# Schema Design

mongoDB

# Agenda

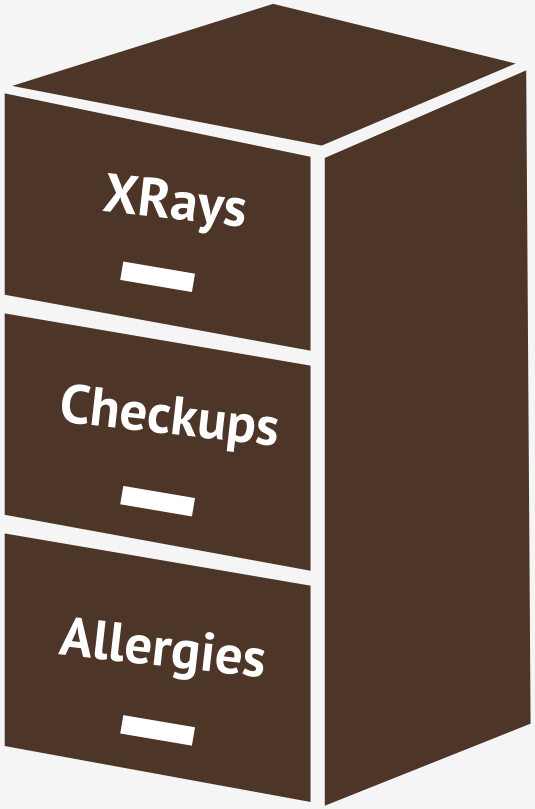
- Working with documents
- Evolving a Schema
- Queries and Indexes
- Common Patterns

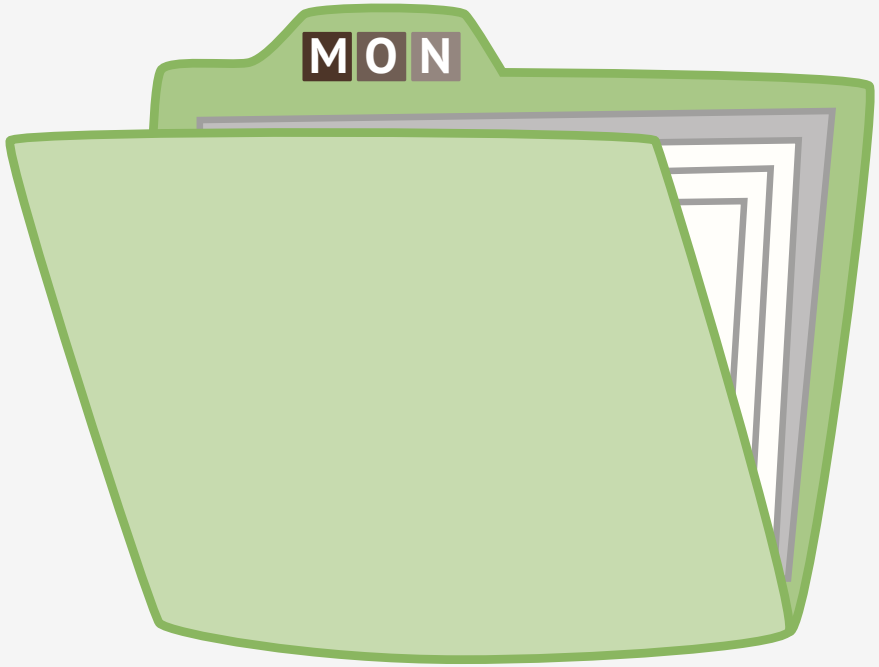
| RDBMS | | MongoDB |
|---|---|---|
| Database | ➜ | Database |
| Table | ➜ | Collection |
| Row | ➜ | Document |
| Index | ➜ | Index |
| Join | ➜ | Embedded Document |
| Foreign Key | ➜ | Reference |

