mongoDB

Schema Design By Example

Emily Stolfo - twitter: @EmStolfo

Schema design in mongoDB is special

Why is schema design important?

Questions to ask yourself when designing a schema.

What schema elements are available for you to design around?

Examples illustrating different solutions.

Why is schema design important?

- Flexibility
- Data modeled based on usage
- Priorities change

Questions to Ask Yourself

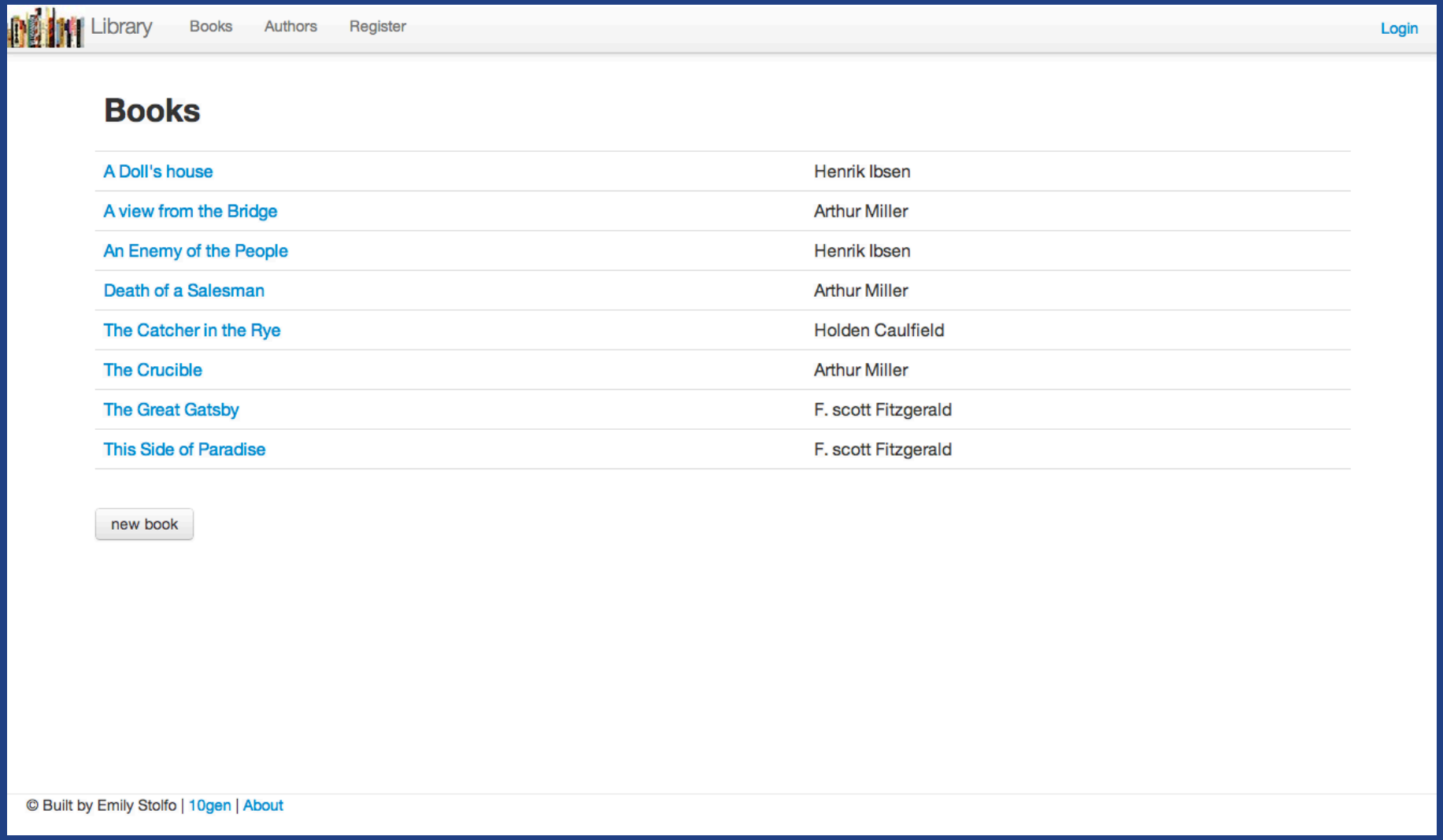
- Will your application be read or write intensive?
- Will documents change over time?
  - Grow or shrink?
  - Normalized or denormalized?
- Are there immutables fields that can be leveraged?
- Will you shard your data?

