#CodeMash

Schema Design

Sridhar Nanjundeswaran

Engineer, 10gen,

@snanjund

10gen | the MongoDB company

mongoDB

Agenda

- Working with documents
- Schema design by example
- Other common Patterns

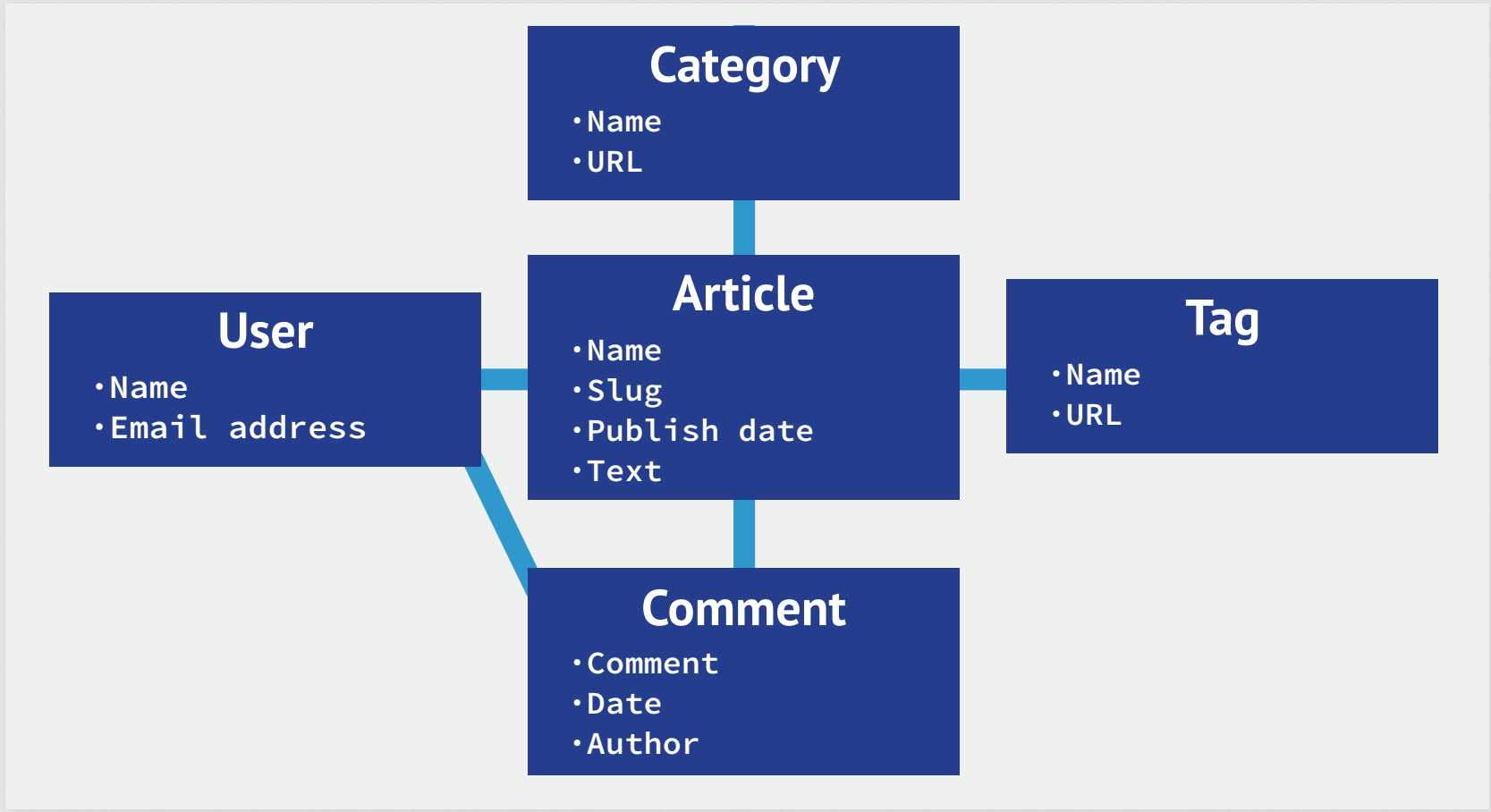
RDBMS		MongoDB
Database	➜	Database
Table	➜	Collection
Row	➜	Document
Index	➜	Index
Join	➜	Embedding & Linking