# Working with Documents

mongoDB

# Modeling Data

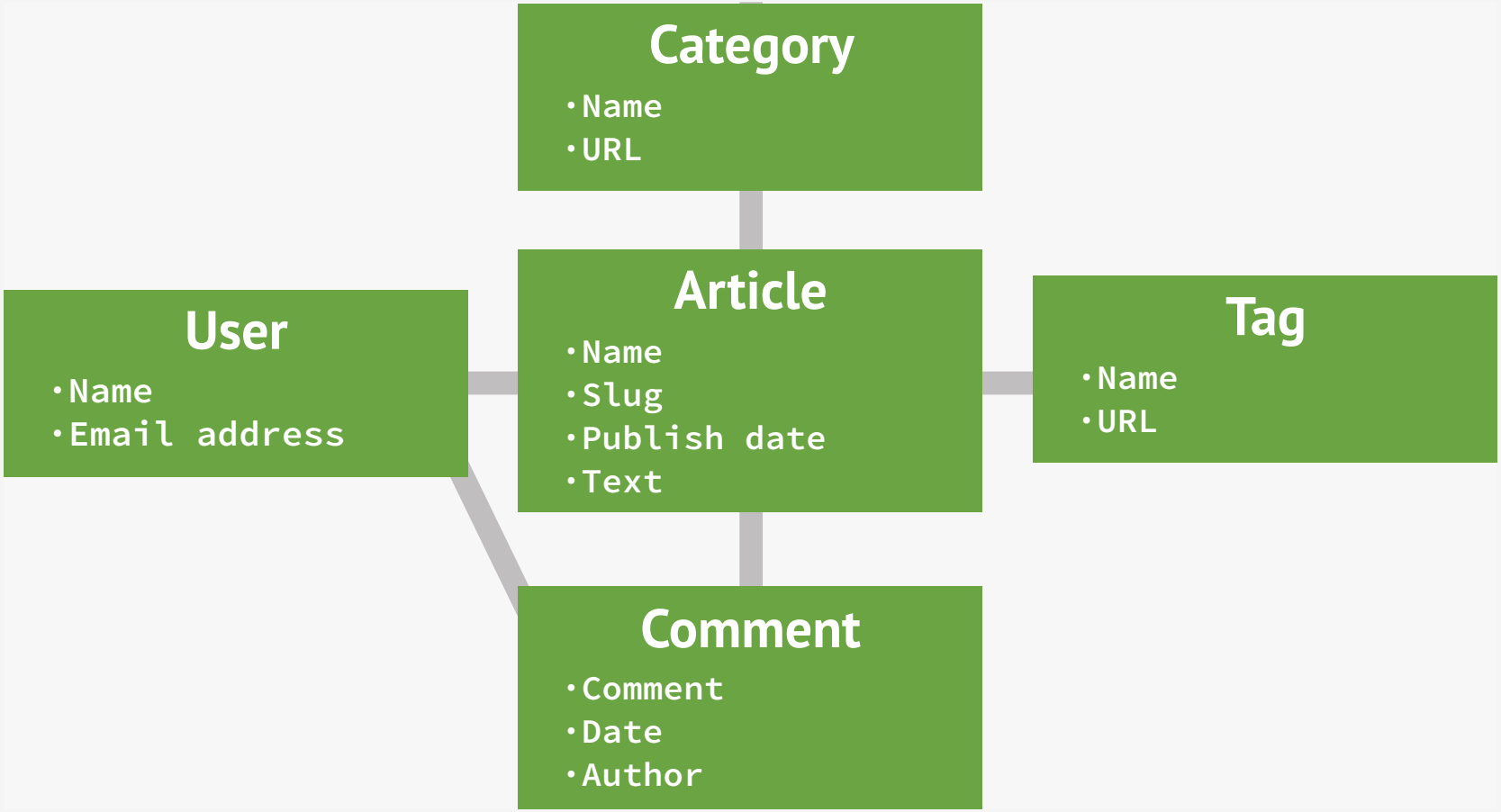


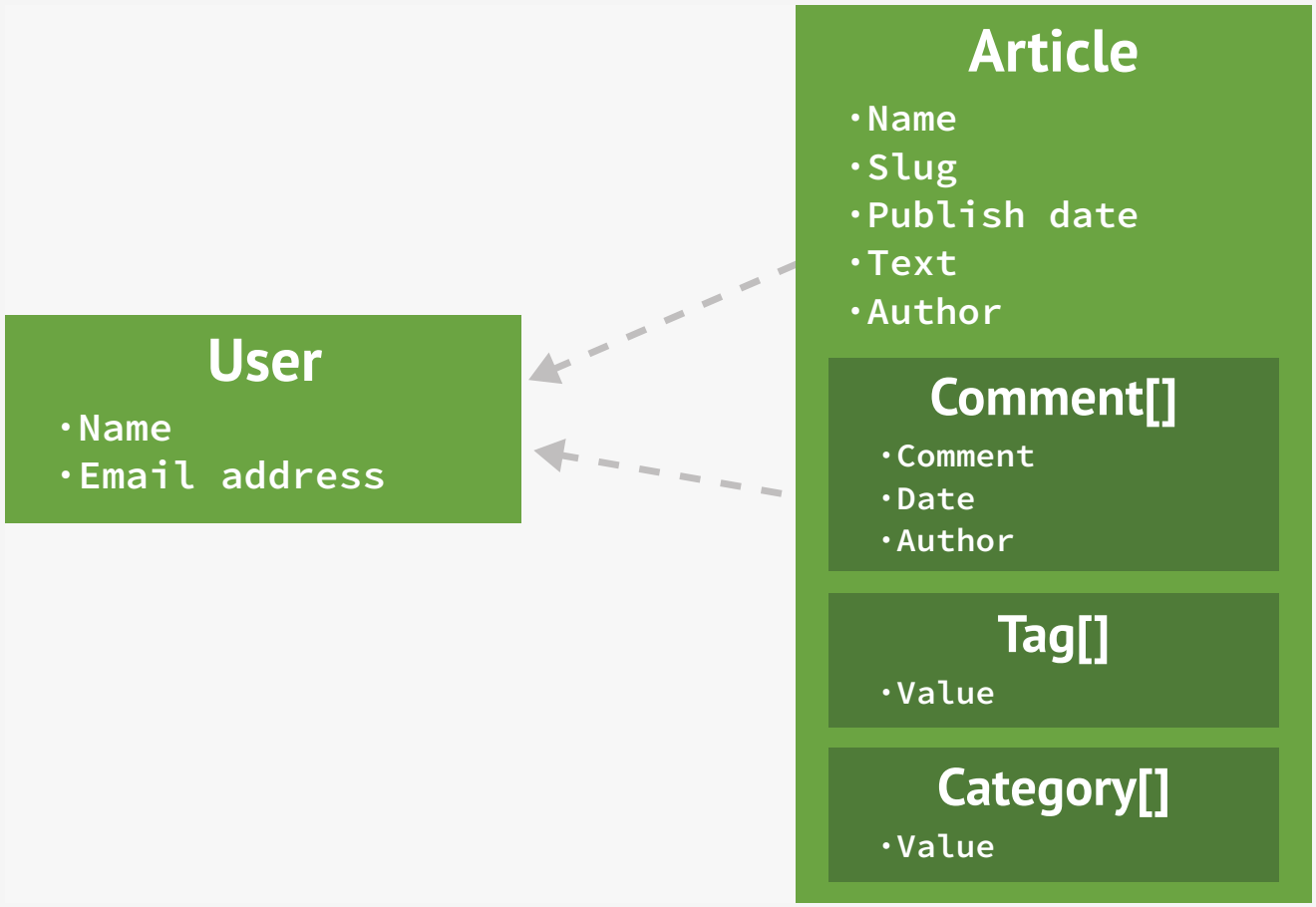
Documents

# Provide flexibility and performance

# Normalized Data



# De-Normalized (embedded) Data



# Relational Schema Design

## Focus 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

Query Selectors

- **Scalar**: $ne, $mod, $exists, $type, $lt, $lte, $gt, $gte
- **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

# Getting Started

mongoDB

# Library Management Application

Patrons

Books

Authors

Publishers

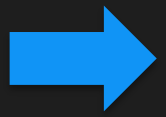
An Example

# 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
    }
}
```

# One to One Relations

Mostly the same as the relational approach

Generally good idea to embed “contains” relationships

Document model provides a holistic representation of objects

An Example

# One To Many Relations

# Modeling Patrons

