



MongoDB Schema Design Best Practices

Joe Karlsson | Developer Advocate | @JoeKarlsson1

Why is schema
design so important?

Critical for improving the
performance and scalability
of your database



```
{  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  title: [  
    "Developer Advocate",  
    "Software Engineer"  
  ],  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com",  
  opinions: "my own",  
  links: "bit.ly/IoTKittyBox"  
}
```



Agenda

- Relational vs. MongoDB
Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships

Agenda

- **Relational vs. MongoDB
Schema Design Approaches**
- Embedding vs. Referencing
- Types of Relationships



Relational vs. MongoDB Schema Design Approaches

**RELATIONAL
SCHEMA
DESIGN**

**MONGODB
SCHEMA
DESIGN**

Corporate needs you to find the differences
between this picture and this picture.


Most Devs

They're the same picture.



Relational Schema Design

Model data independent of queries



What
data do
I have?

Prescribed approaches

Normalize in the 3rd form



tl;dr:

Don't duplicate data

Users

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	44755750561 1	London	45.123	47.232

Professions

ID	user_id	profession
10	1	banking
11	1	finance
12	1	trader

Cars

ID	user_id	model	year
20	1	Bentley	1973
21	1	Rolls Royce	1965

MongoDB Schema Design

MongoDB

Schema

Design

No Rules

No Process

No Algorithm

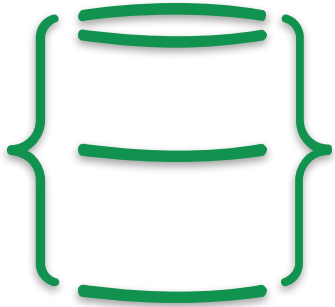


A close-up shot of a woman with dark hair pulled back, wearing heavy, colorful eye makeup in shades of red, orange, and yellow. She has a cigarette in her mouth and is looking off to the side with a serious expression. The background is blurred, showing other people at what appears to be a party or social gathering. The text "THERE ARE NO RULES" is overlaid at the bottom in a bold, white, sans-serif font.

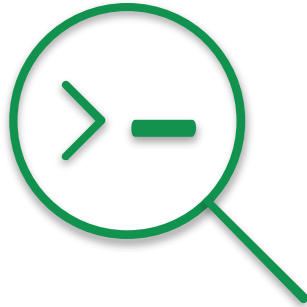
THERE ARE NO RULES

Design a schema that will work well for a given application

MongoDB Schema Design



How to store the data



Query Performance



Reasonable amount of
hardware



Users

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	44755750561 1	London	45.123	47.232

Professions

ID	user_id	profession
10	1	banking
11	1	finance
12	1	trader

Cars

ID	user_id	model	year
20	1	Bentley	1973
21	1	Rolls Royce	1965

Relational

Users

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	447557505611	London	45.123	47.232

MongoDB

```
{  
  first_name: "Paul",  
  surname: "Miller",  
  cell: "447557505611",  
  city: "London",  
  location: [45.123,47.232],  
}
```

Relational

Users

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	447557505611	London	45.123	47.232

Professions

ID	user_id	profession
10	1	banking
11	1	finance
12	1	trader

MongoDB

```
{  
  first_name: "Paul",  
  surname: "Miller",  
  cell: "447557505611",  
  city: "London",  
  location: [45.123,47.232],  
  profession: ["banking", "finance", "trader"],  
}
```

Relational

Users

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	447557505611	London	45.123	47.232

Professions

ID	user_id	profession
10	1	banking
11	1	finance
12	1	trader

Cars

ID	user_id	model	year
20	1	Bentley	1973
21	1	Rolls Royce	1965

MongoDB

```
{
  first_name: "Paul",
  surname: "Miller",
  cell: "447557505611",
  city: "London",
  location: [45.123,47.232],
  profession: ["banking", "finance", "trader"],
  cars: [
    {
      model: "Bentley",
      year: 1973
    },
    {
      model: "Rolls Royce",
      year: 1965
    }
  ]
}
```

● Relational vs. MongoDB Schema Design Approaches

RECAP

● Relational vs. MongoDB Schema Design Approaches

● Relational Schema Design

RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries

RECAP



RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	447557505611	London	45.123	47.232

ID	user_id	profession
10	1	banking
12	1	trader

ID	user_id	model	year
20	1	Bentley	1973

RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form
 - MongoDB Schema Design {🔗}

RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form
 - MongoDB Schema Design {💡}
 - No rules, no process, no algorithm

RECAP

- Relational vs. MongoDB Schema Design Approaches

- Relational Schema Design

- Model data independent of queries
 - Normalize in the 3rd form

- MongoDB Schema Design {🔗}

- No rules, no process, no algorithm

- Considerations:

- How to store the data
 - Query Performance

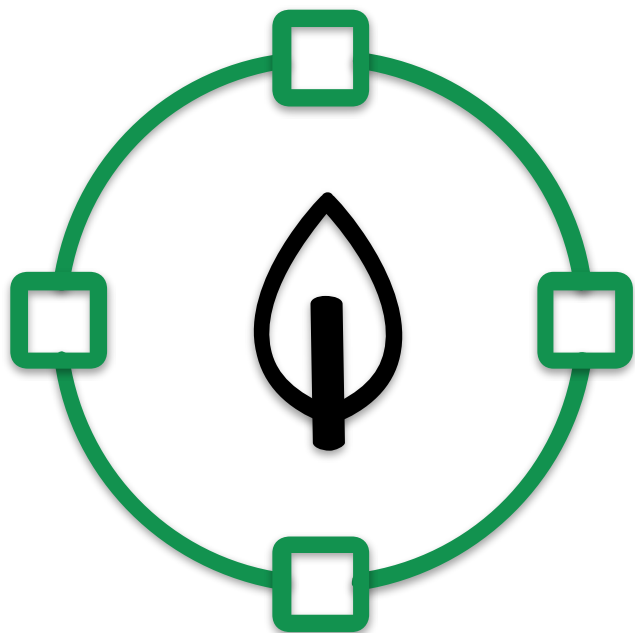
RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form
 - MongoDB Schema Design {🔗}
 - No rules, no process, no algorithm
 - Considerations:
 - How to store the data
 - Query Performance
 - Design a schema that works for your application

Agenda

- Relational vs. MongoDB
Schema Design Approaches
- **Embedding vs. Referencing**
- Types of Relationships

Embedding vs. Referencing



Embedding

```
{  
  _id : ObjectId('AAA'),  
  name: 'Kate Monster',  
  ssn: '123-456-7890',  
  addresses: [  
    {  
      street: '123 Sesame St',  
      city: 'Anytown', cc: 'USA'  
    },  
    {  
      street: '123 Avenue Q',  
      city: 'New York',  
      cc: 'USA'  
    }  
  ]  
}
```

Embedding

```
{  
  ...  
  a: "b",  
  ...  
  c: {  
    d: "e"  
    ...  
  },  
  ...  
}
```

Join

...	d	...
1	e	...
...

ID	a	...
1	b	...
2
3



Embedding

Pro



Retrieve all data with a single query



Avoids expensive JOINs or \$lookup



Update all data with a single atomic operation

Con

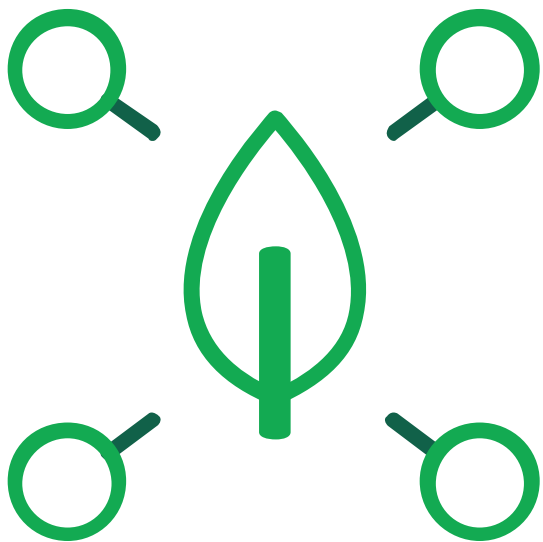


Large docs === more overhead



16 MB Document size limit





Referencing

```
{  
  name : 'left-handed smoke shifter',  
  manufacturer : 'Acme Corp',  
  catalog_number: 1234,  
  parts : [  
    ObjectID('AAAA'),  
    ObjectID('BBBB'),  
  ]  
}  
  
{  
  _id : ObjectID('AAAA'),  
  partno : '123-aff-456',  
  name : '#4 grommet',  
  qty: 94,  
  cost: 0.94,  
  price: 3.99  
}  
  
{  
  _id : ObjectID('BBB'),  
  partno : '425-EFF-123',  
  name : '#8 Frombet',  
  qty: 13,  
  cost: 0.34,  
  price: 7.99  
}
```

Two dashed green arrows originate from the 'parts' array in the first JSON object. One arrow points from 'ObjectID('AAAA')' to the first object in the second block, and the other points from 'ObjectID('BBBB')' to the second object in the second block.

Referencing

Pro



Smaller documents



Less likely to reach 16 MB limit



No duplication of data



Infrequently accessed data
not accessed on every query

Con



Two queries or \$lookup
required to retrieve all data



- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing

RECAP

RECAP

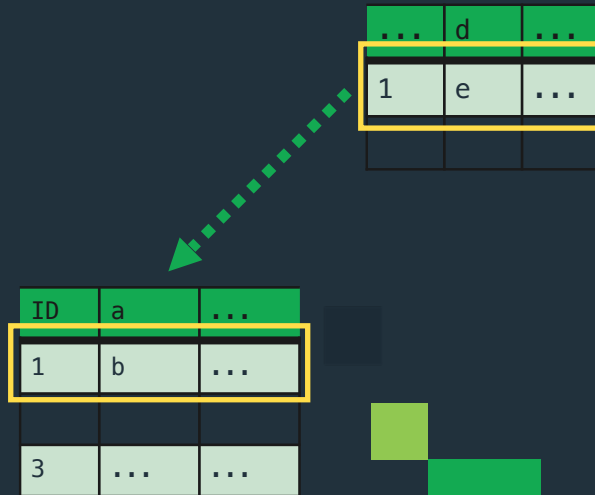
- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Embedding:

```
{
  _id : ObjectId('AAA'),
  name: 'Kate Monster',
  ssn: '123-456-7890',
  addresses: [
    {
      street: '123 Sesame St',
      city: 'Anytown', cc: 'USA'
    },
    {
      street: '123 Avenue Q',
      city: 'New York',
      cc: 'USA'
    }
  ]
}
```