Terminology

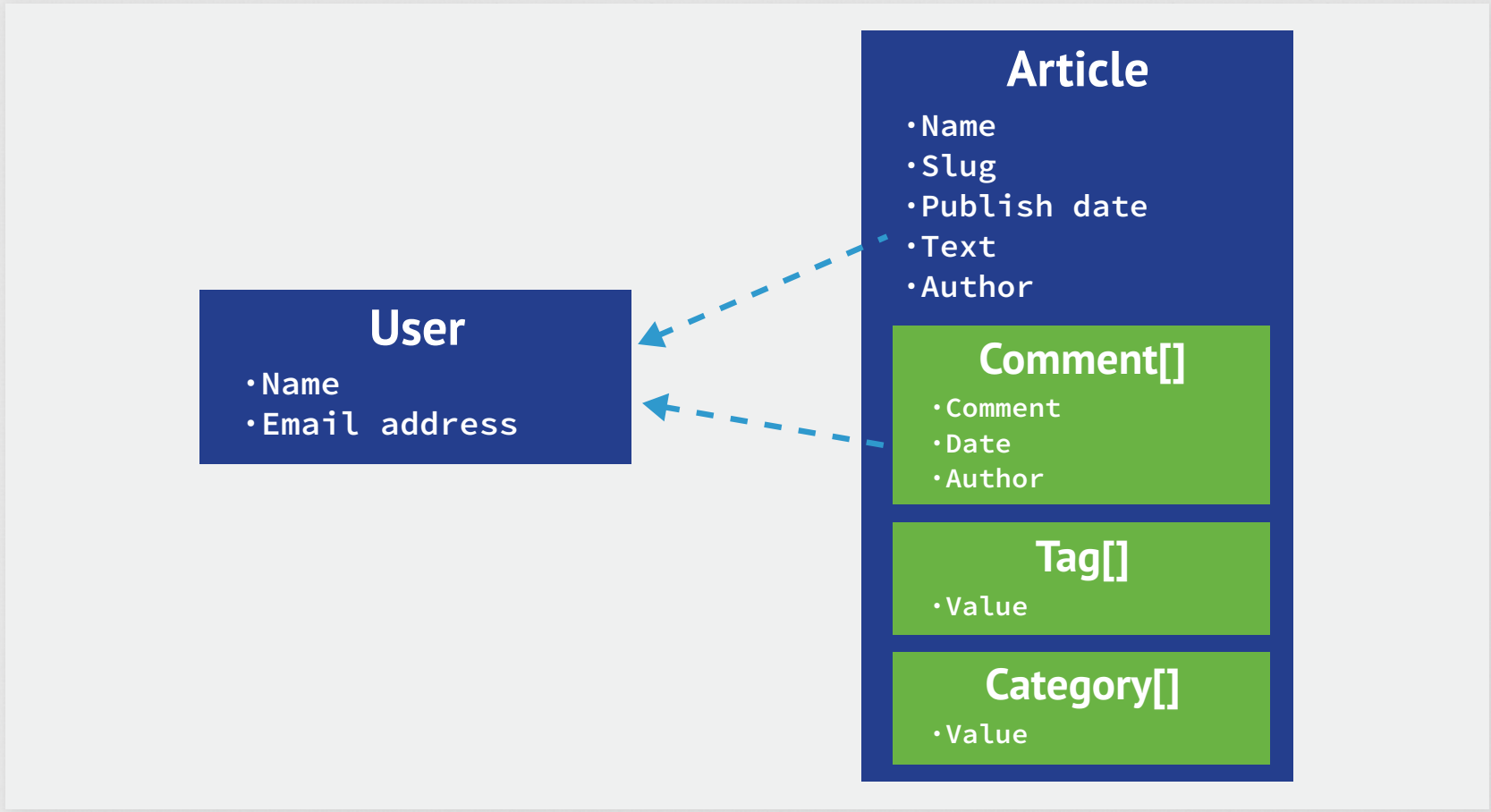
Working with Documents

Documents

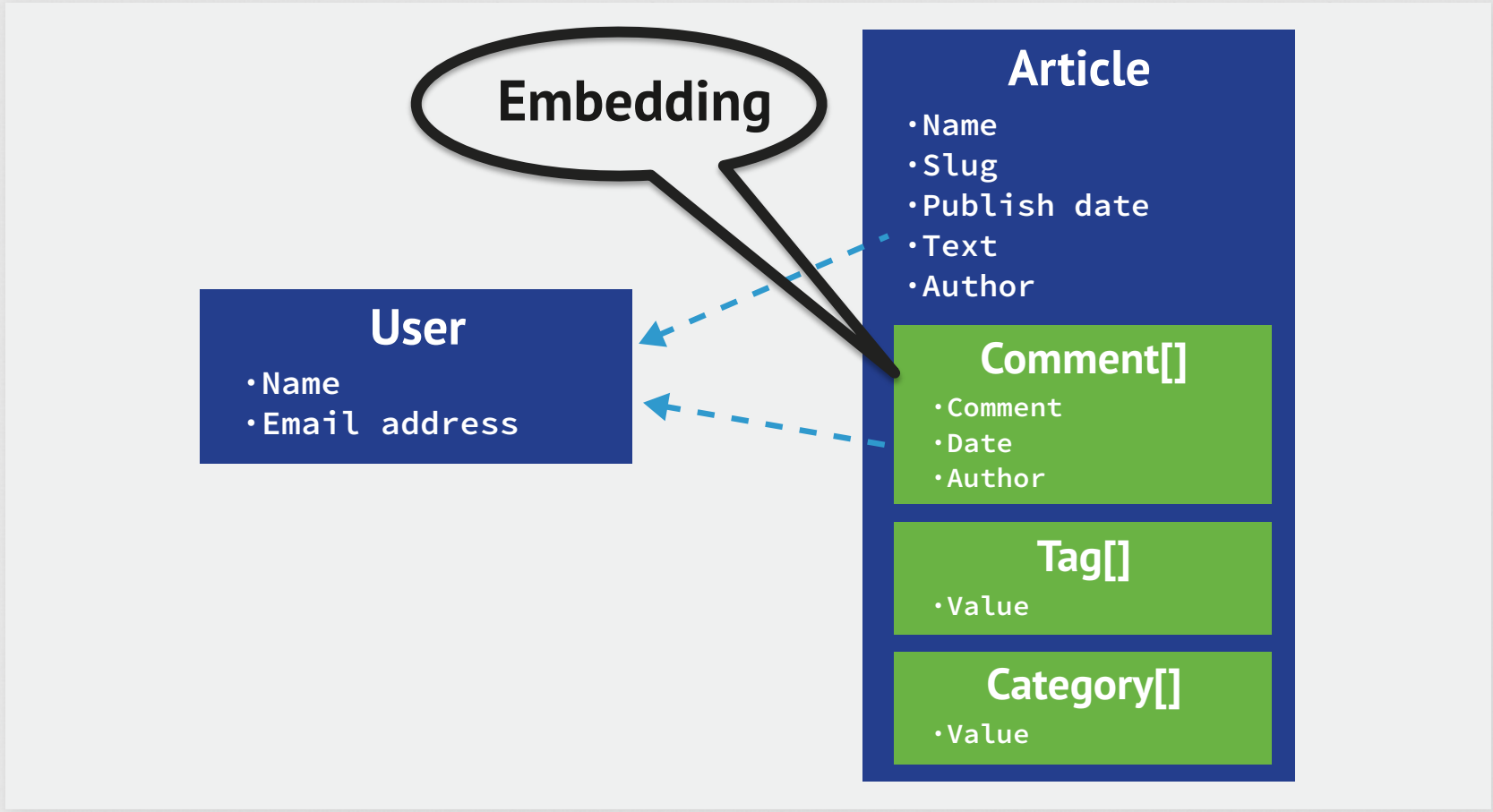
Provide flexibility and performance



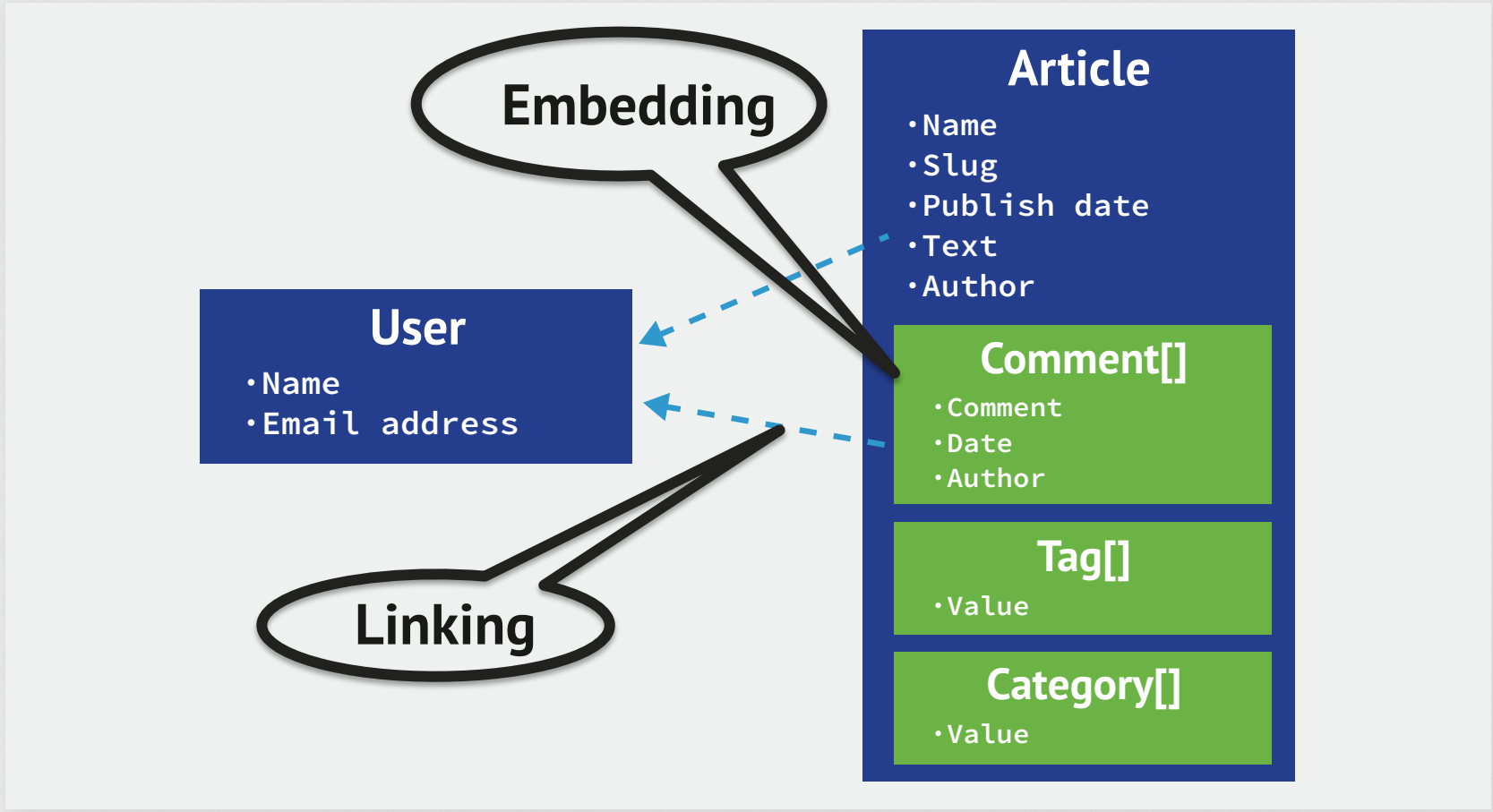
Normalized Schema (RDBMS)



De-Normalized Schema (MongoDB)



De-Normalized Schema (MongoDB)



De-Normalized Schema (MongoDB)

Relational Schema Design

Focuses on data storage

Document Schema Design

Focus on data use

Schema Design Considerations

- How do we manipulate the data?
  - Dynamic Ad-Hoc Queries
  - Atomic Updates
  - Map Reduce
- What are the access patterns of the application?
  - Read/Write Ratio
  - Types of Queries / Updates
  - Data life-cycle and growth rate

Data Manipulation

- Conditional Query Operators
  - **Scalar**: $ne, $mod, $exists, $type, $lt, $lte, $gt, $gte, $ne
  - **Vector**: $in, $nin, $all, $size
- Atomic Update Operators
  - **Scalar**: $inc, $set, $unset
  - **Vector**: $push, $pop, $pull, $pushAll, $pullAll, $addToSet

