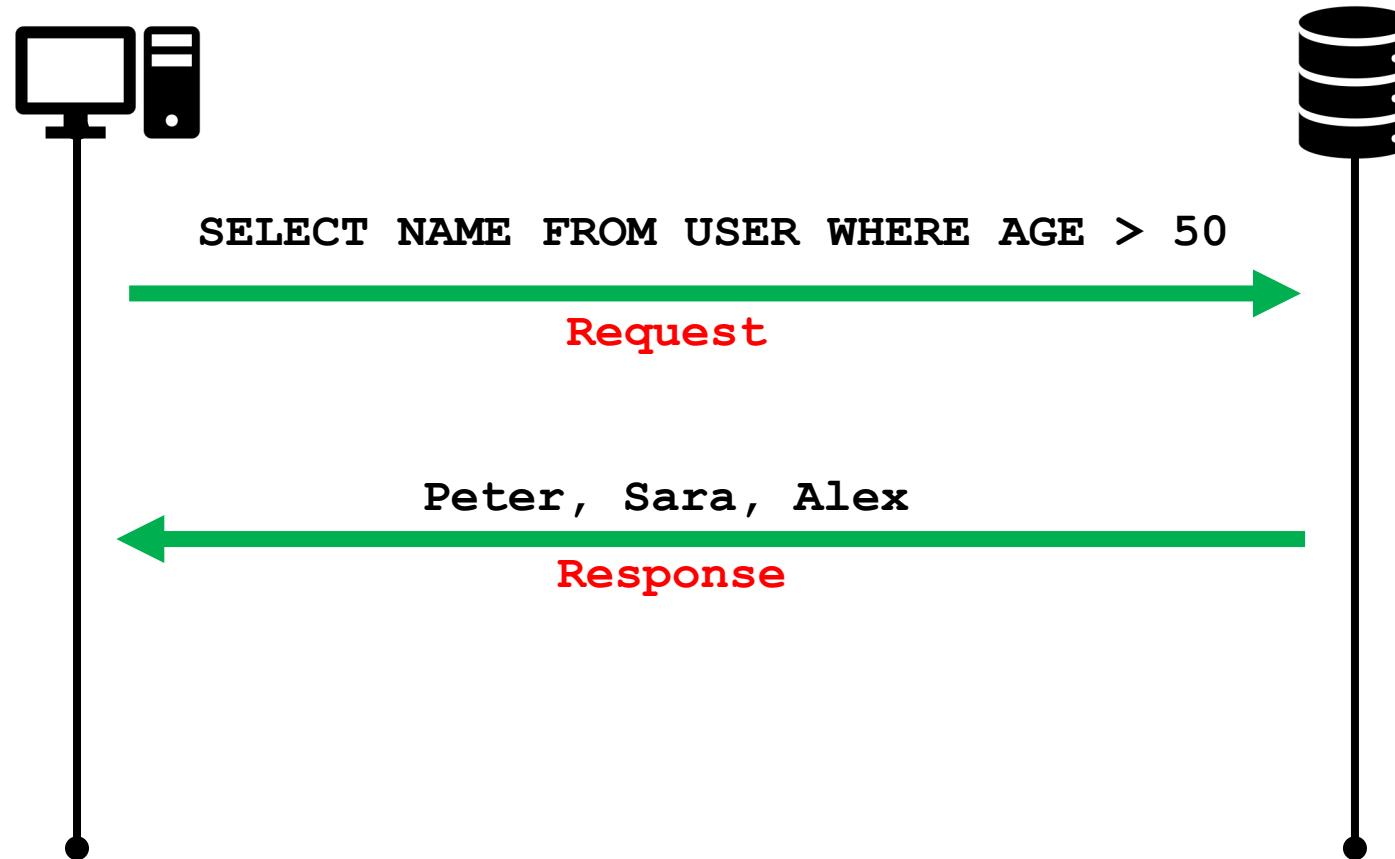
แนวคิดการออกแบบฐานข้อมูล RDMS



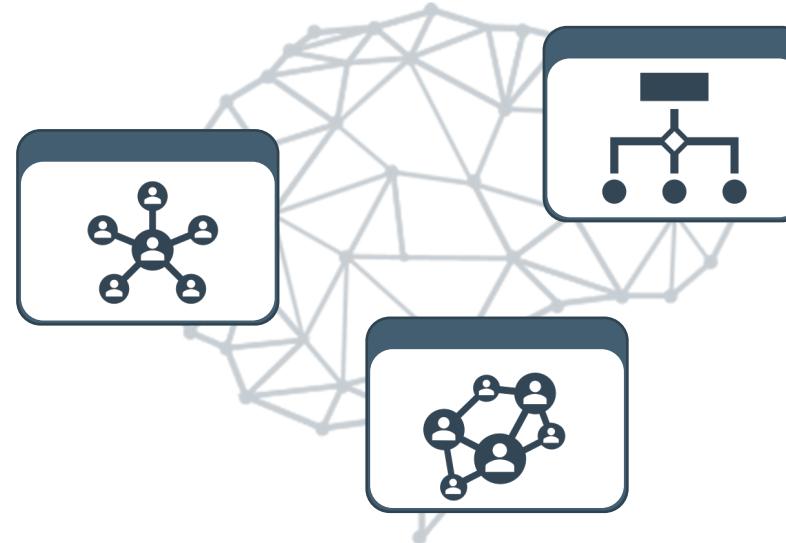
Relational Database

สำหรับ RDMS ผู้พัฒนาสามารถออกแบบ **โครงสร้างข้อมูลให้อยู่ในรูปแบบหรือมาตรฐานที่ต้องการได้โดย** ปราศจากการพิจารณารูปแบบการเรียกใช้งานข้อมูล (Query) อีกทั้งยังสามารถขยายหรือเพิ่มเติมรูปแบบการเรียกใช้งานข้อมูลได้ในภายหลัง

การเรียกใช้งานข้อมูล (Query)



แนวคิดการออกแบบฐานข้อมูล NoSQL



NoSQL Database

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

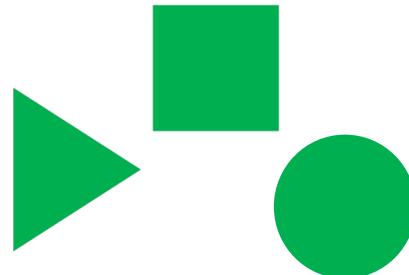
แนวคิดการออกแบบฐานข้อมูล NoSQL

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



Data Size

ต้องทราบจำนวนของข้อมูลที่จะจัดเก็บต่อหนึ่งครั้ง สิ่งนี้สามารถช่วยให้ออกแบบการแบ่งข้อมูลได้มีประสิทธิภาพสูงขึ้น



Data Shape

ในฐานข้อมูลประเภท RDMS บางคำสั่ง Query จะมีการเปลี่ยนรูปร่างหรือสร้างตารางชั่วคราวมาเพื่อประมวลผลบาง Query สิ่งนี้ส่งผลทำให้ใช้เวลาในการประมวลผลที่สูงขึ้นทำให้ประสิทธิภาพด้านเวลา และการเปลี่ยนแปลงขนาดไม่ได้

shapeที่ใช้งานกับshapeที่เก็บควรเป็นshapeเดียวกัน



Data Velocity

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

ทำให้การทำงานมีประสิทธิภาพมากขึ้น

NoSQL

แนวคิดการออกแบบฐานข้อมูล NoSQL

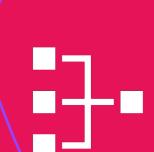
CountryID	CName	yearID	Year	CustID	CustName	CustLastName
1	Thailand	1	1990	1	Sara	Smith
2	Japan	2	2000	2	Alex	Potter

