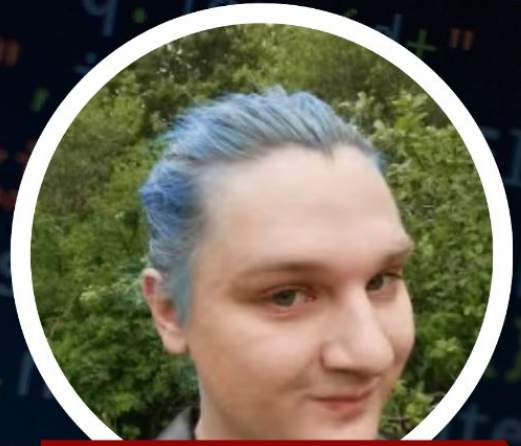
CityJS™

Effective Data Modeling for Document Databases



Anna Henningsen

Who am I?

- `@addaleax` online
- Former full-time Node.js core contributor & TSC member
- Working on Developer Tools @ MongoDB for 5½ years now
- Triple cat mom

Slides @
https://addaleax.net/cityjs-singapore26.pdf

Agenda

Why am I talking about this

The Core Idea

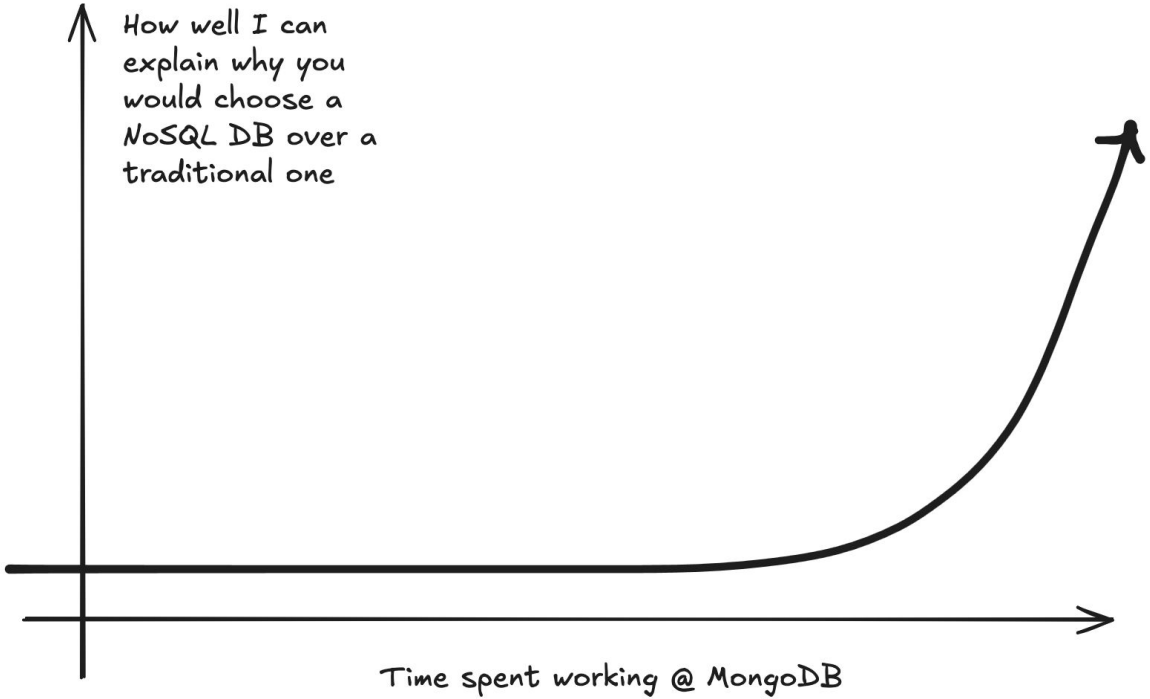
A Myth Or Two

What we're working on

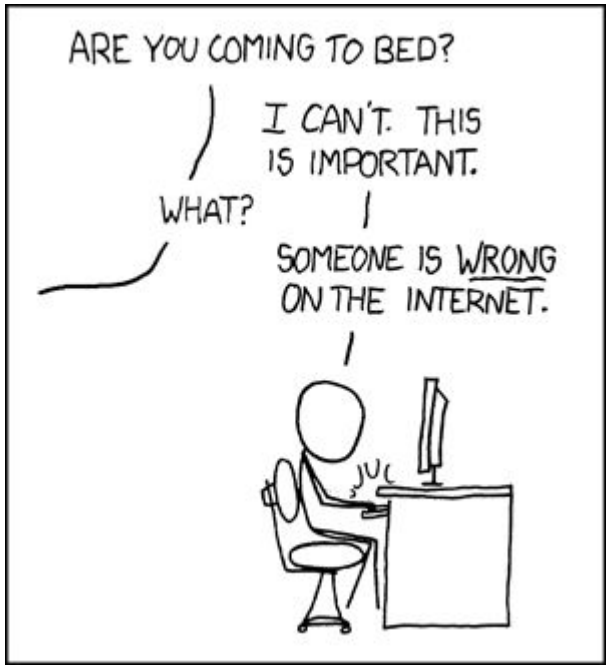
Why am I talking about this?