Data Access

- Flexible Schemas
- Ability to embed complex data structures
- Secondary Indexes
- Multi-Key Indexes
- Aggregation Framework
  - $project, $match, $limit, $skip, $sort, $group, $unwind
- No Joins

Schema Design by Example

Library Management Application

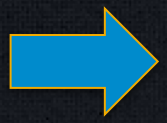
- Patrons
- Books
- Authors
- Publishers

An Example
One to One Relations

Modeling Patrons

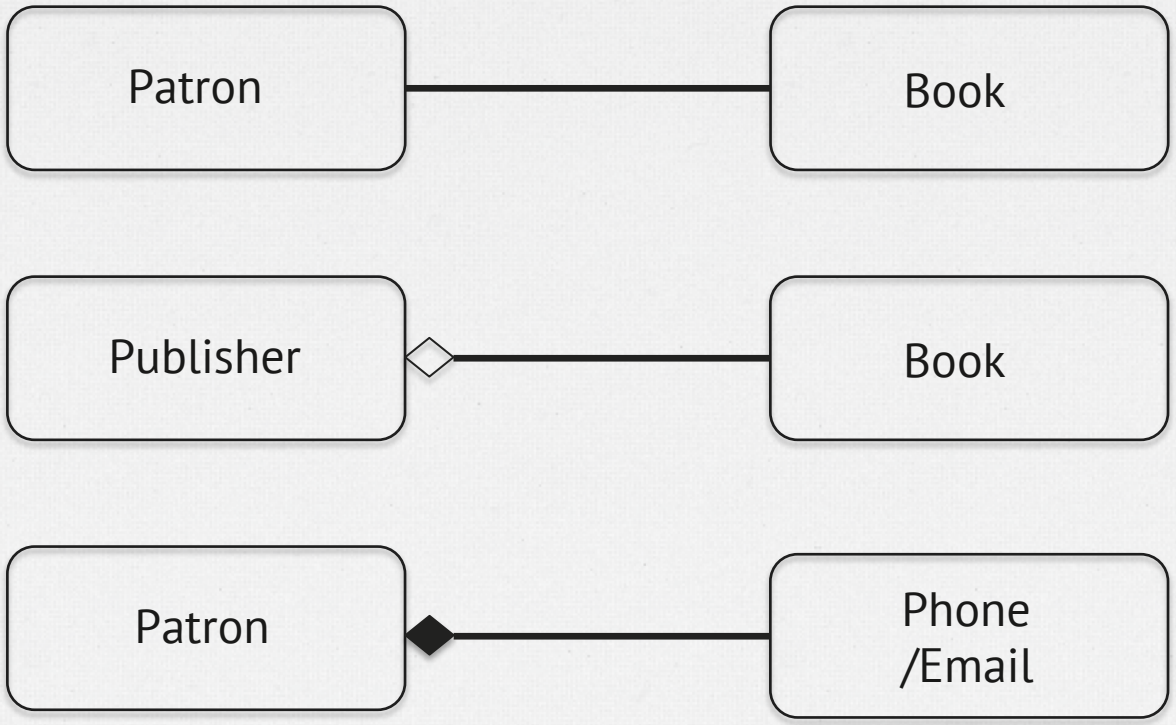
```
patron = {
    _id: "joe"
    name: "Joe Bookreader”
}

address = {
    patron_id = "joe",
    street: "123 Fake St. ",
    city: "Faketon",
    state: "MA",
    zip: 12345
}
```