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

ความสัมพันธ์ถูกสะท้อนด้วย FK

Process

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



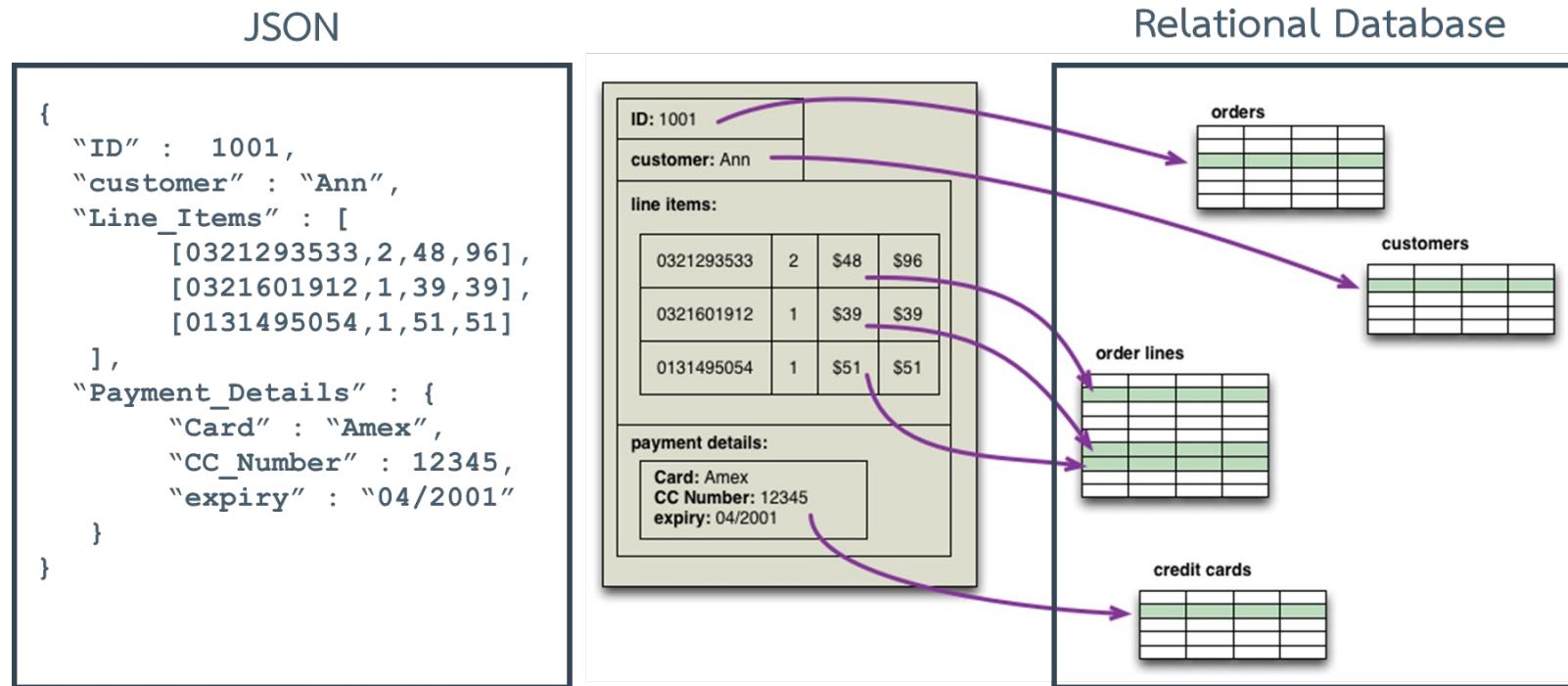
Keep related data together

การจัดเก็บความสัมพันธ์ร่วมกับข้อมูล

ข้อมูลที่ใช้ด้วยการควรอยู่ในตารางเดียวกัน

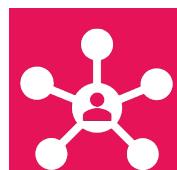
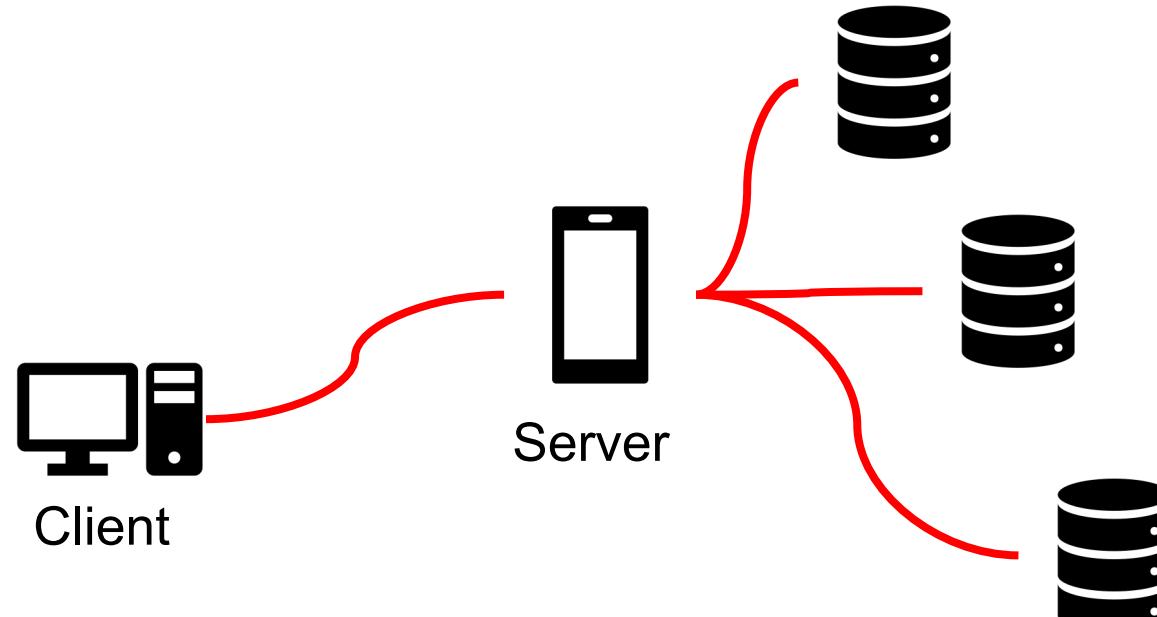
ตลอด 20 ปีที่ผ่านมาการจัดการและเชื่อมต่อข้อมูล (เรียกว่า ความสัมพันธ์ภายใน locality of reference) เป็นปัจจัยที่สำคัญมากที่ส่งผลต่อเวลาในการตอบสนองของฐานข้อมูล ดังนั้น จึงมีแนวคิดที่ว่า **ถ้าสามารถจัดเก็บความสัมพันธ์ภายในและข้อมูลไว้ที่เดียวกันได้จะสามารถลดเวลาในการประมวลผลลงได้**

แนวคิดการออกแบบฐานข้อมูล NoSQL



ซึ่งสอดคล้องกับลักษณะของฐานข้อมูลชนิด NoSQL ขณะที่ฐานข้อมูลชนิด RDBS ความสัมพันธ์ต่าง ๆ และข้อมูลลูกกรະเจัยเก็บในตารางต่าง ๆ จึงทำให้เสียเวลาในการประมวลผล ถึงแม้ว่าการออกแบบฐานข้อมูลที่ดีควรจะออกแบบใหม่ตารางเพียงตารางเดียว แต่อย่างไรก็ตาม ก็อาจจะมีมากกว่าหนึ่งตารางได้ตามความเหมาะสม เช่น มี Query หลากหลายแบบ ข้อมูลที่ถูกแบ่งเก็บเป็นช่วงเวลาที่มีขนาดใหญ่มาก ๆ เป็นต้น

แนวคิดการออกแบบฐานข้อมูล NoSQL



Distribute Queries

การกระจายคำตาม

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

ฐานข้อมูล NoSQL ชนิดเอกสาร



ฐานข้อมูลที่มีแนวคิดจากการออกแบบเพื่อใช้ **จัดเก็บเอกสารที่มีโครงสร้างและรายละเอียดแตกต่างกัน** ซึ่งอาจจะกล่าวได้ว่าในเอกสารได้ ๆ ข้อมูล (Data) หรือสารสนเทศ (Information) จะถูกเข้ารหัส (Encapsulation) เอาไว้ในรูปแบบใดรูปแบบหนึ่ง เช่น XML, JSON, BSON, PHP Array, Python Dictionary, Ruby Hash และ อื่น ๆ

เปรียบเทียบโครงสร้าง MongoDB กับ RDMS



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 ไม่จำเป็นที่โครงสร้างของแต่ Document ต้องเหมือนกัน ขณะที่ฐานข้อมูล RDMS ต้องกำหนดให้แต่ละแถว (row) มีโครงสร้างสอดคล้องกับที่กำหนดไว้ข้างต้น

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

● ● ● Document Schema Design

```

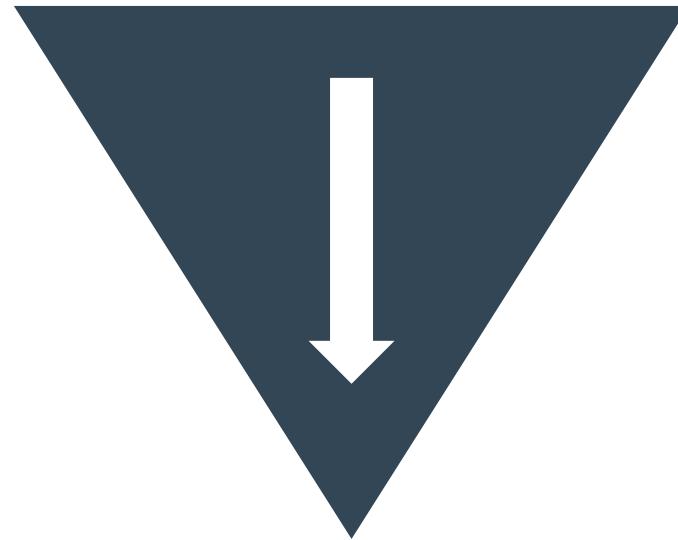
1 {
2   _id : 1,
3   owner : 3,
4   color : "Blue",
5   brand : "Toyota"
6 }
7 {
8   _id : 5,
9   owner : 3,
10  color : 234,
11  brand : "Honda",
12  MP3 : False
13 }
```

แนวทางการออกแบบฐานข้อมูล MongoDB

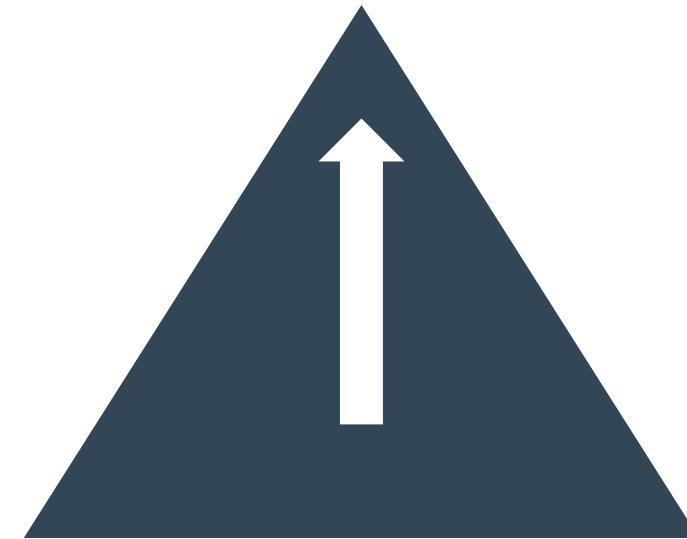


- หลักเลี้ยงแนวคิดหรือวิธีการออกแบบที่อาศัยความสัมพันธ์
- คาดการณ์เหตุการณ์ที่จะเกิดขึ้นเมื่อโครงสร้างของฐานข้อมูลถูกปรับเปลี่ยนในภายหลัง
- พิจารณาขนาดของข้อมูล คีย์ และไฟล์เอกสาร
- รูปแบบการสอบถาม (Query)