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
 - Embedding:

```
{  
  ...  
  a: "b",  
  ...  
  c: {  
    d: "e"  
    ...  
  },  
  ...  
}
```



RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
 - Embedding:
 - ✓ Retrieve all data with a single query
 - ✓ Avoids expensive JOINS or \$lookup
 - ✓ Update all data with a single atomic operation

RECAP

- Relational vs. MongoDB Schema Design Approaches

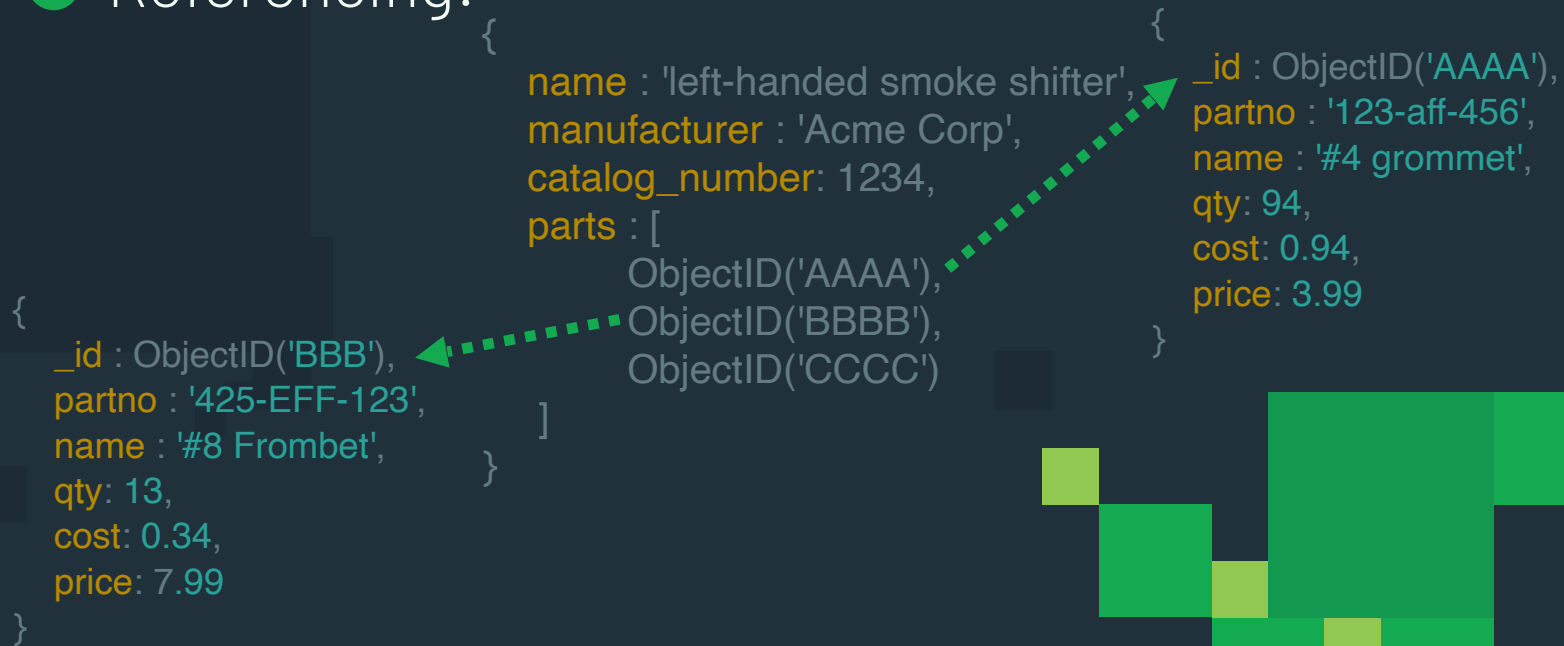
- Embedding vs. Referencing

- Embedding:

- ✓ Retrieve all data with a single query
 - ✓ Avoids expensive JOINS or \$lookup
 - ✓ Update all data with a single atomic operation
 - ✗ Large docs === more overhead
 - ✗ 16 MB Document size limit

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
 - Embedding
 - Referencing:



RECAP

- Relational vs. MongoDB Schema Design Approaches

- Embedding vs. Referencing

- Embedding

- Referencing:

- ✓ Smaller documents

- ✓ Less likely to reach 16 MB limit

- ✓ No duplication of data

- ✓ Infrequently accessed data not accessed on every query

RECAP

- Relational vs. MongoDB Schema Design Approaches

- Embedding vs. Referencing

- Embedding

- Referencing:

- ✓ Smaller documents

- ✓ Less likely to reach 16 MB limit

- ✓ No duplication of data

- ✓ Infrequently accessed data not accessed on every query

- ✗ Two queries or \$lookup required to retrieve all data

Agenda

- Relational vs. MongoDB
Schema Design Approaches
- Embedding vs. Referencing
- **Types of Relationships**

Types of Relationships

One to One

One to One



Use Key-Value pairs

User:

```
{  
  _id: ObjectId("AAA"),  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com"  
}
```

One to Few

One to Few

```
{
  _id: ObjectId("AAA"),
  name: "Joe Karlsson",
  company: "MongoDB",
  twitter: "@JoeKarlsson1",
  twitch: "joe_karlsson",
  tiktok: "joekarlsson",
  website: "joekarlsson.com",
  addresses: [
    { street: "123 Sesame St", city: "Anytown", cc: "USA" },
    { street: "123 Avenue Q", city: "New York", cc: "USA" }
  ]
}
```



Prefer embedding

Rule 1:

Favor embedding unless
there is a compelling
reason not to.



Rule 2:

Needing to access an object on its own is a compelling reason not to embed it.



One to Many

One to Many



Prefer referencing

Products:

```
{
  _id: ObjectId("123"),
  name: "left-handed smoke shifter",
  manufacturer: "Acme Corp",
  catalog_number: 1234,
  parts: [
    ObjectId("AAA"),
    ObjectId("BBB"),
    ObjectId("CCC"),
  ]
}
```

Parts:

```
{
  _id: ObjectId("AAA"),
  partno: "123-ABC-456",
  name: "#4 grommet",
  qty: 94,
  cost: 0.54,
  price: 2.99,
}
```

Rule 3:

Avoid JOINS and \$lookups if they can be, but don't be afraid if they can provide a better schema design.



One to Squillions

One to Squillions



Prefer referencing

Hosts:

```
{
  _id: ObjectId("AAA"),
  name: "goofy.example.com",
  ipaddr: "127.66.66.66",
}
```

Log Message:

```
{
  _id: ObjectId("123"),
  time: ISODate("2014-03-28T09:42:41.382Z"),
  message: "The CPU is on fire!!!",
  host: ObjectId("AAA"),
},
{
  _id: ObjectId("456"),
  time: ISODate("2014-03-28T09:42:41.382Z"),
  message: "Drive is hosed",
  host: ObjectId("AAA"),
}
```

Rule 4:

Arrays should not
grow without bound.



Many to Many

Many to Many



Prefer referencing

Person:

```
{
  _id: ObjectId("AAF1"),
  name: "Joe Karlsson",
  tasks: [
    ObjectId("ADF9"),
    ObjectId("AE02"),
    ObjectId("ZDF2"),
  ]
}
```

Tasks:

```
{
  _id: ObjectId("ADF9"),
  description: "Learn MongoDB",
  due_date: ISODate("2014-03-28T09:42:41.382Z"),
  owner: ObjectId("AAF1"),
},
{
  _id: ObjectId("AE02"),
  description: "Write lesson plan",
  due_date: ISODate("2014-03-28T09:42:41.382Z"),
  owner: ObjectId("AAF1"),
},
```

Rule 5:

How you model your data depends – entirely – on your particular application's data access patterns.



RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs

```
{  
  _id: ObjectId("AAA"),  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com"  
}
```

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs
 - One to Few: Prefer embedding

```
{
  _id: ObjectId("AAA"),
  name: "Joe Karlsson",
  company: "MongoDB",
  twitter: "@JoeKarlsson1",
  website: "joekarlsson.com",
  addresses: [
    { street: "123 Sesame St", city: "Anytown", cc: "USA" },
    { street: "123 Avenue Q", city: "New York", cc: "USA" }
  ]
}
```



- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs
 - One to Few: Prefer embedding
 - One to Many: Prefer Referencing

```
{
  _id: ObjectId("123"),
  name: "left-handed smoke shifter",
  manufacturer: "Acme Corp",
  catalog_number: 1234,
  parts: [
    ObjectId("AAA"),
    ObjectId("BBB"),
    ObjectId("CCC"),
  ]
}
```

```
{
  _id: ObjectId("AAA"),
  partno: "123-ABC-456",
  name: "#4 grommet",
  qty: 94,
  cost: 0.54,
  price: 2.99,
}
```

RECAP

● Relational vs. MongoDB Schema Design Approaches

● Embedding vs. Referencing

● Types of Relationships

- One to One: Use Key-Value pairs
- One to Few: Prefer embedding
- One to Many: Prefer Referencing
- One to Squillions: Prefer Referencing