BSON Documents fields

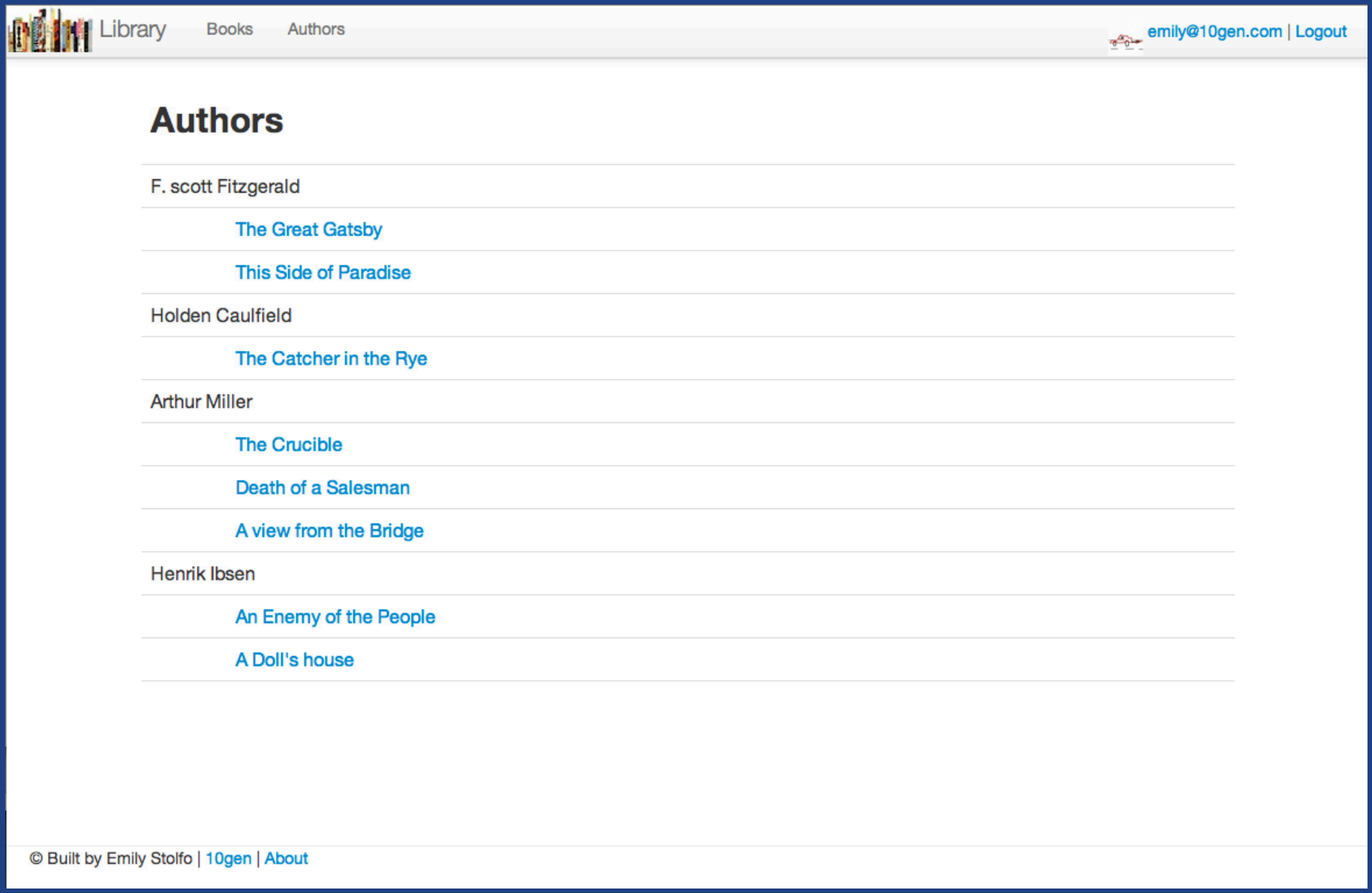
- Primitive Types
  - Double
  - UTF-8 String
  - Binary
  - ObjectId
  - Boolean
  - UTC DateTime
  - Null
  - Regular Expression
  - 32-bit Integer
  - Timestamp
  - 64-bit Integer
- Rich Types
  - Documents
  - Arrays

Example: Library Application

An application for saving a collection of books.