```
patron = {
  _id: "joe",
  name: "Joe Bookreader",
  join_date: ISODate("2011-10-15"),
  addresses: [
    {street: "1 Vernon St.", city: "Newton", state: "MA", ...},
    {street: "52 Main St.", city: "Boston", state: "MA", ...}
  ]
}
```

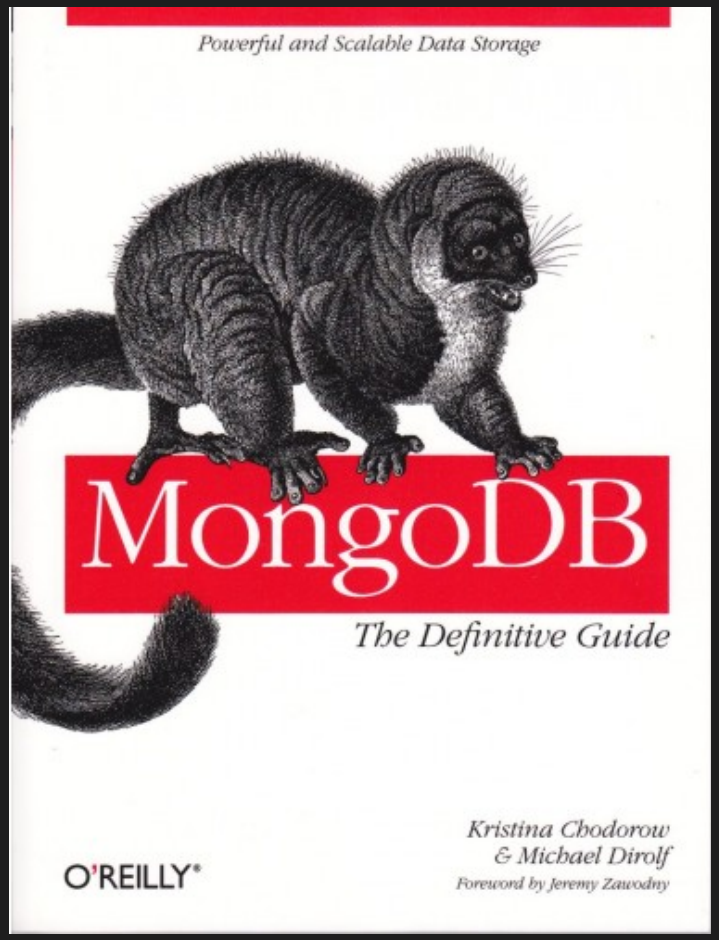
# Publishers and Books

- Publishers put out many books
- Books have one publisher

# Book

**MongoDB: The Definitive Guide,**
**By Kristina Chodorow and Mike Dirolf**
**Published: 9/24/2010**
**Pages: 216**
**Language: English**

**Publisher: O'Reilly Media, CA**



# 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 & Publisher Relationship

```
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 Foreign Key

```
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"
}
```

# Book _id as a Foreign Key

```
publisher = {
    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"
}
```

# Where Do You Put the Foreign Key?

Array of books inside of publisher

- Makes sense when many means a handful of items
- Useful when items have bound on potential growth

Reference to single publisher on books

- Useful when items have unbounded growth (unlimited # of books)

SQL doesn't give you a choice, no arrays

Another Example

# One to Many Relations

# Books and Patrons

Book can be checked out by one Patron at a time

Patrons can check out many books (but not 1000's)

# Modeling Checkouts

```
patron = {
    _id: "joe",
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... }
}

book = {
    _id: "123456789",
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    ...
}
```

# Modeling Checkouts

```
patron = {
    _id: "joe",
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... },
    checked_out: [
        { _id: "123456789", checked_out: "2012-10-15" },
        { _id: "987654321", checked_out: "2012-09-12" },
        ...
    ]
}
```

Denormalization

**Provides data locality**

# Modeling Checkouts: Denormalized

```
patron = {
    _id: "joe",
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... },
    checked_out: [
        { _id: "123456789",
          title: "MongoDB: The Definitive Guide",
          authors: [ "Kristina Chodorow", "Mike Dirolf" ],
          checked_out: ISODate("2012-10-15")
        },
        { _id: "987654321"
          title: "MongoDB: The Scaling Adventure",
                                ...
        }, ...
    ]
}
```

# Referencing vs. Embedding

- Embedding is a bit like pre-joined data
- Document-level ops are easy for server to handle
- Embed when the 'many' objects always appear with (i.e. viewed in the context of) their parent
- Reference when you need more flexibility