แนวทางการออกแบบฐานข้อมูล MongoDB



การ Normalization โดยอาศัยหลักการ
Document Reference

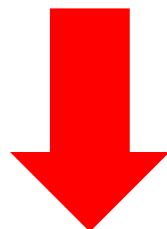


การ Denormalization โดยอาศัยหลักการ
Embedded Document

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"  
    }]  
}
```

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

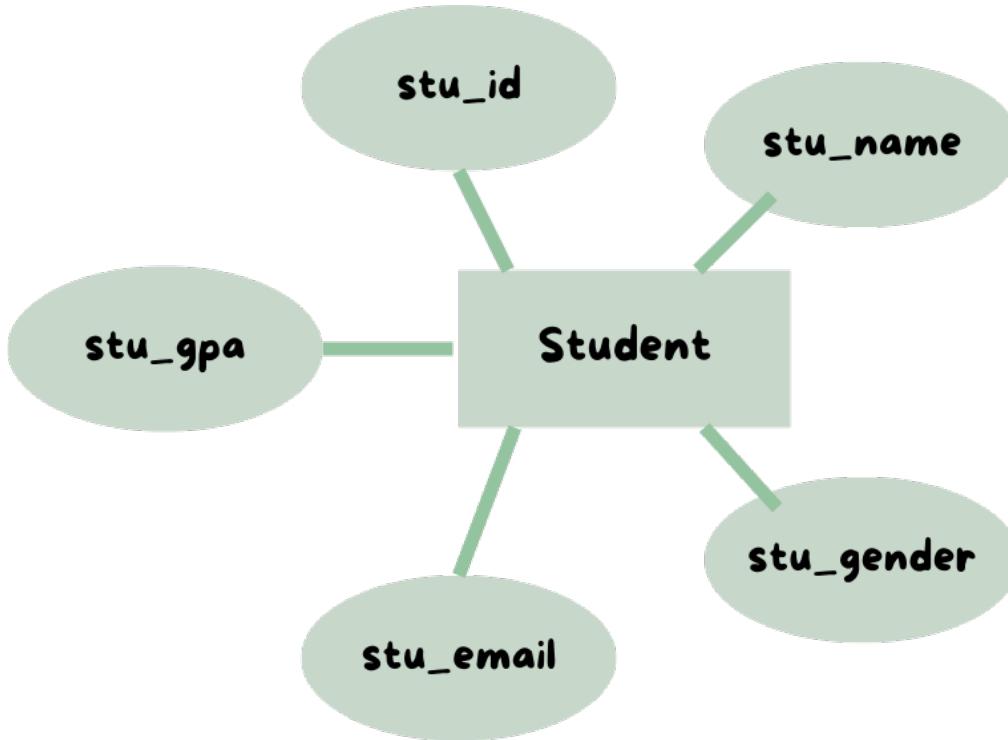
- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

ขั้นตอนที่ 1.1: Identifying and Quantifying Entities



เอนทิตี้ (Entity) คือ ชื่อของสิ่งใดสิ่งหนึ่งไม่ว่าจะเป็น คน, สถานที่, สิ่งของ, การกระทำ และ อื่น ๆ (มักจะพูดเป็นคำ Noun หรือ Gerund) ซึ่งต้องการจัดเก็บข้อมูลไว้ เช่น แผนก พนักงาน สินค้า ลูกค้า ใบเสร็จ เป็นต้น

แอทริบิวต์ (Attribute) หมายถึง รายละเอียดของข้อมูลใน Entity เช่น Entity พนักงาน ประกอบด้วย Attribute ต่าง ๆ อาทิเช่น รหัสพนักงาน, ชื่อ, ที่อยู่, โทรศัพท์, อีเมล เป็นต้น โดยทั่วไปแล้ว Attribute มี 5 ประเภทหลัก ได้แก่ Attribute, Key Attribute, Composite Attribute, Multivalued Attribute, Derived Attribute

ขั้นตอนที่ 1.1: Identifying and Quantifying Entities

author	publisher	eBook	audiobook	printed book
<ul style="list-style-type: none"> + authorId + name + birthYear + biography + socials 	<ul style="list-style-type: none"> + publisherId + name + founded + description + headquarters + socials 	<ul style="list-style-type: none"> + SKU + title + author + publisher + language + price + summary + rating + releaseDate + pages 	<ul style="list-style-type: none"> + SKU + title + author + publisher + language + price + summary + rating + releaseDate + duration + narrators 	<ul style="list-style-type: none"> + SKU + title + author + publisher + language + price + summary + rating + releaseDate + pages + stockLevel + deliveryTime

Database Entities

ขั้นตอนที่ 1.1: Identifying and **Quantifying** Entities

Entity	Quantity
eBooks	450,000
Audiobooks	200,000
Printed books	50,000
Authors	20,000
Publishers	500
Users	25,000,000
Reviews	1,000,000,000

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

ขั้นตอนที่ 1.2: Identifying Reads and Writes

Entities	Operation	Information Needed	Type
Books*	Fetch book details	Book details + rating	Read
Authors, Books*	Fetch an author and their books	Book titles + author details	Read
Print Books	Fetch printed book titles where the stock level has fallen below 50	Book details + stock level	Read
Books*	Add/update book	Book details + stock level	Write
Print Books	Sell copy of printed book	Stock level	Write
Reviews	Fetch 10 reviews for a book	Reviews + reviewer rating	Read

การระบุการดำเนินการ Read และ Write ก่อนที่จะสร้างแบบจำลองข้อมูลเป็นสิ่งสำคัญ เนื่องจากเป็นการช่วยในการระบุข้อมูลที่จำเป็นต้องจัดเก็บไว้ด้วยกัน

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

ขั้นตอนที่ 1.3: Quantifying Reads and Writes

Entities	Operation	Information Needed	Type	Rate
Books*	Fetch book details	Book details + rating	Read	1000/sec
Authors, Books*	Fetch an author and their books	Book titles + author details	Read	50/sec
Print Books	Fetch printed book titles where the stock level has fallen below 50	Book details + stock level	Read	2/day
Books*	Add/update book	Book details + stock level	Write	10/hour
Print Books	Sell copy of printed book	Stock level	Write	5/sec
Reviews	Fetch 10 reviews for a book	Reviews + reviewer rating	Read	200/sec

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

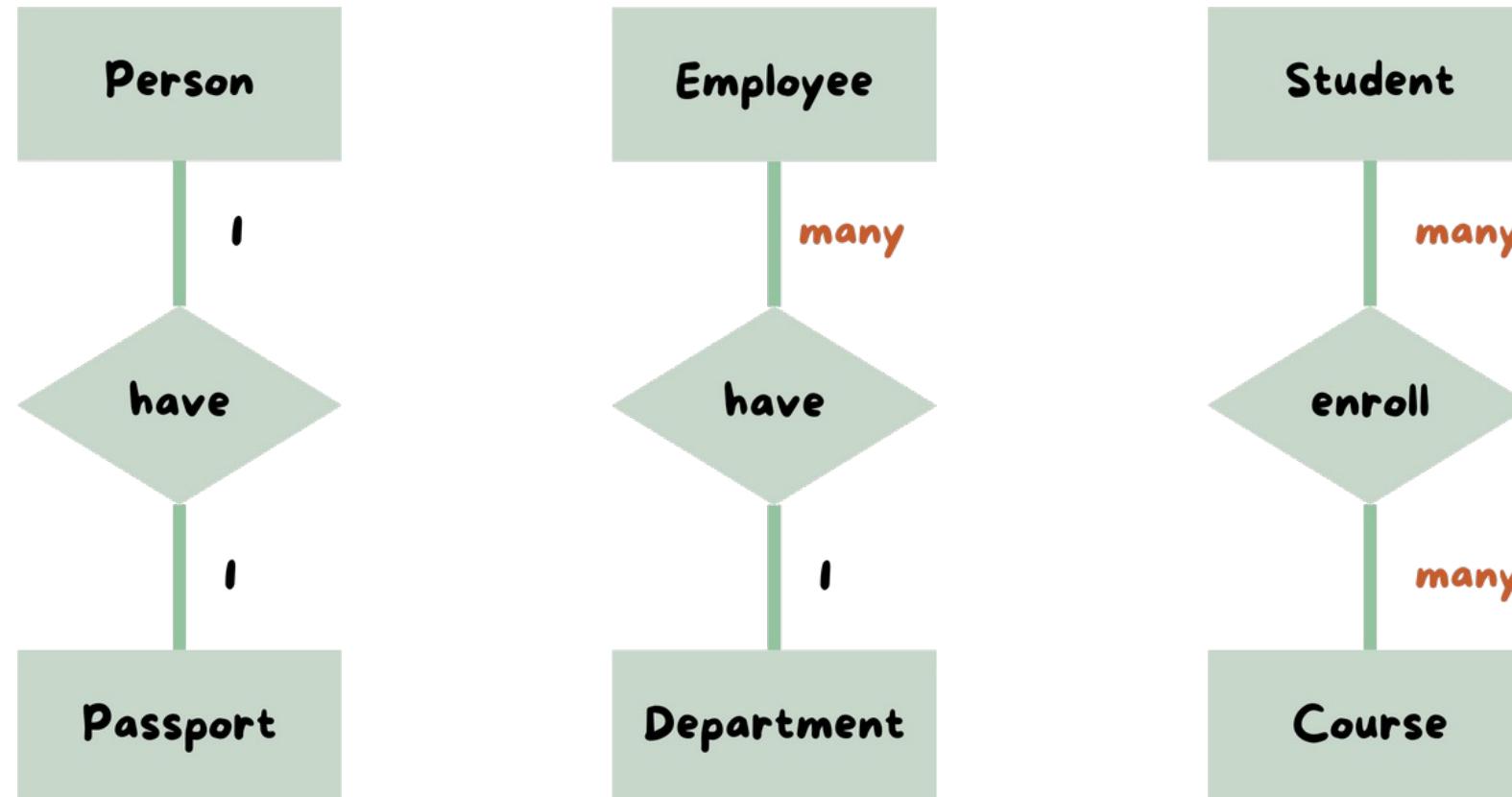
แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

ขั้นตอนที่ 2.1: Identifying Relationships

การระบุความสัมพันธ์ระหว่าง ENTITY ว่าเป็นความสัมพันธ์แบบ ONE TO ONE, ONE TO MANY, หรือ MANY TO MANY



แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

ขั้นตอนที่ 2.2: Embedding or Referencing



Embedded Document



Document Reference

ขั้นตอนที่ 2.2: Embedding or Referencing



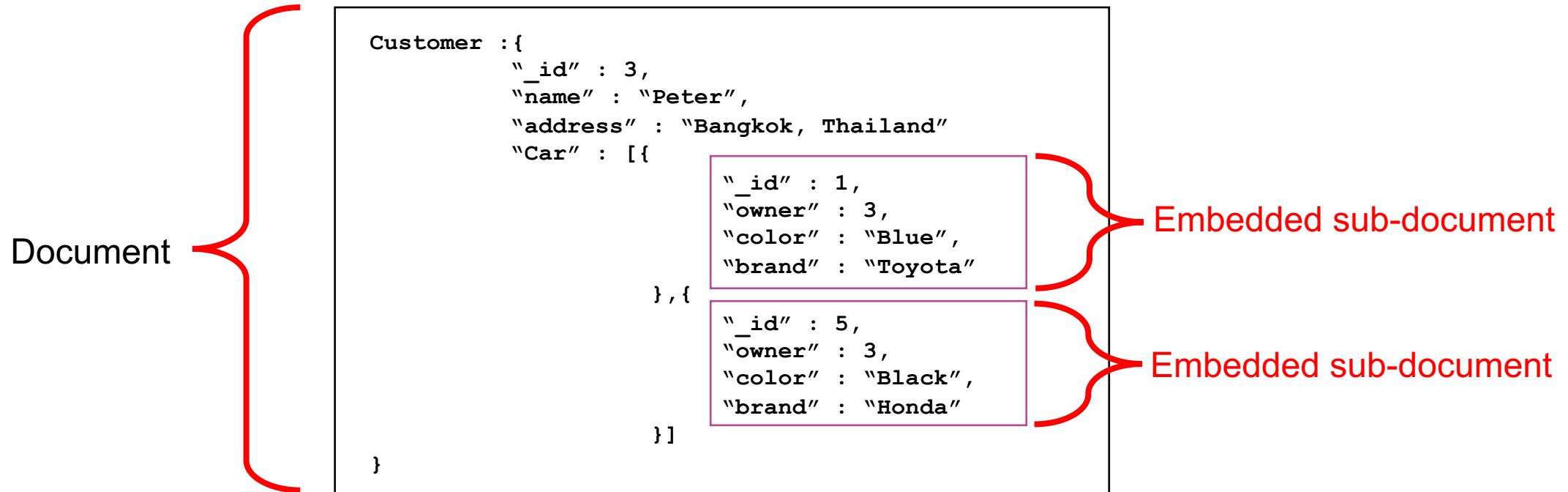
Embedded Document



Document Reference

Embedded Document

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



Embedded Document

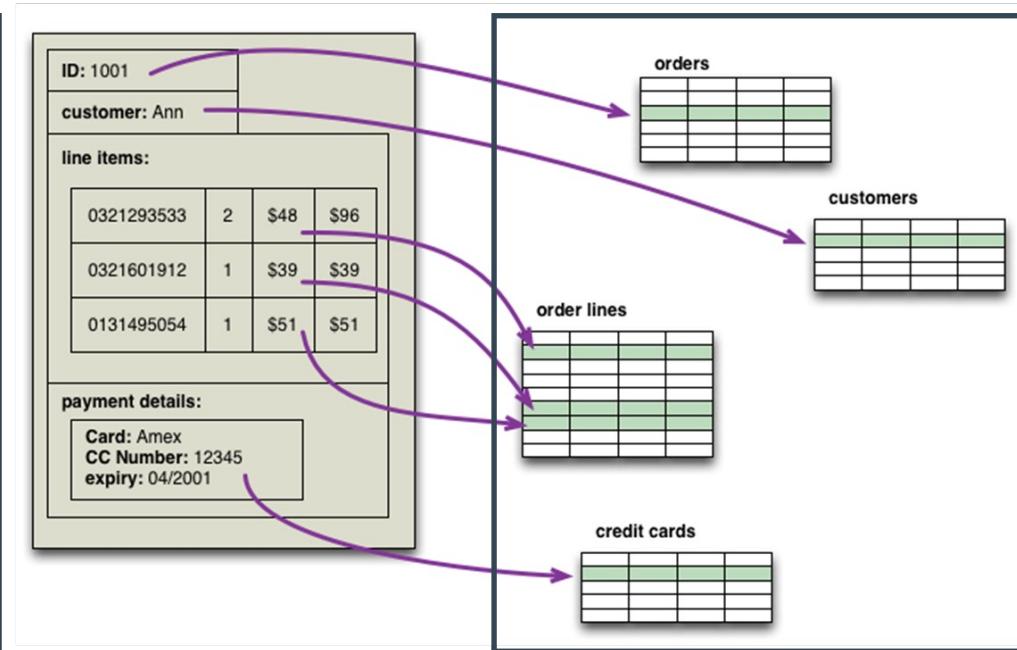


การ **Embedded Document** เป็นวิธีการที่ทำให้การอ่านข้อมูลมีประสิทธิภาพดีขึ้น เนื่องจาก **ข้อมูลที่เกี่ยวข้องกันได้รวมกลุ่มกันเป็นหนึ่งเดียวกัน** ส่งผลให้ประสิทธิภาพในขั้นตอนการขอรับบริการได้รับการ**ตอบสนองที่รวดเร็ว** และข้อมูลที่สัมพันธ์กันสามารถปรับปรุงรูปแบบความสัมพันธ์ได้ โดยไม่ต้องคำนึงถึงโครงสร้าง (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



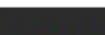
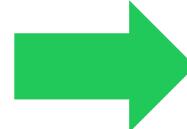
การออกแบบ Collection จากความสัมพันธ์แบบ one-to-one

Embedded Document Pattern ข้อมูล Address มักถูกดึงข้อมูลมาพร้อมกับข้อมูล Customer เพื่อให้ระบบของนักศึกษาสามารถดึงข้อมูลที่จำเป็นทั้งหมดด้วยการสืบค้นเพียงครั้งเดียว ให้ฝังข้อมูลที่อยู่ไว้ในเอกสารด้วยวิธีการ Embedding Document



Document Schema Design