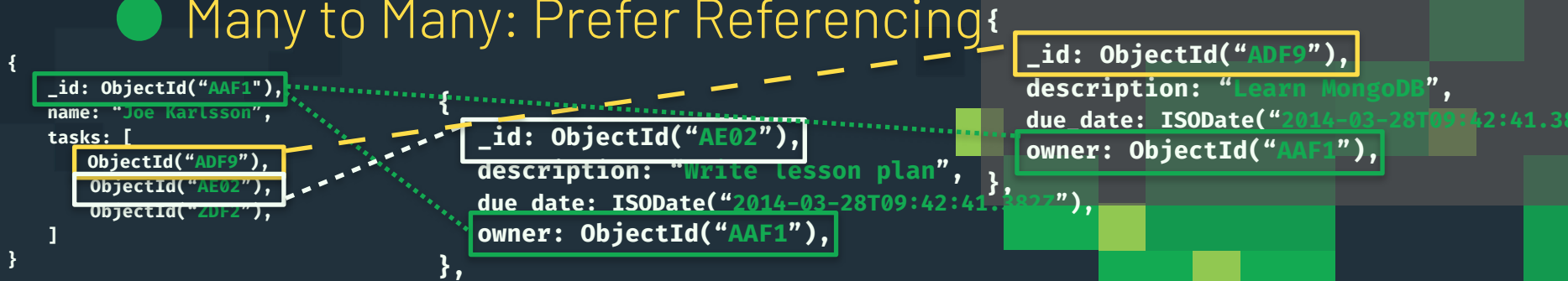
```
{
  _id: ObjectId("AAA"),
  name: "goofy.example.com",
  ipaddr: "127.66.66.66",
}
```

```
{
  _id: ObjectId("456"),
  time: ISODate("2014-03-28T09:42:41.382Z"),
  message: "Drive is hosed",
  host: ObjectId("AAA"),
}
```

```
{
  _id: ObjectId("123"),
  time: ISODate("2014-03-28T09:42:41.382Z"),
  message: "The CPU is on fire!!!",
  host: ObjectId("AAA"),
},
```

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs
 - One to Few: Prefer embedding
 - One to Many: Prefer Referencing
 - One to Squillions: Prefer Referencing
 - Many to Many: Prefer Referencing



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to



MongoDB Schema Design Rules

RECAP

- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it
- Avoid JOINS and \$lookups if they can be avoided



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it
- Avoid JOINS and \$lookups if they can be avoided
- Arrays should not grow without bound



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it
- Avoid JOINS and \$lookups if they can be avoided
- Arrays should not grow without bound
- How you model your data depends – entirely – on your particular application's data access patterns



Hybrid Approaches



Advanced



Not all documents in a collection
need to have the same fields.



Polymorphic Pattern

```
{  
  _id: ObjectId("AAA"),  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com"  
}
```

```
{  
  _id: ObjectId("BBB"),  
  name: "BMO",  
  city: "Minneapolis"  
}
```

Imagine you are
building a Twitter like
social media site...



A Normal User

```
{
  _id: "JoeKarlsson1",
  displayName: "Joe Karlsson",
  numFollowers: "11563",
  followers: [
    "mongodb",
    "jdrumgoole",
    "Lauren_Schaefer",
    ...
  ]
}
```

But what if
Kim Kardashian
joined your site?





Outlier Pattern

```
{
  _id: "KimKardashian",
  display_name: "Kim Kardashian West",
  num_followers: "64500485",
  followers: [
    "KrysJenner",
    "Caitlyn_Jenner",
    "chrissyteigen",
    ...
  ],
  has_extras: true
}
```

```
{
  _id: "KimKardashian_1",
  twitter_id: "KimKardashian",
  is_overflow: "True",
  followers: [
    "kanyewest",
    "TheEllenShow",
    "Oprah",
    ...
  ],
  has_extras: true
}
```

But wait, there are a
ton of other patterns in
your tool belt



Use Case Categories

Patterns

Approximation
Attribute
Bucket
Computed
Document Versioning
Extended Reference
Outlier
Preallocated
Polymorphic
Schema Versioning
Subset
Tree and Graph

	Catalog	Content Management	Internet of Things	Mobile	Personalization	Real-Time Analytics	Single View
Approximation	✓		✓	✓		✓	
Attribute	✓	✓					✓
Bucket			✓			✓	
Computed	✓		✓	✓	✓	✓	✓
Document Versioning	✓	✓			✓		✓
Extended Reference	✓			✓		✓	
Outlier			✓	✓	✓		
Preallocated			✓			✓	
Polymorphic	✓	✓		✓			✓
Schema Versioning	✓	✓	✓	✓	✓	✓	✓
Subset	✓	✓		✓	✓		
Tree and Graph	✓	✓					

● Relational vs. MongoDB Schema Design Approaches

RECAP

● Relational vs. MongoDB Schema Design Approaches

● Relational Schema Design

RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries

RECAP



RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form

ID	first_name	surname	cell	city	location_x	location_y
1	Paul	Miller	447557505611	London	45.123	47.232

ID	user_id	profession
10	1	banking
12	1	trader

ID	user_id	model	year
20	1	Bentley	1973

RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form
 - MongoDB Schema Design {🔗}

RECAP

- Relational vs. MongoDB Schema Design Approaches
 - Relational Schema Design
 - Model data independent of queries
 - Normalize in the 3rd form
 - MongoDB Schema Design {💡}
 - No rules, no process, no algorithm

RECAP

- Relational vs. MongoDB Schema Design Approaches

- Relational Schema Design

- Model data independent of queries
 - Normalize in the 3rd form

- MongoDB Schema Design {🔗}

- No rules, no process, no algorithm

- Considerations:

- How to store the data
 - Query Performance

RECAP

- Relational vs. MongoDB Schema Design Approaches

- Relational Schema Design

- Model data independent of queries
- Normalize in the 3rd form

- MongoDB Schema Design {🔗}

- No rules, no process, no algorithm
- Considerations:

- How to store the data
- Query Performance

- Design a schema that works for your application

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing

RECAP

RECAP

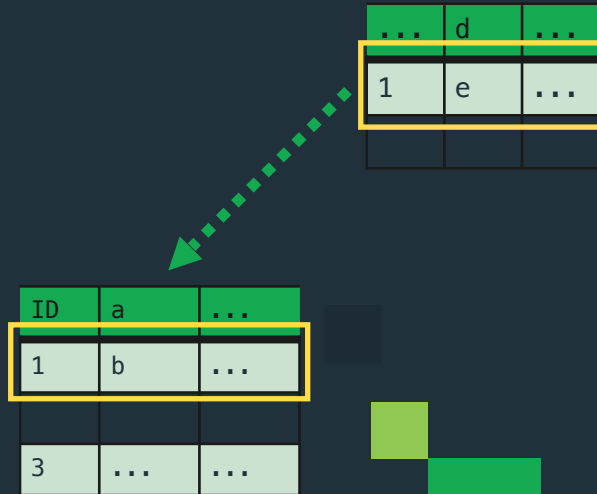
- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Embedding:

```
{
  _id : ObjectId('AAA'),
  name: 'Kate Monster',
  ssn: '123-456-7890',
  addresses: [
    {
      street: '123 Sesame St',
      city: 'Anytown', cc: 'USA'
    },
    {
      street: '123 Avenue Q',
      city: 'New York',
      cc: 'USA'
    }
  ]
}
```

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
 - Embedding:

```
{  
  ...  
  a: "b",  
  ...  
  c: {  
    d: "e"  
    ...  
  },  
  ...  
}
```



RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
 - Embedding:
 - ✓ Retrieve all data with a single query
 - ✓ Avoids expensive JOINS or \$lookup
 - ✓ Update all data with a single atomic operation

RECAP

- Relational vs. MongoDB Schema Design Approaches

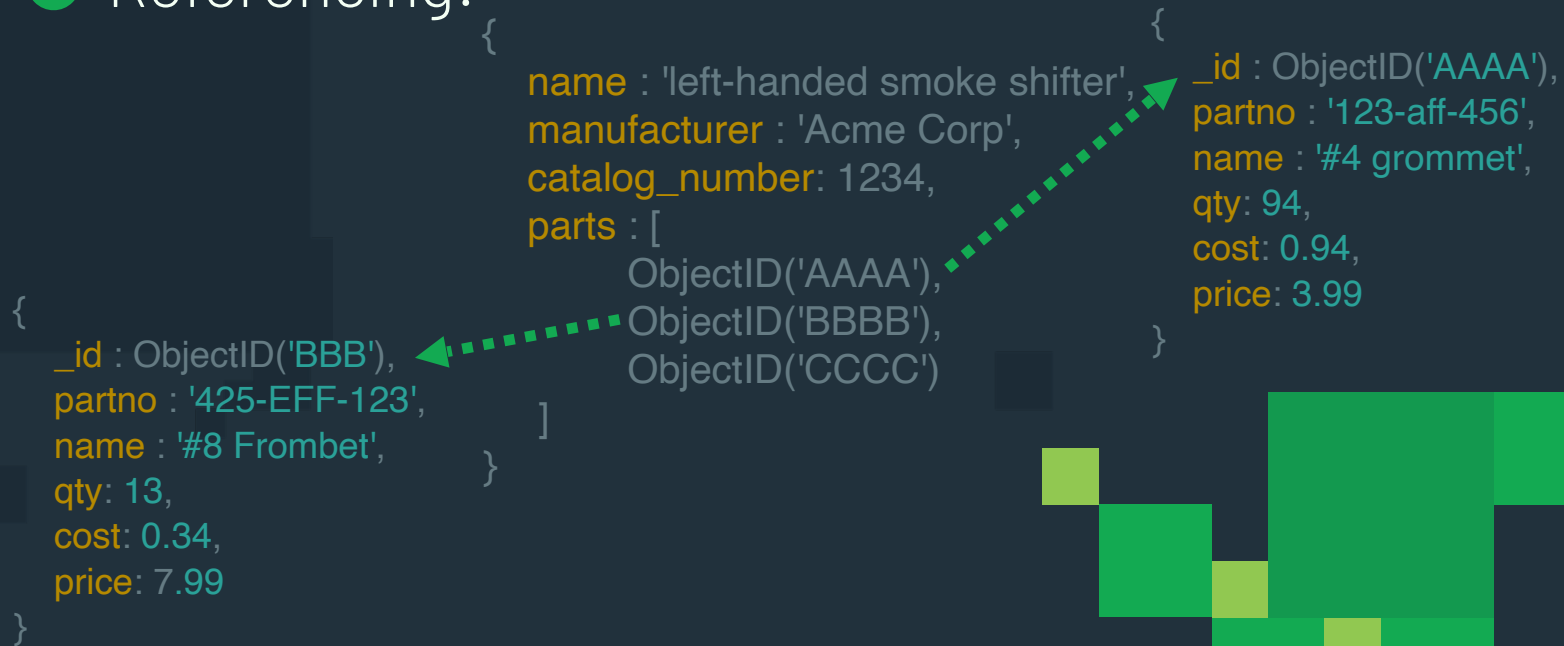
- Embedding vs. Referencing

- Embedding:

- ✓ Retrieve all data with a single query
 - ✓ Avoids expensive JOINS or \$lookup
 - ✓ Update all data with a single atomic operation
 - ✗ Large docs === more overhead
 - ✗ 16 MB Document size limit

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
 - Embedding
 - Referencing:



RECAP

- Relational vs. MongoDB Schema Design Approaches

- Embedding vs. Referencing

- Embedding

- Referencing:

- ✓ Smaller documents

- ✓ Less likely to reach 16 MB limit

- ✓ No duplication of data

- ✓ Infrequently accessed data not accessed on every query

RECAP

- Relational vs. MongoDB Schema Design Approaches

- Embedding vs. Referencing

- Embedding

- Referencing:

- ✓ Smaller documents
 - ✓ Less likely to reach 16 MB limit
 - ✓ No duplication of data
 - ✓ Infrequently accessed data not accessed on every query
 - ✗ Two queries or \$lookup required to retrieve all data

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs

```
{  
  _id: ObjectId("AAA"),  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com"  
}
```

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs
 - One to Few: Prefer embedding

```
{
  _id: ObjectId("AAA"),
  name: "Joe Karlsson",
  company: "MongoDB",
  twitter: "@JoeKarlsson1",
  website: "joekarlsson.com",
  addresses: [
    { street: "123 Sesame St", city: "Anytown", cc: "USA" },
    { street: "123 Avenue Q", city: "New York", cc: "USA" }
  ]
}
```



- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs
 - One to Few: Prefer embedding
 - One to Many: Prefer Referencing

```
{
  _id: ObjectId("123"),
  name: "left-handed smoke shifter",
  manufacturer: "Acme Corp",
  catalog_number: 1234,
  parts: [
    ObjectId("AAA"),
    ObjectId("BBB"),
    ObjectId("CCC"),
  ]
}
```

```
{
  _id: ObjectId("AAA"),
  partno: "123-ABC-456",
  name: "#4 grommet",
  qty: 94,
  cost: 0.54,
  price: 2.99,
}
```

RECAP

● Relational vs. MongoDB Schema Design Approaches

● Embedding vs. Referencing

● Types of Relationships

- One to One: Use Key-Value pairs
- One to Few: Prefer embedding
- One to Many: Prefer Referencing
- One to Squillions: Prefer Referencing

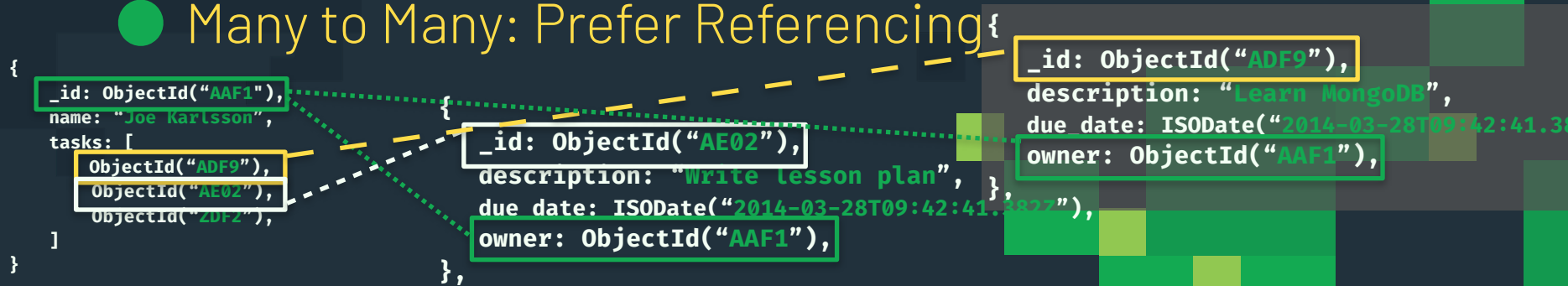
```
{
  _id: ObjectId("AAA"),
  name: "goofy.example.com",
  ipaddr: "127.66.66.66",
}
```

```
{
  _id: ObjectId("456"),
  time: ISODate("2014-03-28T09:42:41.382Z"),
  message: "Drive is hosed",
  host: ObjectId("AAA"),
}
```

```
{
  _id: ObjectId("123"),
  time: ISODate("2014-03-28T09:42:41.382Z"),
  message: "The CPU is on fire!!!",
  host: ObjectId("AAA"),
},
```

RECAP

- Relational vs. MongoDB Schema Design Approaches
- Embedding vs. Referencing
- Types of Relationships
 - One to One: Use Key-Value pairs
 - One to Few: Prefer embedding
 - One to Many: Prefer Referencing
 - One to Squillions: Prefer Referencing
 - Many to Many: Prefer Referencing



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to



MongoDB Schema Design Rules

RECAP

- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it
- Avoid JOINS and \$lookups if they can be avoided



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it
- Avoid JOINS and \$lookups if they can be avoided
- Arrays should not grow without bound



MongoDB Schema Design Rules



- Favor embedding unless there is a compelling reason not to
- Needing to access an object on its own is a compelling reason not to embed it
- Avoid JOINS and \$lookups if they can be avoided
- Arrays should not grow without bound
- How you model your data depends – entirely – on your particular application's data access patterns



Questions?