- Yes, I know it’s not 2012
- But I’m always excited to share what I learned!
- Nobody is forcing me to speak about MongoDB things

Why am I talking about this?



Why am I talking about this?



When talking about DB performance

Try to picture this:

When talking about DB performance

Try to picture this:

- Drawers = Tables/Collections
- Folders = Rows/Documents
- Labels = Indexes

The Core Idea

The Core Idea

1. **You** are in **control** over your data layout

2. Data that is **accessed together** is **stored together**

The Core Idea ... and its consequences

1. **You** are in **control** over your data layout
    - **You** will **have to think** about your data layout
2. Data that is **accessed together** is **stored together**
    - Your data will **not always be normalized**
    - **Duplicating your data** may be okay or even good!

A Classic Example

Let’s create a web shop

- We have customers, items, orders, reviews, the whole thing

Let’s create a web shop

- We have customers, items, orders, reviews, the whole thing
- In the relational world, we don’t have to think
    - Every type of entity just gets its own table

Let’s create a web shop

- We have customers, items, orders, reviews, the whole thing
- In the relational world, we don’t have to think
  - Every type of entity just gets its own table
- For Document Databases, we first have to think…
  - What do our *most* common read and write patterns look like?

Let’s create a web shop

1. *Customers* have *addresses*, but almost always a small number
2. When *Items* are fetched, a summary of *Reviews* is displayed
3. Older *Orders* are rarely accessed
    - But a user might want to see a summary of its last few orders
4. *Items* may come in variants that share many common attributes
    - E.g.: Clothing in different sizes and different colors

Let's create a web shop

- *Customers* have *addresses*, but almost always a small number

Customers

```
{
  _id: "username",
  addresses: [
    {
      street: "...",
      country: "...",
      ...
    }
  ],
  ...
}
```

No separate collection for addresses – just store them in-line with users

Let's create a web shop

- When *Items* are fetched, a summary of *Reviews* is displayed

Reviews	Items
{ username: "username", item: "item1234", rating: 4, content: "..." }	{ _id: "item1234", title: "...", **review_summary**: [12, 18, 53, 86, 43], ... }

- ... and yes, this means *two* writes per new review
  - and you may or may not want a transaction for that!

Let's create a web shop

- Older *Orders* are rarely accessed
  - But a user might want to see a summary of its last few orders

Customers	Orders

<pre>{
  _id: "username",
  addresses: [...],
  order_summaries: [
  {
  _id: 1234529835,
  date: ISODate('2026-01-04T10:04:08.461Z'),
  total: 123.45,
  total_currency: "SGD",
  status: 'delivered' | 'processing' | ...
  },
  ...
  ]
}</pre> Orders are mostly static – duplication is fine
	<pre>{
	  _id: 1234529835,
	  date: ISODate('2026-01-04T10:04:08.461Z'),
	  total: 123.45,
	  total_currency: "SGD",
	  status: 'delivered' | 'processing' | ...,
	  items: [
	  { id: 1234, quantity: 10 },
	  { id: 4567, quantity: 3, variant: "Red" },
	  ],
	  delivery_address: { ... }
	}</pre> There should be exactly one source of truth – other copies are caches

Let’s create a web shop

- *Items* may come in variants that share many common attributes
  - E.g.: Clothing in different sizes and different colors

Items

```
{
  _id: "item1234",
  title: "...",
  review_summary: [ ... ],
  variants: [
    { id: "Red", description: "..." }
  ]
}
```

Fully optional subfield

A Myth Or Two

Myth I: NoSQL is just JSON storage

Myth I: NoSQL is just JSON storage

- MongoDB alone supports …
  - Joins
  - Views
  - Transactions
  - Full-text search
  - Data encryption & encrypted search
  - Vector storage for AI
  - …

Myth II: NoSQL is schemaless

Myth II: NoSQL is schemaless

- Schema validation *is* possible
- Schema validation *is* encouraged
- Schema validation *is* accurate