```
1 // patron document
2 {
3   _id: "joe",
4   name: "Joe Bookreader"
5 }
6 // address document
7 {
8   street: "123 Fake Street",
9   city: "Faketown",
10 state: "MA",
11 zip: "12345"
12 }
```



Document Schema Design

```
1 {
2   _id: "joe",
3   name: "Joe Bookreader",
4   address: {
5     street: "123 Fake Street",
6     city: "Faketown",
7     state: "MA",
8     zip: "12345"
9   }
10 }
```

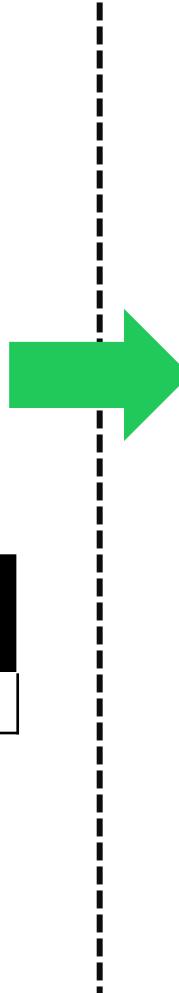
การออกแบบ Collection จากความสัมพันธ์แบบ one-to-one

Person

_id	name
9	Sara Wille

Address

people_id	street	city	zip
9	Rama III Rd.	Bangkok	10120



Document Schema Design