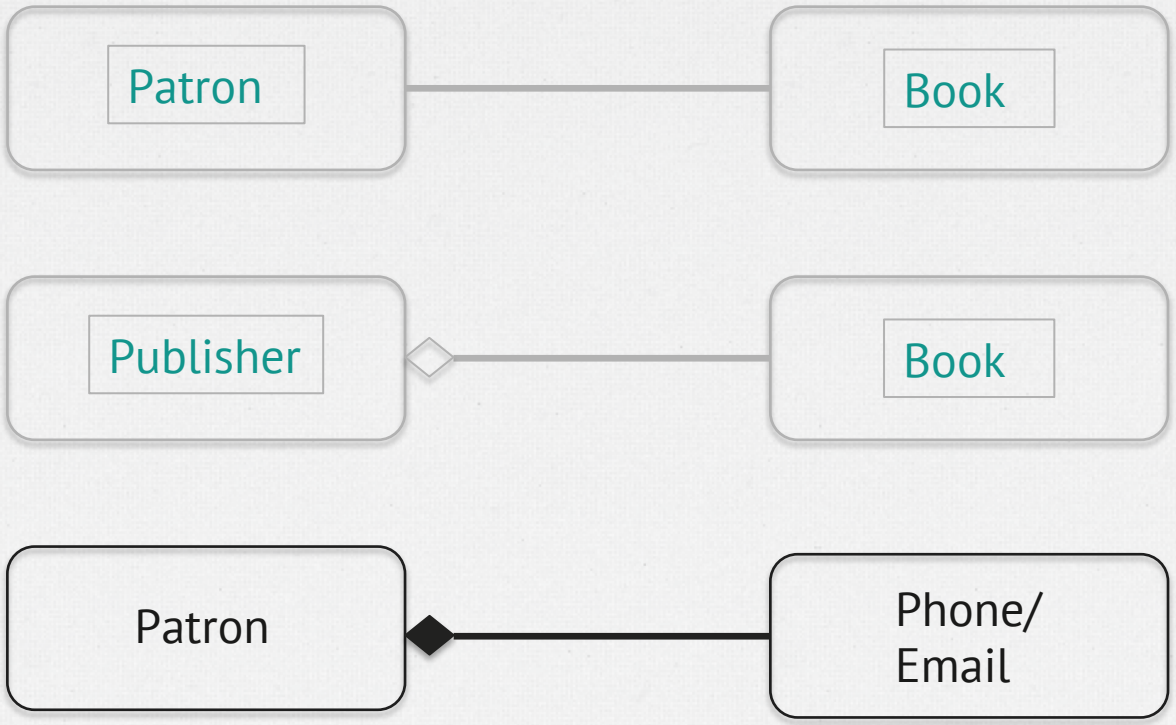


```
patron = {
    _id: "joe"
    name: "Joe Bookreader",
    address: {
        street: "123 Fake St. ",
        city: "Faketon",
        state: "MA",
        zip: 12345
    }
}
```

An Example
One To Many Relations



One to Many Relations



Strong ownership/lifecycle

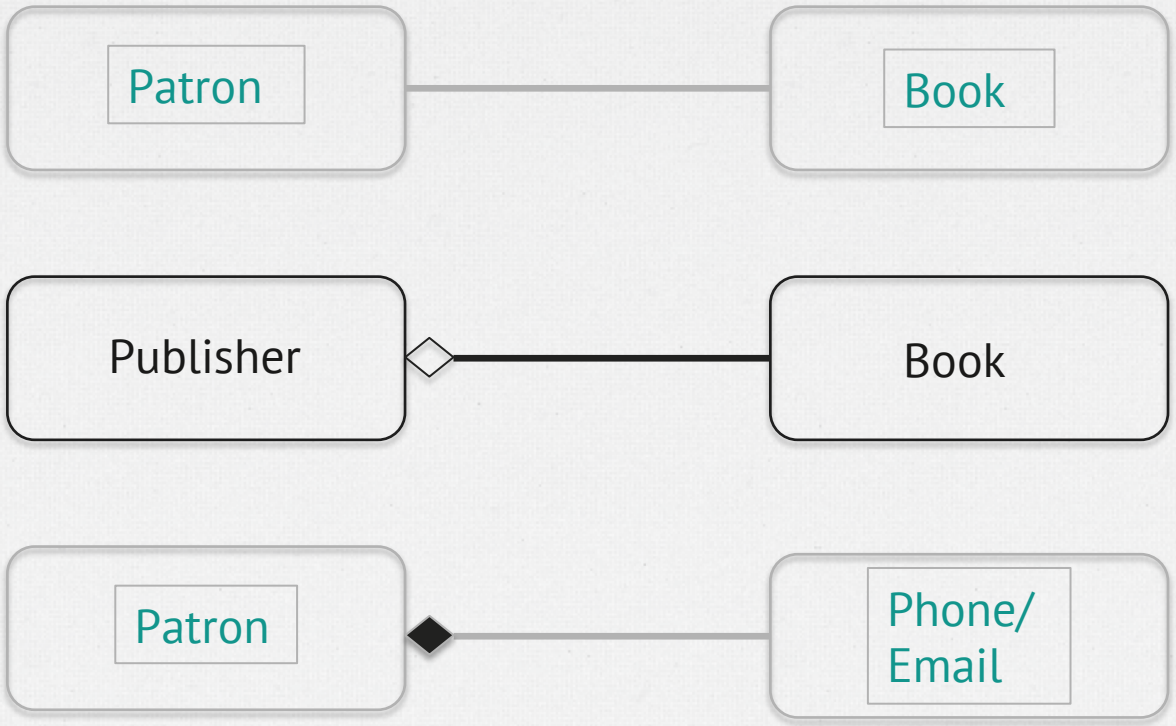
Modeling Patrons

```
patron = {
  _id: "joe"
  name: "Joe Bookreader",
  join_date: ISODate("2011-10-15"),
  addresses: {street: "123 Fake St.", city: "Faketon", state: "MA", ...},

  email: ["email@domain1.com","email@domain2.com"],

  phone:[
    {phone: "888-111-2222", type: "home"},
    {phone: "888-111-3333", type: "cell"}
  ]

}
```