Myth II: NoSQL is schemaless

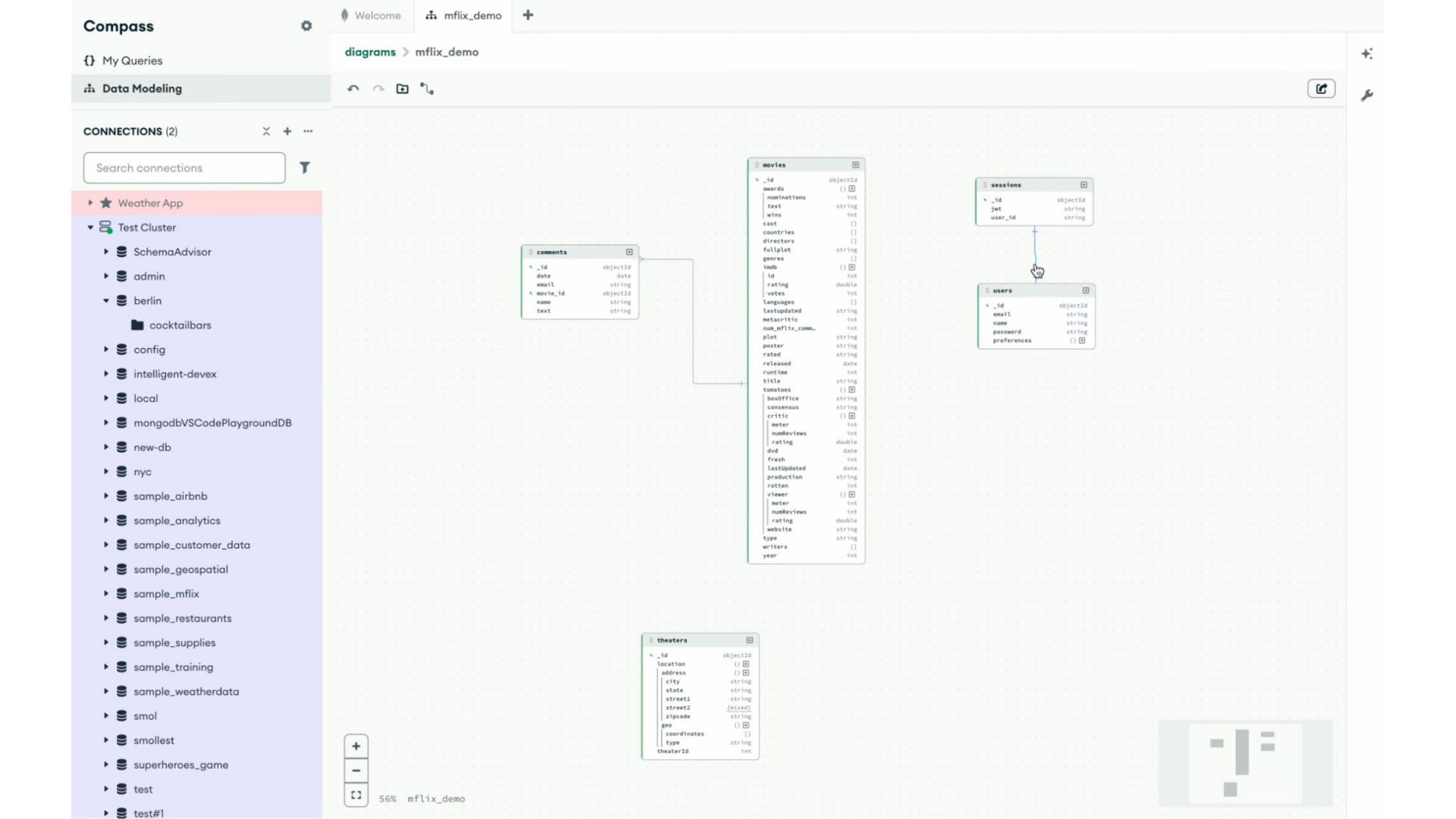
- Schema validation *is* possible
- Schema validation *is* encouraged
- Schema validation *is* accurate

```
import { withSchema } from '@mongodb-js/mongodb-zod';

const client = await MongoClient.connect(process.env.MONGODB_URI);
const coll = withSchema(
  client.db('test').collection('users'),
  z.object({
    _id: z.string(),
    name: z.string(),
    age: z.number().min(0),
  }),
);

await coll.installSchemaValidation();
```

What we're working on

What we're working on

- Our team is working on tools to ...
  - Visualize your MongoDB schema
  - Improve your MongoDB schema with intelligent recommendations
  - Analyze your Database Cluster for performance gaps
  - Make schema validation easier to integrate
  - Document best practices in a central location
  - Make it easier to migrate from one Data Model to another


Compass
My Queries
Data Modeling
CONNECTIONS (2)
Search connections
Weather App
Test Cluster
SchemaAdvisor
admin
berlin
cocktailbars
config
intelligent-devex
local
mongodbVSCodePlaygroundDB
new-db
nyc
sample_airbnb
sample_analytics
sample_customer_data
sample_geospatial
sample_mflix
sample_restaurants
sample_supplies
sample_training
sample_weatherdata
smol
smollest
superheroes_game
test
test#1
Welcome
mflix_demo
diagrams
mflix_demo
comments
movies
sessions
users
theaters
56%
mflix_demo

Thank you for your time.