```
1 // Person Collection
2 {
3     _id: 9,
4     name: "Sara Wille"
5 }
6 // Address Collection
7 {
8     people_id: 9,
9     street: "Rama III Rd.",
10    city: "Bangkok",
11    zip: "10120"
12 }
```

การออกแบบ Collection จากความสัมพันธ์แบบ one-to-one

Person

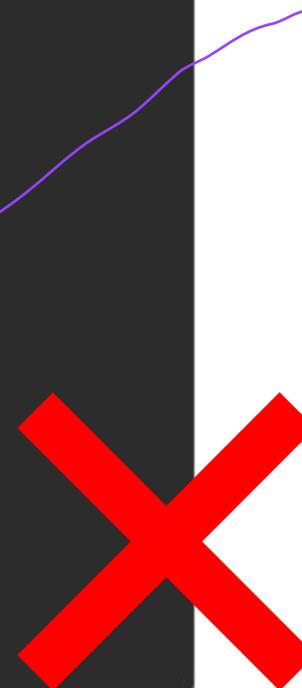
_id	name	street	city	zip
9	Sara Wille	Rama III Rd.	Bangkok	10120



address ควร group ออกมา
เพื่อให้มี meaning ยังอยู่

Document Schema Design

```
1 // Person Collection
2 {
3     _id: 9,
4     name: "Sara Wille",
5     street: "Rama III Rd.",
6     city: "Bangkok",
7     zip: "10120"
8 }
```



การออกแบบ Collection จากความสัมพันธ์แบบ one-to-one

Person

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

เนื่องจากกลุ่มคุณลักษณะดังกล่าวเป็นคุณลักษณะของที่อยู่ (Address) ซึ่งประกอบด้วยคุณลักษณะย่อย ๆ โดย **แต่ละ collection** อาจจะมี **ไม่มี** เมื่อนักออกแบบนี้ ยังเป็นหลักการออกแบบโดยอาศัยเทคนิค Embedded Document นั้นคือ การนำห้องโครงสร้างเดิมซ่อนทับหรือใส่เข้าไปร่วมกับอีกโครงสร้างหนึ่ง

Document Schema Design

```
1 // Person Collection
2 {
3     _id: 9,
4     name: "Sara Wille",
5     Address : {
6         street: "Rama III Rd.",
7         city: "Bangkok",
8         zip: "10120"
9     }
10 }
```

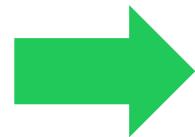


การออกแบบ Collection จากความสัมพันธ์แบบ one-to-many

การใช้ Embedding Document สำหรับข้อมูลที่มีความสัมพันธ์แบบหนึ่งต่ออีกหลาย เป็นการเชื่อมต่อข้อมูลไว้ในเอกสารเดียว สิ่งนี้สามารถลดจำนวนการดำเนินการอ่านที่จำเป็นลงได้ โดยทั่วไป นักศึกษาควรจัดโครงสร้างข้อมูลเพื่อให้ระบบสามารถได้รับข้อมูลที่จำเป็นทั้งหมดในการดำเนินการอ่านเพียงครั้งเดียว

Document Schema Design

```
1 // patron document
2 {
3   _id: "joe",
4   name: "Joe Bookreader"
5 }
6 // address one
7 {
8   street: "123 Fake Street",
9   city: "Faketon",
10 state: "MA",
11 zip: "12345"
12 }
13 // address two
14 {
15   street: "1 Some Other Street",
16   city: "Boston",
17   state: "MA",
18   zip: "12345"
19 }
```



Document Schema Design

```
1 {
2   "_id": "joe",
3   "name": "Joe Bookreader",
4   "addresses": [
5     {
6       "street": "123 Fake Street",
7       "city": "Faketon",
8       "state": "MA",
9       "zip": "12345"
10      },
11      {
12        "street": "1 Some Other Street",
13        "city": "Boston",
14        "state": "MA",
15        "zip": "12345"
16      }
17   ]
18 }
```

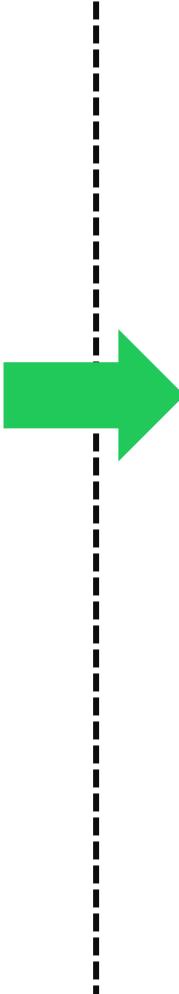
การออกแบบ Collection จากความสัมพันธ์แบบ one-to-many

Person

_id	name
9	Ron Weasley

Address

people_id	soi	road	city	zip
9	-	Rama III Rd.	Bangkok	10120
9	53	-	Bangkok	10150



Document Schema Design

```
1 // Person Collection
2 {
3   _id: 9,
4   name: "Ron Weasley"
5 }
6 // Address Collection
7 {
8   person_id: 9,
9   road: "Rama III Rd.",
10  city: "Bangkok",
11  zip: "10120"
12 }
13 {
14   person_id: 9,
15   soi: "53",
16   city: "Bangkok",
17   zip: "10150"
18 }
```

การออกแบบ Collection จากความสัมพันธ์แบบ one-to-many

Person

_id	name	address			
		soi	road	city	zip
9	Ron Weasley	-	Rama III Rd.	Bangkok	10120
		53	-	Bangkok	10150

Document Schema Design

```
1 // Person Collection
2 {
3     _id: 9,
4     name: "Ron Weasley",
5     address : [
6         {
7             road: "Rama III Rd.",
8             city: "Bangkok",
9             zip: "10120"
10        },
11        {
12            soi: "53",
13            city: "Bangkok",
14            zip: "10150"
15        }
16    ]
17 }
```

ขั้นตอนที่ 2.2: Embedding or Referencing



Embedded Document



Document Reference

การออกแบบ Collection จากความสัมพันธ์แบบ one-to-many

Document Schema Design

```
1 {
2   title: "MongoDB: The Definitive Guide",
3   author: [ "Kristina Chodorow", "Mike Dirolf" ],
4   published_date: ISODate("2010-09-24"),
5   pages: 216,
6   language: "English",
7   publisher: {
8     name: "O'Reilly Media",
9     founded: 1980,
10    location: "CA"
11  }
12}
13{
14  title: "50 Tips and Tricks for MongoDB Developer",
15  author: "Kristina Chodorow",
16  published_date: ISODate("2011-05-06"),
17  pages: 68,
18  language: "English",
19  publisher: {
20    name: "O'Reilly Media",
21    founded: 1980,
22    location: "CA"
23  }
24}
```

การใช้ Document Reference สำหรับข้อมูลที่มีความสัมพันธ์แบบหนึ่งต่ออีกถุ่ม เพื่อลีกเลี้ยงไม่ให้ข้อมูลซ้ำกัน (Repetition Problem) จากการ Embedding Document ที่จะทำให้ข้อมูลซ้ำซ้อนกัน

Document Reference เป็นหนึ่งในวิธีสำหรับการออกแบบฯ จำลองข้อมูล (Data Model Design) ที่อยู่บนพื้นฐานของ Normalization ซึ่งอาศัยการอ้างอิงจาก object หรือ document อันหนึ่งไปสู่อีกอันหนึ่ง เพื่อ **หลีกเลี้ยงการซ้ำซ้อนของข้อมูลที่มากเกินไป**

การออกแบบ Collection จากความสัมพันธ์แบบ one-to-many

Document Schema Design

```
1 {
2   _id: "oreilly",
3   name: "O'Reilly Media",
4   founded: 1980,
5   location: "CA"
6 }
7 {
8   _id: 123456789,
9   title: "MongoDB: The Definitive Guide",
10  author: [ "Kristina Chodorow", "Mike Dirolf" ],
11  published_date: ISODate("2010-09-24"),
12  pages: 216,
13  language: "English",
14  publisher_id: "oreilly"
15 }
16 {
17   _id: 234567890,
18   title: "50 Tips and Tricks for MongoDB Developer",
19   author: "Kristina Chodorow",
20   published_date: ISODate("2011-05-06"),
21   pages: 68,
22   language: "English",
23   publisher_id: "oreilly"
24 }
```

Document Schema Design

```
1 {
2   name: "O'Reilly Media",
3   founded: 1980,
4   location: "CA",
5   books: [123456789, 234567890, ...]
6 }
7 {
8   _id: 123456789,
9   title: "MongoDB: The Definitive Guide",
10  author: [ "Kristina Chodorow", "Mike Dirolf" ],
11  published_date: ISODate("2010-09-24"),
12  pages: 216,
13  language: "English"
14 }
15 {
16   _id: 234567890,
17   title: "50 Tips and Tricks for MongoDB Developer",
18   author: "Kristina Chodorow",
19   published_date: ISODate("2011-05-06"),
20   pages: 68,
21   language: "English"
22 }
```

การออกแบบ Collection จากความสัมพันธ์แบบ one-to-many

embed
ref

bi direction

The screenshot shows a 'Document Schema Design' interface with two documents. The first document is for 'Publisher' and the second for 'Book'. Both documents have a field 'publisher_id' which is highlighted with a purple border, indicating it's a reference to the other collection.

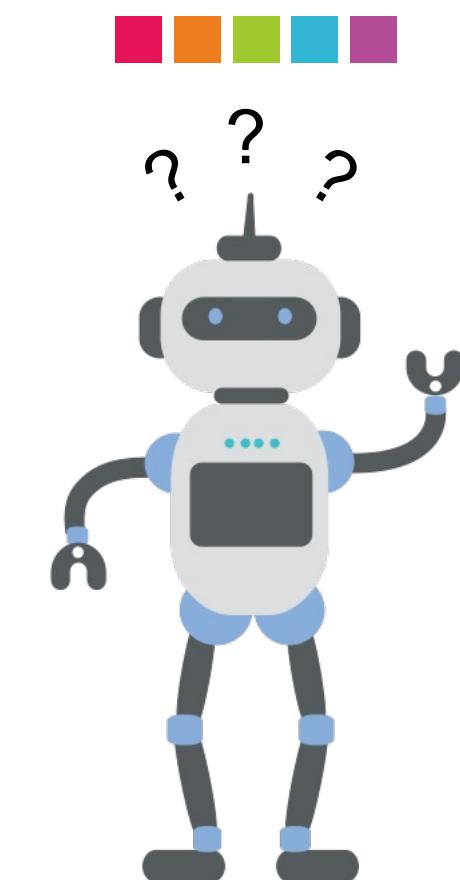
```
1 {
2   _id: "oreilly",
3   name: "O'Reilly Media",
4   founded: 1980,
5   location: "CA",
6   books: [123456789, 234567890]
7 }
8 {
9   _id: 123456789,
10  title: "MongoDB: The Definitive Guide",
11  author: [ "Kristina Chodorow", "Mike Dirolf" ],
12  published_date: ISODate("2010-09-24"),
13  pages: 216,
14  language: "English",
15  publisher_id: "oreilly"
16 }
17 {
18   _id: 234567890,
19   title: "50 Tips and Tricks for MongoDB Developer",
20   author: "Kristina Chodorow",
21   published_date: ISODate("2011-05-06"),
22   pages: 68,
23   language: "English",
24   publisher_id: "oreilly"
25 }
```

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,  
}
```