What's Next?

MongoDB Community

- university.mongodb.com
- Recommended Course:
M320: Data Modeling



<http://bit.ly/fromSQLToNoSQL>



Want \$100 in FREE MongoDB Atlas credits?

Use code **JoeK100**

<http://bit.ly/FreeAtlasCredits>

@JoeKarlsson1 

Additional Resources



- Modeling: <https://university.mongodb.com/courses/M320/about>
- 6 Rules of Thumb for MongoDB Schema Design:
Part 1: <https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1>
- Data Model Design: <https://docs.mongodb.com/manual/core/data-model-design/>
- Data Model Examples and Patterns: <https://docs.mongodb.com/manual/applications/data-models/>
- Building with Patterns: A Summary: <https://www.mongodb.com/blog/post/building-with-patterns-a-summary>

@JoeKarlsson1

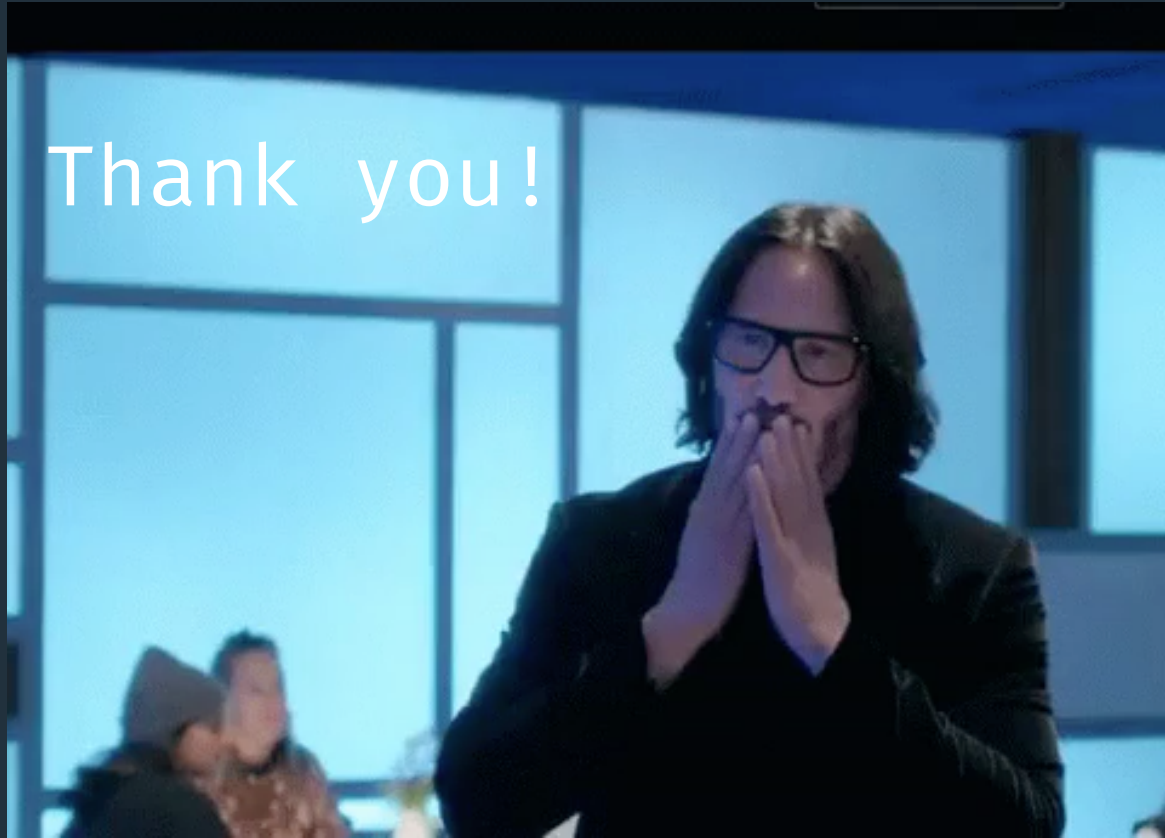




```
{  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  title: [  
    "Developer Advocate",  
    "Software Engineer"  
  ],  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com",  
  opinions: "my own",  
  links: "bit.ly/IoTKittyBox"  
}
```



Thank you!





```
{  
  name: "Joe Karlsson",  
  company: "MongoDB",  
  title: [  
    "Developer Advocate",  
    "Software Engineer"  
  ],  
  twitter: "@JoeKarlsson1",  
  twitch: "joe_karlsson",  
  tiktok: "joekarlsson",  
  website: "joekarlsson.com",  
  opinions: "my own",  
  links: "bit.ly/IoTKittyBox"  
}
```