Library
Books
Authors
Register
Login
Books
A Doll's house
Henrik Ibsen
A view from the Bridge
Arthur Miller
An Enemy of the People
Henrik Ibsen
Death of a Salesman
Arthur Miller
The Catcher in the Rye
Holden Caulfield
The Crucible
Arthur Miller
The Great Gatsby
F. scott Fitzgerald
This Side of Paradise
F. scott Fitzgerald
new book
© Built by Emily Stolfo | 10gen | About

library


Library
Books
Authors
emily@10gen.com | Logout
Authors
F. scott Fitzgerald
The Great Gatsby
This Side of Paradise
Holden Caulfield
The Catcher in the Rye
Arthur Miller
The Crucible
Death of a Salesman
A view from the Bridge
Henrik Ibsen
An Enemy of the People
A Doll's house
© Built by Emily Stolfo | 10gen | About

Possible schema for a Book

```
book = {
  "_id": int,
  "title": string,
  "author": int,
  "isbn": string,
  "slug": string,
  "publisher": {
    "name": string,
    "date": timestamp,
    "city": string
  },
  "available": boolean,
  "pages": int,
  "summary": string,
  "subjects": [string, string]
  "notes": [{
        "user": int,
        "note": string
  }],
  "language": string
}
```

Possible Author schema

```
author = {
  "_id": int,
  "first_name": string,
  "last_name": string
}
```

Possible User schema

```
user = {
  "_id": int,
  "username": string,
  "password": string
}
```

Some sample documents

```
> db.authors.findOne()
{
  _id: 1,
  first_name: "F. Scott",
  last_name: "Fitgerald"
}

> db.users.findOne()
{
  _id: 1,
  username: "craig.wilson@10gen.com",
  password: "slsjfk4odk84k209dlkdj90009283d"
}
```

```
> db.books.findOne()
{
  _id: 1,
  title: "The Great Gatsby",
  slug: "9781857150193-the-great-gatsby",
  author: 1,
  available: true,
  isbn: "9781857150193",
  pages: 176,
  publisher: {
    publisher_name: "Everyman's Library",
    date: ISODate("1991-09-19T00:00:00Z"),
    publisher_city: "London"
  },
  summary: "Set in the post-Great War...",
  subjects: ["Love stories", "1920s", "Jazz Age"],
  notes: [
    { user: 1, note: "One of the best..."},
    { user: 2, note: "It's hard to..."}
  ],
  language: "English"
}
```

Is this the only way to structure the data?

Principles and Patterns

Remember: Schema-free != No-Schema

Application-level defined schema (and expectations)

```
user = {
  "_id": int,
  "username": string,
  "password": string
}
```

Flexibility should be leveraged only when appropriate.

```
db.everything.find()
{
  _id: 893,
  first_name: "Jim",
  last_name: "McJim"
}
{
  _id: "'66 mustang",
  make: "Ford",
  model: "Mustang",
  year: "1966"
}
```

Remember: Use Rich Documents (3 patterns)

- Embedded Documents

```
book.publisher:
{
  publisher_name: "Everyman's Library",
  date: ISODate("1991-09-19T00:00:00Z"),
  city: "London",
},
```

- Embedded Arrays

```
book.subjects: ["Love stories", "1920s", "Jazz Age"],
```

- Embedded Arrays of Documents

```
book.notes: [
  { user: 1, note: "One of the best..."},
  { user: 2, note: "It's hard to..."}
],
```

What is the next step?

Example Application Queries

Query for all the books by a specific author

```
> author = db.authors.findOne({first_name: "F. Scott", last_name: "Fitzgerald"});

> db.books.find({author: author._id})
{
  ...
}
{
  ...
}
```

Query for books by title

```
> db.books.find({title: "The Great Gatsby"})
{
  ...
}
```

Query for books in which I have made notes

```
> db.books.find({notes.user: 1})
{
  ...
}
{
  ...
}
```

What is the read usage of my data?