one direction ควรเก็บแบบซ้ายหรือขวา
- ต้องดูว่าตรงไหนจะบวม

ref

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  
}
```

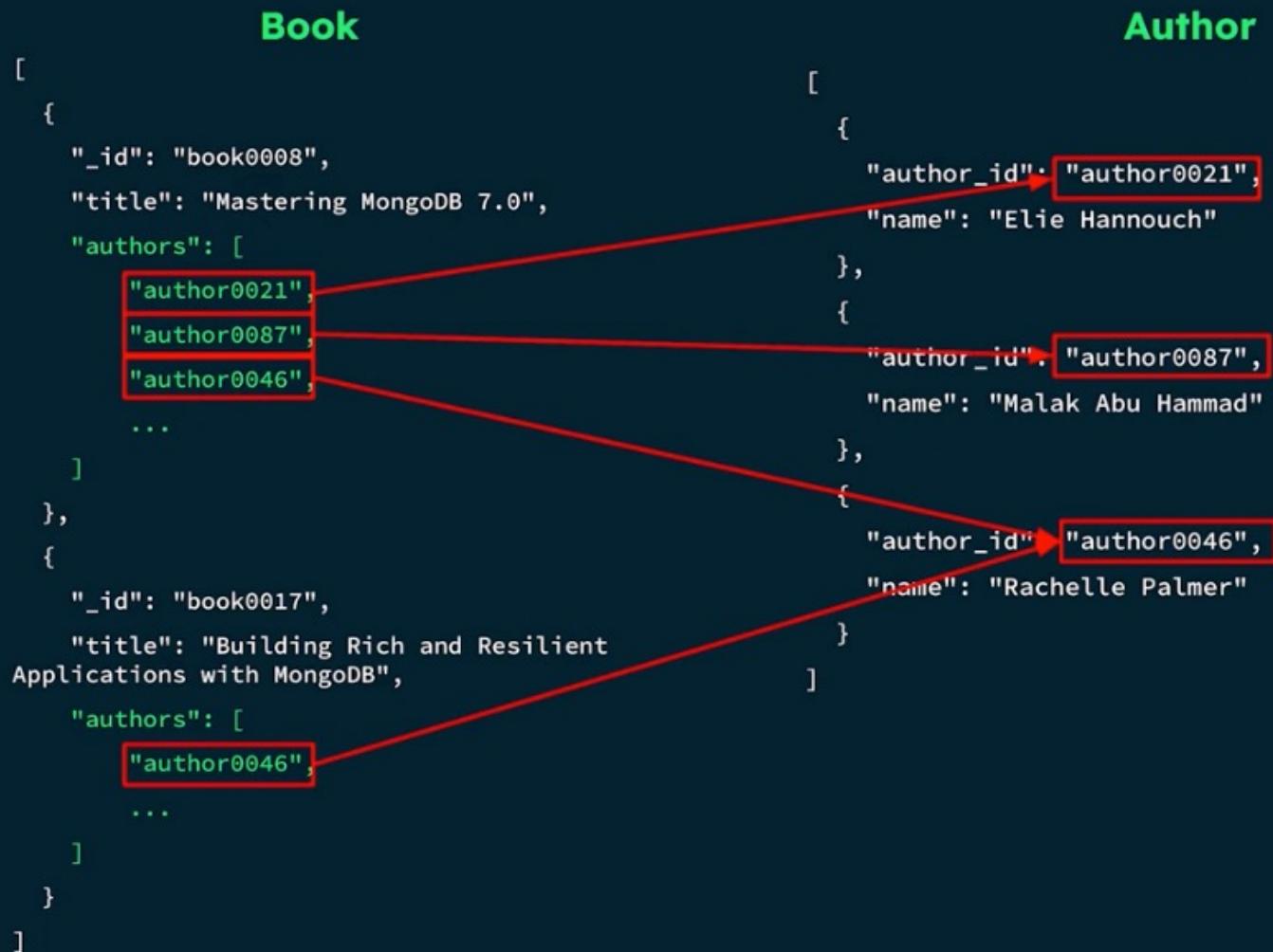
อย่างไรก็ตาม ในรูปแบบด้านซ้ายหมายความกับจำนวนหนังสือต่อผู้แต่งน้อย ขณะที่ในรูปแบบด้านขวาหมายความกับจำนวนหนังสือต่อผู้แต่งมาก เนื่องจากในรูปแบบด้านซ้าย **ขนาดของ array จะเพิ่มสูงมาก** ถ้าจำนวนหนังสือต่อผู้แต่งมาก

การออกแบบ Collection จากความสัมพันธ์แบบ many-to-many

```
{  
  "_id": "book0008",  
  "title": "Mastering MongoDB 7.0",  
  "authors": [  
    {  
      "author_id": "author0021",  
      "name": "Elie Hannouch"  
    },  
    {  
      "author_id": "author0087",  
      "name": "Malak Abu Hammad"  
    },  
    {  
      "author_id": "author0046",  
      "name": "Rachelle Palmer"  
    },  
    ...  
  ]  
}
```

สำหรับข้อมูลที่มีความสัมพันธ์แบบกลุ่มต่อกลุ่มนักศึกษาสามารถใช้งาน Embedding Document หรือ Document Reference ได้ทั้ง 2 วิธีเพื่อหลีกเลี่ยงไม่ให้ข้อมูลของซ้ำกัน (Repetition Problem)

การออกแบบ Collection จากความสัมพันธ์แบบ many-to-many

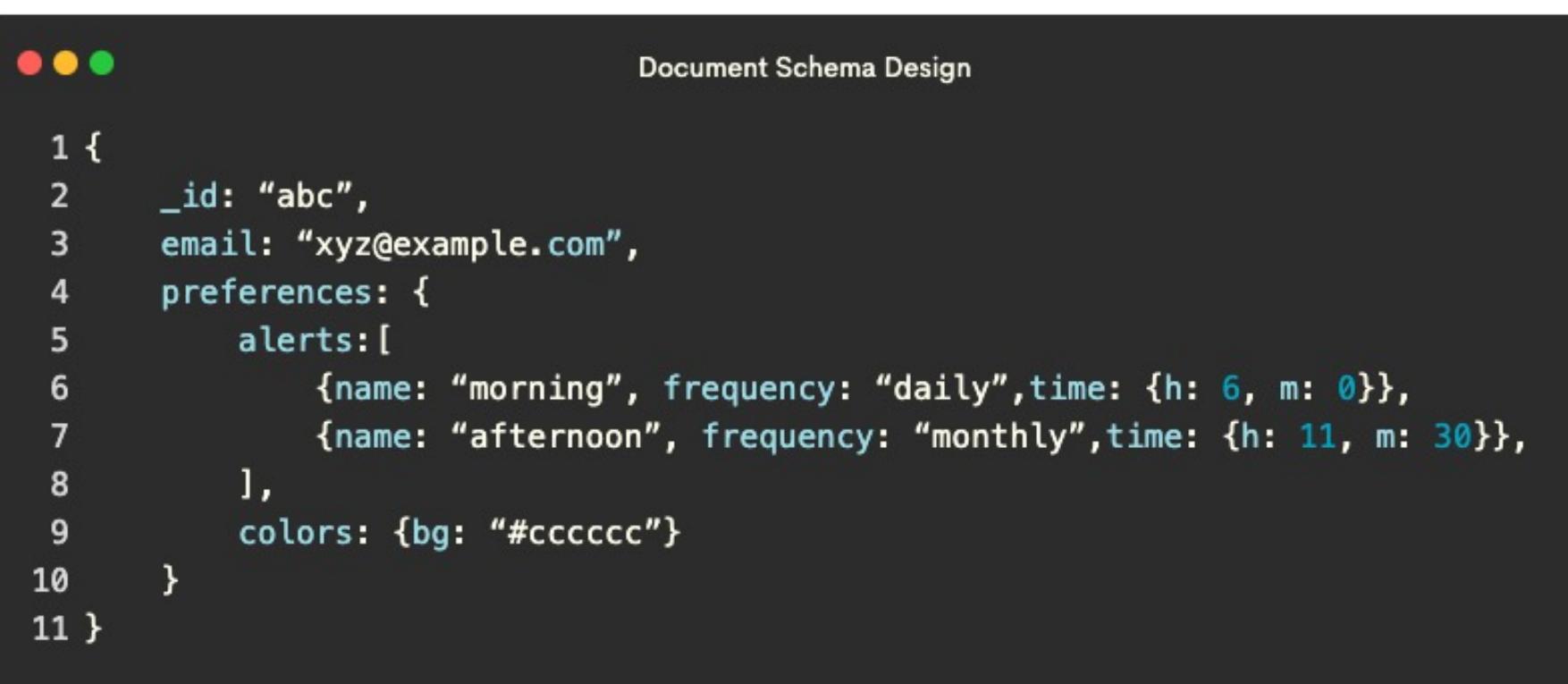


Document Schema Design Cheatsheet - Prefer Embedding

ข้อมูลที่เข้าถึงร่วมกันควรเก็บไว้ด้วยกัน

- ออกแบบและใช้โครงสร้างแบบ Object เพื่อจัดระเบียบข้อมูลภายใต้เอกสาร

จะใช้เป็น embed ได้ ก็เป็น embed ตีกั่ว



The screenshot shows a code editor with a dark theme. At the top left are three colored circular icons (red, yellow, green). The title bar says "Document Schema Design". The code is a JSON object:

```
1 {
2   _id: "abc",
3   email: "xyz@example.com",
4   preferences: {
5     alerts: [
6       {name: "morning", frequency: "daily", time: {h: 6, m: 0}},
7       {name: "afternoon", frequency: "monthly", time: {h: 11, m: 30}},
8     ],
9     colors: {bg: "#cccccc"}
10  }
11 }
```

เทคนิค Sub-document ช่วยให้นักศึกษาสามารถแยกส่วนต่างๆ ของฟิลด์ที่เกี่ยวข้องภายใต้ Document ได้อย่างชัดเจน และช่วยให้นักศึกษาสามารถแก้ไขหรือปรับปรุงส่วนย่อยของ Document ได้โดยไม่ต้องอาศัยการทำธุรกรรมหลาย Document

Document Schema Design Cheatsheet - Prefer Embedding

ข้อมูลที่เข้าถึงร่วมกันควรเก็บไว้ด้วยกัน

- ออกแบบและใช้โครงสร้างแบบ Array เพื่อร่วมข้อมูลที่เกี่ยวข้องกันหรือมีโอกาสเป็นได้หลายค่า (Multivalued)



Document Schema Design

```

1 {
2   _id: "def",
3   name: "Bake and Go",
4   addresses: [
5     {street: "40 Elm", state: "NY"},
6     {street: "101 Main St", state: "VT"}
7   ]
8 }

```

โครงสร้างข้อมูลแบบ Array ของ Sub-document ช่วยให้นักศึกษาสามารถรวมข้อมูลที่เกี่ยวข้องได้ อาทิเช่น ที่อยู่หรือบัญชี

การกำหนดจำนวนระดับ (degree) ของการซ่อนเอกสาร (embedding document) ที่เหมาะสมที่สุด ขึ้นกับลักษณะของการต้องการของระบบและการเติบโตของระบบ MongoDB ได้รับการออกแบบมาเพื่อยอมรับการเปลี่ยนแปลง

Document Schema Design Cheatsheet - Know when not to Embed

ของที่ใช้แยกกันควรเก็บแยกออกจากกัน

- ย้าย Embedding Document ที่มีการเข้าถึงบ่อยไปยัง Collection ของตนเอง

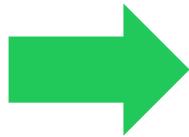
Document Schema Design

```

1 // db.manufacturer:
2 {
3   _id: "ghi",
4   name: "Swaab Automotive",
5   type: "auto",
6   models: [
7     {name: "X", year: 2018, sku: "AB-123Z"}
8   ]
9 }  

  ถ้า subdocument ถูกใช้บ่อยๆ ให้แยก document

```



Document Schema Design

```

1 // db.manufacturer:
2 {
3   _id: "ghi",
4   name: "Swaab Automotive",
5   type: "auto"
6 }
7 // db.model:
8 {
9   _id: "jkl",
10  name: "X",
11  year: 2018,
12  sku: "AB-123Z",
13  manufacturer_id: "ghi"
14 }

```

ถ้า Sub-document ของนักศึกษา (หรือ Array ของเอกสาร) เป็น Object ที่นักศึกษาใช้งานบ่อยจากภายนอก Document ให้พิจารณาถ่าย Sub-document ไปยัง Collection ของตนเอง

จากตัวอย่าง นักศึกษาสามารถเข้าถึงข้อมูล Model (โมเดล) ต่างๆ ได้อย่างอิสระจาก Manufacturer (ผู้ผลิต) สิ่งนี้มีประโยชน์อย่างมาก ถ้านักศึกษามีการ read หรือ write ตัว Document ของ Model ในปริมาณมาก

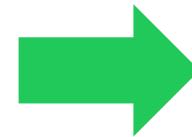
Document Schema Design Cheatsheet - Know when not to Embed

ของที่ใช้แยกกันควรเก็บแยกออกจากกัน

- อย่า Embedding รายการ (Object or Array) ที่เติบโตอย่างไรขอบเขต

```
● ● ● Document Schema Design
1 // db.user:
2 {
3   _id: "mno",
4   username: "frankysezhey",
5   login_times: [
6     {date: "1/1/2020", time: "00:14:09"},
7     {date: "2/10/2020", time: "10:14:09"}
8   ]
9 }
```

รายการใดๆ ที่ถูกเพิ่มเข้าไปอย่างต่อเนื่องไม่ควรถูก Embedding มันควรจะจัดเก็บใน Collect ของตัวเอง



```
● ● ● Document Schema Design
1 //db.user:
2 {
3   _id: "mno",
4   username: "frankysezhey",
5   login_times: [ "pqr", "abc"]
6 }
7 // db.login_audit:
8 {
9   _id: "pqr",
10  time: { date: "1/1/2020",time: "00:14:09" },
11  user_id: "mno"
12 }
13 {
14   _id: "abc",
15   time: { date: "5/2/2022",time: "10:23:19" },
16   user_id: "mno"
17 }
```

ในการนี้ การสร้าง Collect ใหม่เป็นตัวเลือกที่เหมาะสมกว่าเนื่องจากสำหรับรายการที่สามารถเติบโตได้ และนักศึกษามารถใช้ index เพื่อการสืบค้น Document ที่เกี่ยวข้องในอีก Collection ได้

Document Schema Design Cheatsheet - Embrace Duplication

สำหรับข้อมูลใช้คู่กันควรเก็บไว้ด้วยกันถึงแม้ว่าจะเก็บข้อมูลซ้ำช้อนกัน

- จัดเก็บข้อมูลที่เป็นประโยชน์ไว้ใน Document เดียวกัน

● ● ● Document Schema Design

```

1 //db.user:
2 {
3     _id: "stu",
4     email: "hello@example.com"
5 }
6 //db.post:
7 {
8     _id: "vwx",
9     text: "Hello World",
10    user_id: "stu",
11    user_email: "hello@example.com"
12 }
```

จัดการข้อมูลที่ซ้ำกัน

- ใช้ bidirection reference --> ทั้งสองฝ่ายจะต้องมี key ของอีกฝ่ายนึงเพื่อโยงกันว่าตรงไหนอัพเดทข้อมูลแล้ว อีกตรงก็อัพเดทข้อมูลตาม

หลีกเลี่ยงการใช้งานตัวดำเนินการ joins ที่ไม่จำเป็นโดยอาศัยการเก็บข้อมูลบางฟิลด์ซ้ำ โดยแยกกับความซับซ้อนที่เพิ่มเข้ามาสำหรับการจัดการข้อมูลให้ทันสมัยอยู่เสมอ ซึ่งเป็นหลักการออกแบบ “extended reference” design pattern

Document Schema Design Cheatsheet - Embrace Duplication

สำหรับข้อมูลใช้คู่กันควรเก็บไว้ด้วยกันถึงแม้ว่าจะเก็บข้อมูลซ้ำซ้อนกัน

- คำนึงถึงความสอดคล้องของข้อมูลใน Document ที่ต่างกัน

Document Schema Design

```

1 //db.user:
2 {
3     _id: "stu",
4     email: "helloOld@example.com"
5 }
6 //db.post:
7 {
8     _id: "vwx",
9     text: "Hello World",
10    user_id: "stu",
11    user_email: "helloNew@example.com"
12 }
```

ในการณ์ที่นักศึกษาไม่ต้องการให้ข้อมูลที่ซ้ำกันไม่ซิงค์กัน นักศึกษาสามารถอาศัยหลักการ Change Streams (เป็นกลไกที่ใช้จัดการปัญหานี้)

การจัดเก็บข้อมูลรายการเดียวกันไว้ใน Document หลายฉบับในฐานข้อมูลเป็นเทคนิคที่มีประสิทธิภาพ แต่อย่างไรก็ตาม นักศึกษาควรคำนึงถึงกฎความสอดคล้องของข้อมูลอยู่เสมอ

อ้างอิง <https://www.mongodb.com/docs/manual/changeStreams/>

Document Schema Design Cheatsheet - Don't be Scared to Relate

มีวิธีการเชื่อมโยงที่หลากหลาย

- ใช้ Array ของ ID หรือการอ้างอิงย้อนกลับสำหรับความสัมพันธ์แบบ one to many