An Example

# Single Table Inheritance

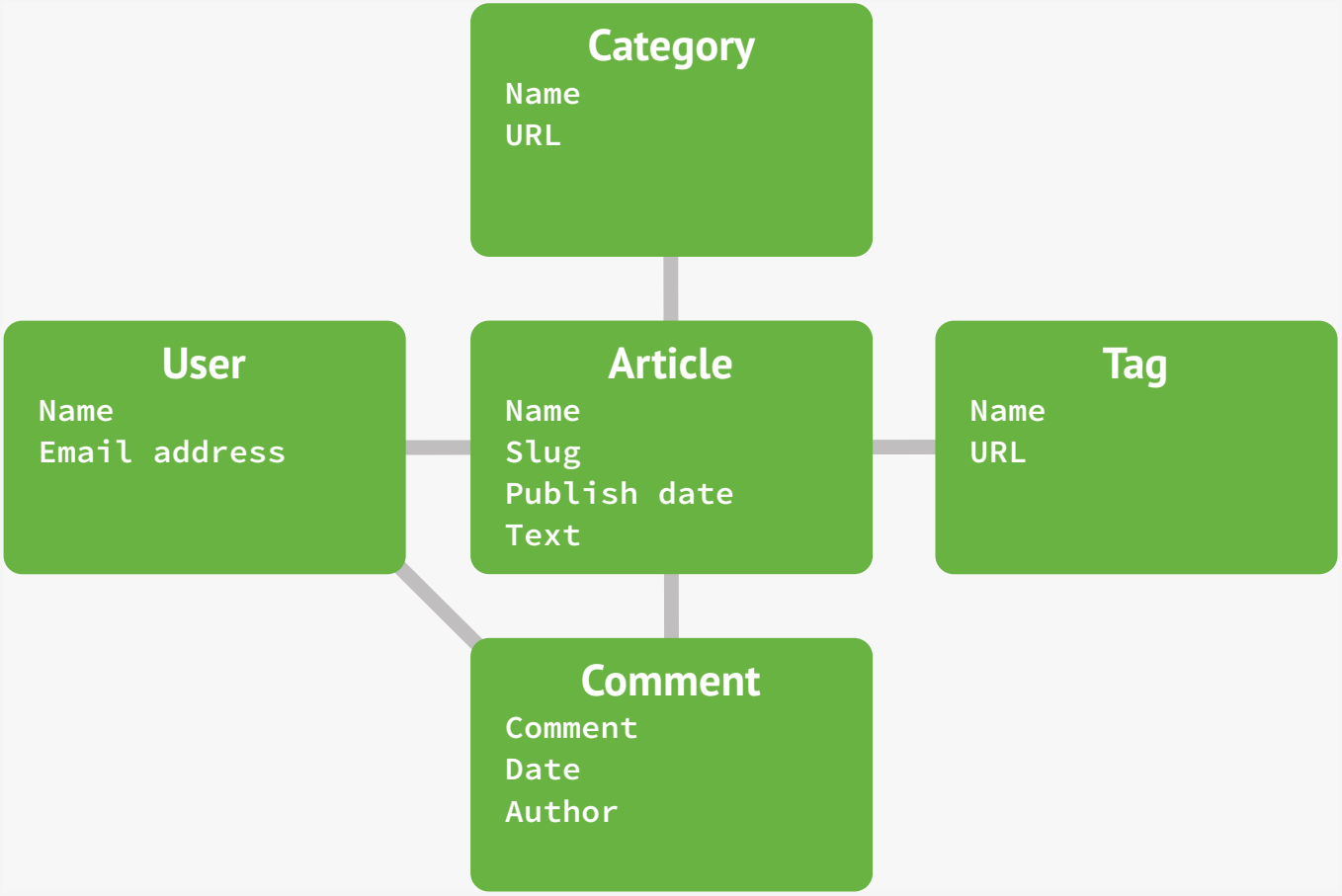
# Single Table Inheritance

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    kind: "loanable",
    locations: [ ... ],
    pages: 216,
    language: "English",
    publisher: {
        name: "O'Reilly Media",
        founded: "1980",
        location: "CA"
    }
}
```

An Example

# Many to Many Relations

# Relational Approach



# Books and Authors

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

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

# 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, ... ]
}
```

# 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" }
```

# Child Links

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

category = { _id: MongoDB, children: [ 123456789, ... ] }
category = { _id: Databases, children: ["MongoDB", "Postgres"}
category = { _id: Programming, children: ["DB", "Languages"] }
```

# Modeling Trees

- Parent Links
  - Each node is stored as a document
  - Contains the id of the parent
- Child Links
  - Each node contains the id's of the children
  - Can support graphs (multiple parents / child)

# Array of Ancestors

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

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

# An Example

# Queues

# Book Document

```
book = {
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    available: 3
}

db.books.findAndModify({
    query: { _id: 123456789, available: { "$gt": 0 } },
    update: { $inc: { available: -1 } }
})
```

#ConferenceHashTag

# Thank You


Speaker Name

*Job Title, 10gen*


mongoDB