Library Books Authors emily@10gen.com | Logout

Authors

F. scott Fitzgerald

- The Great Gatsby
- This Side of Paradise

Holden Caulfield

- The Catcher in the Rye

Arthur Miller

- The Crucible
- Death of a Salesman
- A view from the Bridge

Henrik Ibsen

- An Enemy of the People
- A Doll's house

Schema Evolution for read-heavy use #1

Frequent queries for books given an author's first name

```
> authors = db.authors.find( { first_name: /^f.*/i })

// for each author in authors, authorIds.append(author._id)

> db.books.find({author: { $in: authorIds } })
{
  ...
}
{
  ...
}
```

Remember: Documents Should Reflect Query Patterns

- Pattern: Partially Embedded Documents

```
book = {
  "_id": int,
  "title": string,
  "author": {
    "author": int,
    "name": string
  },
  ...
}
```

- Query for books by an author's first name using an embedded document with cached name.

```
> db.books.find({author.name: /^f.*/i })
{
  ...
}
{
  ...
}
```

Schema Evolution for read-heavy use #2

Frequent queries for a book's notes

```
> db.books.find({title: "The Great Gatsby"}, {_notes: 1})
{
  _id: 1,
  notes: [
    { user: 1, note: "One of the best..."},
    { user: 2, note: "It's hard to..."}
  ]
}
```

Remember: Take Advantage of Immutable Data

- Username is the natural key and is immutable

```
user = {
  "_id": string,
  "password": string
}

book = {
  // ...
  "notes": [{
    "user": string,
    "note": string
  }],
  // ...
}
```

- Query for a book's notes

```
> db.books.find({title: "The Great Gatsby"}, {_notes: 1})
{
  _id: 1,
  notes: [
    { user: "craig.wilson@10gen.com", note: "One of the best..."},
    { user: "jmcjack@mcjack.com", note: "It's hard to..."}
  ]
}
```

Schema Evolution for read-heavy use #3

Users want to comment on other people's notes

```
bookNotes = {
  "_id": int,
  "book": int, // this is the book id
  "note_count": int,
  "last_changed": datetime,
  "notes": [{
    "user": string,
    "note": string,
    "comments": [{
      "user": string,
      "text": string,
      "replies": [{
        "user": string,
        "text": string,
        "replies": [{...}]
      }]
    }]
  }]
}
```

Why store a tree like this?

- Single document for all comments on a note.
- Single location on disk for the whole tree.
- Legible tree structure.

What are the drawbacks of storing a tree in this way?

- Difficult to search.
- Difficult to get back partial results.
- Document can get large very quickly.

There is no formula.
Maybe store arrays of ancestors?

```
> t = db.mytree;

> t.find()
{ _id: "a" }
{ _id: "b", ancestors: [ "a" ], parent: "a" }
{ _id: "c", ancestors: [ "a", "b" ], parent: "b" }
{ _id: "d", ancestors: [ "a", "b" ], parent: "b" }
{ _id: "e", ancestors: [ "a" ], parent: "a" }
{ _id: "f", ancestors: [ "a", "e" ], parent: "e" }
{ _id: "g", ancestors: [ "a", "b", "d" ], parent: "d" }
```