Containment

Modeling Books – Embedded Publisher

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",

    publisher: {
        name: "O'Reilly Media",
        founded: "1980",
        location: "CA"
    }

}
```

Modeling Books – Embedded Publisher

What is hard?

- **All publishers in the system?**
  - Distinct query
- **Publisher moved location**
  - Need to update every book published

Publisher is its own entity

```
publisher = {
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
}

book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}
```

Publisher _id as a reference

```
publisher = {
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
}

book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}
```

Publisher _id as a reference

Give me books published by publisher_id.

One query

```
> db.book.find({publisher_id:"oreilly"})
```

Give me name of the publisher of "MongoDB: The Definitive Guide"

Two queries

```
> publisher_id  = db.book.find({title:"MongoDB: ...."})
> db.publisher.find({_id:publisher})
```

Give me all books published by publishers in California

Two queries

```
> publisher_ids = db.publisher.find({location: "CA"}, {_id:1})
> db.book.find({publisher_id:{$in:publisher_ids}})
```

Book _id as a reference (embedded)

```
publisher = {
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
    books: [ "123456789", ... ]
}

book = {
    _id: "123456789",
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}
```

Book _id as a reference (embedded)

Give me books published by publisher_id.

Two queries

```
> book_ids = db.publisher.find({_id:"oreilly"},{_id:0,books:1})
> db.book.find({_id:{$in:book_ids}})
```

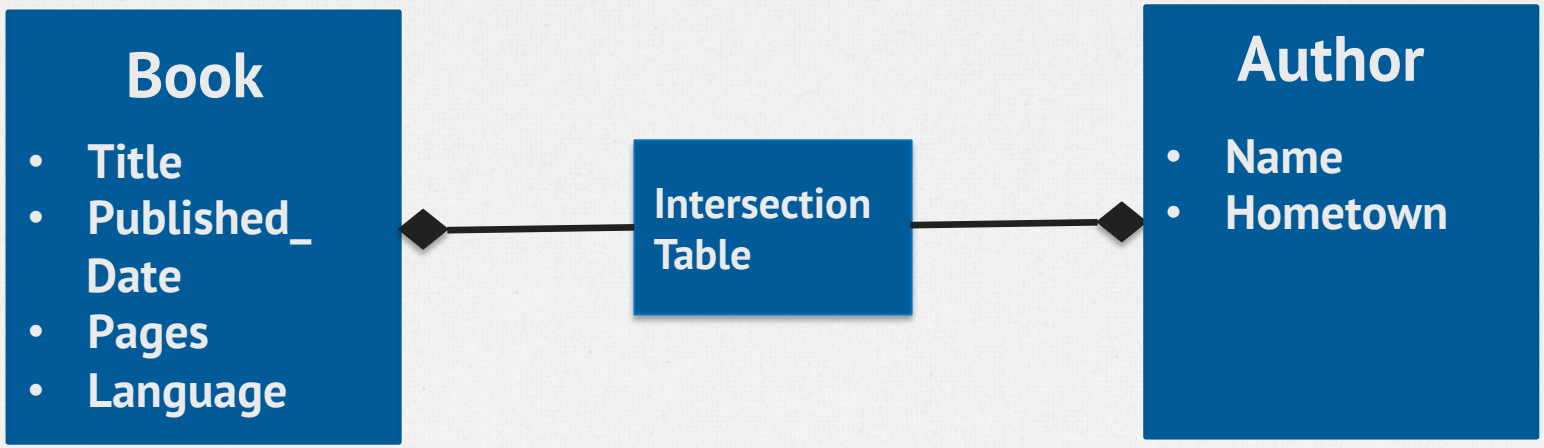
Relation stored on both sides