```
● ● ● Document Schema Design

1 //db.user:
2 {
3     _id: "stu",
4     email: "hello@aol.com",
5     friend_ids: ["aaa", "bbb"]
6 }
7 //db.post:
8 {
9     _id: "vwx",
10    text: "Hello World",
11    user_id: "stu"
12 }
```

นักศึกษาสามารถดึง Document ที่เกี่ยวข้องผ่านการสืบค้นย่อยจาก Array ของ Key (เช่น db.user.friend_ids) หรือสืบค้นโดยเทียบกับ Key (เช่น db.post.user_id) ทั้งนี้ขึ้นอยู่กับรูปแบบการเข้าถึงของนักศึกษา การใช้งานตัวดำเนินการ Joins เมื่อจำเป็นเท่านั้น

Document Schema Design Cheatsheet - Don't be Scared to Relate

มีวิธีการเชื่อมโยงที่หลากหลาย

- การจัดเก็บความสัมพันธ์แบบสองทิศทาง เมื่อเปลี่ยนการอ้างอิงด้านใดด้านหนึ่งของความสัมพันธ์แบบสองทิศทาง นักศึกษาต้องไม่ลืมอัปเดตอีกด้านหนึ่งด้วยเสมอ

● ● ●

Document Schema Design

```

1 //db.user:
2 {
3     _id: "aaa",
4     email: "a@example.com",
5     friend_ids: ["ccc", "bbb"]
6 }
7 {
8     _id: "bbb",
9     email: "b@example.com",
10    friend_ids: ["eee", "aaa"]
11 }
```

การจัดเก็บความสัมพันธ์แบบสองทิศทางเป็นอีกหนึ่งกรณีการใช้งานที่ดีสำหรับ Change Streams

MongoDB ไม่ใช่ฐานข้อมูล "เชิงสัมพันธ์" แบบดั้งเดิม ไม่ได้หมายความว่าจะไม่สามารถสร้างความสัมพันธ์ได้

ขั้นตอนที่ 2.2: Embedding or Referencing

Guideline Name	Question	Embed	Reference
Simplicity	Would keeping the pieces of information together lead to a simpler data model and code?	Yes	No
Go Together	Do the pieces of information have a "has-a," "contains," or similar relationship?	Yes	No
Query Atomicity	Does the application query the pieces of information together?	Yes	No
Update Complexity	Are the pieces of information updated together?	Yes	No
Archival	Should the pieces of information be archived at the same time?	Yes	No
Cardinality	Is there a high cardinality (current or growing) in the child side of the relationship?	No	Yes
Data Duplication	Would data duplication be too complicated to manage and undesired?	No	Yes
Document Size	Would the combined size of the pieces of information take too much memory or transfer bandwidth for the application?	No	Yes
Document Growth	Would the embedded piece grow without bound?	No	Yes
Workload	Are the pieces of information written at different times in a write-heavy workload?	No	Yes
Individuality	For the children side of the relationship, can the pieces exist by themselves without a parent?	No	Yes

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns

ขั้นตอนที่ 3: Schema Design Patterns

นักศึกษาจะได้เรียนรู้ Schema Design Patterns ดังต่อไปนี้

- Inheritance Pattern
- Computed Pattern
- Approximation Pattern
- Extended Reference Pattern
- Schema Versioning Pattern

ขั้นตอนที่ 3: Schema Design Patterns

นักศึกษาจะได้เรียนรู้ Schema Design Patterns ดังต่อไปนี้

- Inheritance Pattern
- Computed Pattern
- Approximation Pattern
- Extended Reference Pattern
- Schema Versioning Pattern

Inheritance Design Patterns

```
{
  "sport": "ten_pin_bowling",
  "athlete_name": "Earl Anthony",
  "career_earnings": {value: NumberDecimal("1441061"), currency: "USD"},
  "300_games": 25,
  "career_titles": 43,
  "other_sports": "baseball"
}

{
  "sport": "tennis",
  "athlete_name": "Martina Navratilova",
  "career_earnings": {value: NumberDecimal("216226089"), currency: "USD"},
  "event": {
    "type": "singles",
    "career_tournaments": 390,
    "career_titles": 167
  }
}
```

Common fields

```
{
  "sport": "tennis",
  "athlete_name": "Martina Navratilova",
  "career_earnings": {value: NumberDecimal("216226089"), currency: "USD"},
  "career_tournaments": 390,
  "career_titles": 167,
  "event": [
    {
      "type": "singles",
      "career_tournaments": 390,
      "career_titles": 167
    },
    {
      "type": "doubles",
      "career_tournaments": 233,
      "career_titles": 177,
      "partner": ["Tomanova", "Fernandez", "Morozova", "Evert", ...]
    }
  ],
  ...
}
```

Polymorphic Sub-Documents

Inheritance Design Patterns นิยมใช้เมื่อ Document มีความคล้ายคลึงมากกว่าที่มีความแตกต่าง ซึ่ง Inheritance Design Patterns เป็นการออกแบบที่ใช้งานง่ายที่ช่วยให้สามารถสืบค้นใน Collection เดียวได้ และเป็นจุดเริ่มต้น สำหรับรูปแบบการออกแบบหลายๆ รูปแบบ

Computed Design Patterns

Screening Information

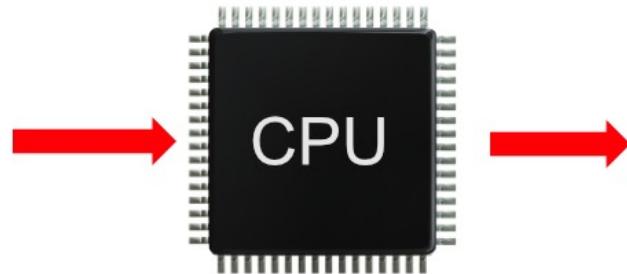
```
{
  "ts": DateTime(XXX),
  "theater": "Alger Cinema",
  "location": "Lakeview, OR",
  "movie_title": "Jack Ryan: Shadow Recruit",
  "num_viewers": 344,
  "revenue": 3440
}
```

```
{
  "ts": DateTime(XXX),
  "theater": "City Cinema",
  "location": "New York, NY",
  "movie_title": "Jack Ryan: Shadow Recruit",
  "num_viewers": 1496,
  "revenue": 22440
}
```

```
{
  "ts": DateTime(XXX),
  "theater": "Overland Park Cinema",
  "location": "Boise, ID",
  "movie_title": "Jack Ryan: Shadow Recruit",
  "num_viewers": 760,
  "revenue": 7600
}
```

Movie Information

```
{
  "ts": DateTime(XXX),
  "title": "Jack Ryan: Shadow Recruit",
  "viewers": 2600,
  "revenue": 33480
}
```



Computed Design Patterns
 นิยมใช้เมื่อมีนักศึกษามีข้อมูลที่
 จำเป็นต้องคำนวณซ้ำ ๆ ในระบบ
 และยังใช้เมื่อมีการอ่านและเข้าถึง
 ข้อมูลในปริมาณมาก ตัวอย่างเช่น
 หากนักศึกษามีการอ่าน
 1,000,000 ครั้งต่อชั่วโมง แต่มี
 การเขียนเพียง 1,000 ครั้งต่อ
 ชั่วโมง การคำนวณในขณะที่เขียน
 จะหารจำนวนการคำนวณด้วย
 1,000 เท่า

Computed Design Patterns

ตัวอย่างฐานข้อมูลภาพยนตร์ (Movie) สามารถคำนวณค่าต่าง ๆ ตามข้อมูลการฉาย (Screening) ทั้งหมดที่เกี่ยวกับภาพยนตร์เรื่องที่ต้องการได้ จากนั้น จึงจัดเก็บหรือปรับปรุงข้อมูลไว้กับข้อมูลภาพยนตร์ได้ ซึ่ง Computed Design Patterns หมายความว่า สถาปัตยกรรมที่มีการเขียนน้อย การคำนวณสามารถทำได้ร่วมกับการอัปเดตข้อมูลต้นฉบับ ในกรณีที่มีการเขียนเป็นประจำ การคำนวณสามารถทำได้ตามช่วงเวลาที่กำหนด เช่น ทุกชั่วโมง

Approximation Design Patterns

นักศึกษาสามารถใช้ Approximation Design Patterns ได้เมื่อ (1) ต้องการลดการแสดงผลที่ได้รับจากการคำนวณ หรือ(2) ทรัพยากรที่ใช้สูงเกิน (เวลา หน่วยความจำ รอบ CPU) ในการคำนวณ และ เมื่อความแม่นยำไม่สำคัญ
ความสำคัญสูงสุด

ตัวอย่างเช่น คำถามเกี่ยวกับการนับจำนวนประชากร อาจจะพบเจอคำถามต่อไปนี้

- ตัวจะคำนวณตัวเลขให้แม่นยำจะต้องใช้ทรัพยากรเท่าไร
- อาจจะมีการเปลี่ยนแปลงตั้งแต่เริ่มการคำนวณหรือไม่
- กลยุทธ์การวางแผนของเมืองจะมีผลกระทบอย่างไรหาก รายงานเป็น 39,000 แต่ในความเป็นจริงเป็น 39,012

แนวคิดการ Data Modeling สำหรับ MongoDB

ประกอบไปด้วย 3 ขั้นตอน ได้แก่

- ขั้นตอนที่ 1: Identifying Database Workloads
 - ขั้นตอนที่ 1.1: Identifying and Quantifying Entities
 - ขั้นตอนที่ 1.2: Identifying Reads and Writes
 - ขั้นตอนที่ 1.3: Quantifying Reads and Writes
- ขั้นตอนที่ 2: Modeling Data Relationships
 - ขั้นตอนที่ 2.1: Identifying Relationships
 - ขั้นตอนที่ 2.2: Embedding or Referencing
- ขั้นตอนที่ 3: Schema Design Patterns