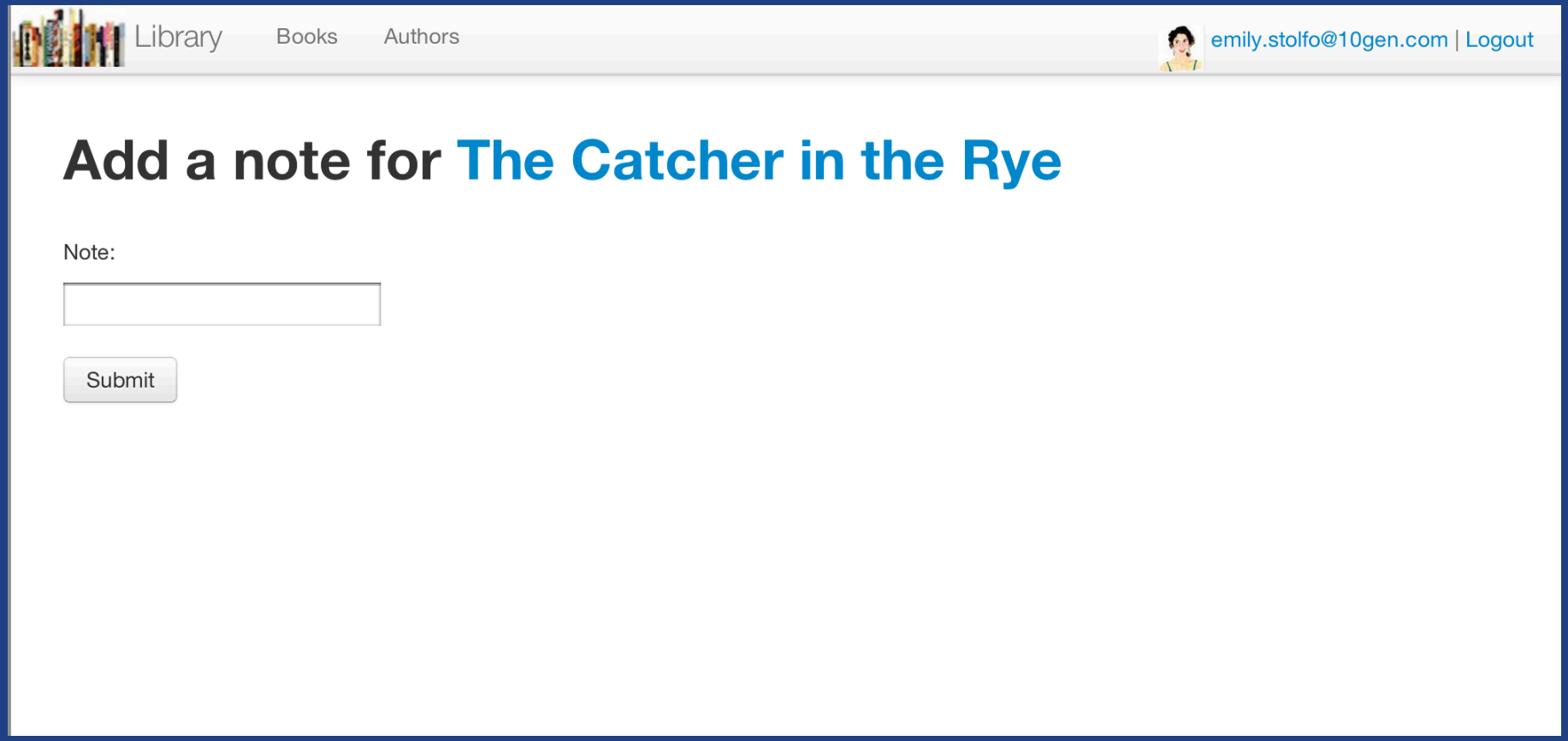
How would you do it?

Read-heavy schemas

Don’t forget to think about indexes and sharding

- Multi-key indexes.
- Secondary indexes.
- Shard key choice.

What about writes?



Schema Evolution for write-heavy use

Add notes to a book

```
> note = { user: "craig.wilson@10gen.com", "I did NOT like this book." }

> db.books.update({ _id: 1 }, { $push: { notes: note }})
```

Remember: Take Advantage of Atomic Operations

- $set - set a value
- $unset - unsets a value
- $inc - increment an integer
- $push - append a value to an array
- $pushAll - append several values to an array
- $pull - remove a value from an array
- $pullAll - remove several values from an array
- $bit - bitwise operations
- $addToSet - adds a value to a set if it doesn't already exist
- $rename - renames a field

Think about anti-patterns: Continually Growing Documents

- Document size limit.
- Storage fragmentation and update performance.

What is an alternative to storing the notes in an array?

Possible Solution: Linking

Move notes to a notes collection.

```
book = {
  "_id": int,
  ... // remove the notes field
}

bookNotes = {
  "_id": int,
  "book": int, // this will be the same as the book id...
  "date": timestamp,
  "user": string,
  "note": string
}
```

- Book document size is consistent.
- Queries for books don't return all the notes.
- Possible slow reads.

Possible solution: Bucketing

bookNotes contains a limited-size array

```
bookNotes = {
  "_id": int,
  "book": int, // this is the book id
  "note_count": int,
  "last_changed": datetime,
  "notes": [{
    "user": string,
    "note": string
  }]
}
```

Atomic operations for updating or creating

```
> note = { user: "craig.wilson@10gen.com", "I did NOT like this book." }

> db.bookNotes.update({
        { book: 1, note_count { $lt: 10 } },
        {
                $inc: { note_count: 1 },
                $push { notes: note },
                $set: { last_changed: new Date() }
        },
        true // upsert
})
```

Your (application + schema) + mongodb = <3

Basic design principles apply.

Focus on how your application uses the data.

Anticipate document and collection growth.

Take advantage of the mongodb's flexibility and features