```
publisher = {
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: "1980",
    location: "CA"
    books: [ "123456789", ... ]
}

book = {
    _id: "123456789",
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}
}
```

An Example

Many to Many Relations



Relational Approach

Relation stored on book end

```
book = {
   title: "MongoDB: The Definitive Guide",
   authors = [
      { _id: "kchodorow", name: "Kristina Chodorow” },
      { _id: "mdirolf", name: "Mike Dirolf” }
   ]
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English"
}

author = {
   _id: "kchodorow",
   name: "Kristina Chodorow",
   hometown: "New York"
}
```

Relation stored on author end

```
book = {
    _id: 123456789
    title: "MongoDB: The Definitive Guide",
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

author = {
    _id: "kchodorow",
    name: "Kristina Chodorow",
    hometown: "Cincinnati",
    books: [ {book_id: 123456789, title : "MongoDB: The Definitive Guide" ]
}
```

Relation stored on both sides

```
book = {
    _id: 123456789
    title: "MongoDB: The Definitive Guide",
    authors = [ "kchodorow", "mdirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

author = {
    _id: "kchodorow",
    name: "Kristina Chodorow",
    hometown: "Cincinnati",
    books: [ 123456789, ... ]
}
```

Referencing vs. Embedding

- Embedding
  - Great for read performance
    - One seek to load entire object
    - One roundtrip to database
  - Writes can be slow
  - Maintaining data integrity
- Reference
  - More flexibility
  - Data integrity is maintained
  - Work is done during reads

An Example
Trees

Parent Links

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    category: "MongoDB"
}

category = { _id: MongoDB, parent: Databases }
category = { _id: Databases, parent: Programming }
```

Easy to query by parent category
Hard to find in subcategories

Array of Ancestors

```
book = {
   title: "MongoDB: The Definitive Guide",
   authors: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   parent: "MongoDB",
   categories: ["MongoDB", "Databases", "Programming" ]
}

book = {
   title: "MySQL: The Definitive Guide",
   authors: [ ”Michael Kofler" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   parent: "MySQL",
   ancestors: ["MySQL", "Databases", "Programming" ]
}
```

Ancestors as path

```
book = {
   title: "MongoDB: The Definitive Guide",
   authors: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   categories: "MongoDB/Databases/Programming"
}

book = {
   title: "MySQL: The Definitive Guide",
   authors: [ ”Michael Kofler" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   parent: "MySQL",
   ancestors: "MySQL/Databases/Programming"
}
```

Other common patterns

Inheritance hierarchy

- RDBMS
  - Multiple Tables vs Single Table

Shapes table

id	type	Area	radius	length	Width
1	circle	3.14	1		
2	square	4		2	
3	rect	10		5	2

- Sparse data
- Is missing value not required or an error?

Single table inheritance - RDBMS

Single collection (table) inheritance - MongoDB

```
> db.shapes.find()

{  _id : 1, type: "circle", area : 3.14, radius : 1 }
{  _id : 2, type: "square", area : 4, length : 2 }
{  _id : 3, type: "rect", area : 10, length : 5, width : 2 }
```

Summary

- Schema design is different in MongoDB
- Basic data design principals stay the same
- Focus on how application accesses/manipulates data
- Rapidly evolve schema to meet your requirements
- Enjoy your new freedom, use it wisely ☺

#CodeMash

Questions?

Sridhar Nanjundeswaran

Engineer, 10gen

@snanjund

10gen | the MongoDB company

mongoDB