MongoDB Database Design Patterns

Török Mihály

Instructor's bio...

Török Mihály

e-mail: mihaly.torok@gmail.com

LinkedIn: https://www.linkedin.com/in/mihaly-torok-b3433366

AgileHub: https://agilehub.ro/

Slack PeakIT: https://peakit004.slack.com/
#workshop-23-oct-10_30am-mihaly-torok-mongodb-database-design-patterns

Expertise: Database development and query tuning – SQL, MongoDB

DIAMOND

SIEMENS

nagarro

Atos

Trimble.

Raiffeisen BANK
Banking așa cum trebuie

SILVER

accenture

endava

cegeka

PARTNERS

coresi

<ScriuCod/>

Codette

CODECAMP_

MEDIA PARTNERS

CAPITAL

BizBrasov
Stirile care conteaza in Brasov

start#up

ROCK FM
It Rocks!

Valorile AgileHub

INDEPENDENȚA

- Trainerii și participanții vin pe cont propriu, fără a promova vreo firmă
- Nimeni nu reprezentă interesele nici unei firme
- La începutul cursului, când ne prezentăm, spunem care e rolul și experiența noastră, fără a specifica firma la care lucrăm

EDUCAȚIA GRATUITĂ

- Cursurile sunt gratuite pentru toți participanții

VOLUNTARIATUL

- Toți trainerii sunt voluntari

ÎMPĂRTĂȘIREA EXPERIENȚEI PROPRII

- Majoritatea formatorilor **NU** sunt traineri profesioniști
- Formatorii lucrează în IT și au multă experiență practică în domeniul pe care îl predau

Participant Introduction

- Name
- Course relevant experience

What to expect

PEAK IT 004

- PowerPoint slides
- Participation in class discussions
- Instructor explanations

The presentation is shared on **GitHub:**

https://github.com/MihalyTorok/PeakIT004-mongodb-design-patterns.git

AgileHub

MongoDB Database Design Patterns

Agenda

1. MongoDB – A document database
2. Data modelling in MongoDB
3. Data access patterns
4. MongoDB database schema patterns
5. Summary

1. MongoDB – A document database

What is NoSQL?

- Non relational database – “non SQL” or “not only SQL”
- Stores relationship information in a different way
- Nesting of related data in a single data structure is allowed
- Supports data in any shape
  - Structured
  - Semi structured
  - Polymorphic
- The stored document’s structure is close to the object used in the application
- Easy iterations in the Agile application development process

1. MongoDB – A document database

Types of NoSQL databases

- Document database
  - The documents are stored in JSON
  - They can scale-out to improve performance of big data stores
- Key-value database
  - Easy querying based on the key data only
  - Common use case: user preference, caching – Redis, DynamoDB
- Wide-column stores
  - Stores data in tables, rows and dynamic columns
  - Commonly used for IoT – Cassandra, HBase
- Graph database
  - Data is stored in nodes and edges
  - Used for social networking, fraud detection and recommendation engines

1. MongoDB – A document database

- Intuitive data model
  - Documents are closely mapped to the objects from the code
- Flexible schema
  - Easy to implement new features which need schema change
- Universal, implemented using JSON
  - Language independent and human readable
- Query data in any way
  - The MongoDB Query API provides a powerful query language
  - The queries are supported by indexes
- Distributed and globally scalable
  - Documents allow distribution across multiple servers
  - Native sharding and live re-sharding

2. Data modelling in MongoDB

Benefits of MongoDB

- Flexible schema

Decisions to take when modelling the data

- Document structure
  - Embedded data
  - References
- Atomicity of write operations
  - Single document atomicity
  - Multi-document transactions

    MongoDB 4.2 in replica sets, 4.4 in sharded clusters
- Data use and performance

  Data access patterns

Movie
title
director
release_location_id FK

Movie:

```
{
   title: "Star Wars",
   director: "George Lucas“,
   releases: [
      {
         location: “US”,
         date: ISODate("1977-05-20T01:00:00+01:00")
      }, …
   ]
}
```

3. Data access patterns

- You need to design a new application
- Your application has strong requirements
- It should be performant
- How do you design your database?
- Choosing the wrong schema will cause bad performance

Movie
title
director
release_location_id FK

Movie:

```
{
   title: "Star Wars",
   director: "George Lucas“,
   releases: [
      {
         location: “US”,
         date: ISODate("1977-05-20T01:00:00+01:00")
      }, …
   ]
}
```

3. Data access patterns

- Consider the use cases implemented by your application
  - Which information are needed together
  - The shape of the queries
- Consider atomicity requirements
  - Is the single document atomicity enough?
- Consider performance
  - Read operation performance
  - Number of database accesses for a single user view

Project	
Id:	15
Name	Website development

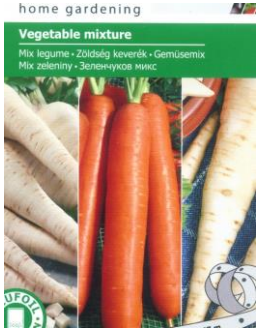
Stages		
Id	**Name**	**Hours**
1	Planning	40
2	POC	80
3	Implementation	160

4. MongoDB database schema patterns

Use case - Data access pattern

Use Case:

- A plant shop needs an application
- It sells flowers, trees and vegetables
- Each plant type has different properties



General use case:

- The modelled objects are similar but not identical
- They have more similarities than differences

Polymorphic Pattern

Implementation:

- A single collection for all plants
- The documents in the collection have slightly different properties

Pros:

- All plants are saved together avoiding joins.

Other use cases:

- Single view applications
- Content management
- Mobile applications
- Product catalog

Plant:

```
{
  _id : 1234
  name : “Tulip”,
  color : “red”,
  height : 50
},
{
  _id : 1235
  name : “Apple tree”,
  height : 300
  planting_info : “Late fall and early spring”
},
{
  _id : 1236
  name : “Carrot”,
  sowing : “From April to early July”
}
```

Use case - Data access pattern

Use Case:

- We have big documents with many similar fields but there is a subset of fields that share common characteristics and we want to sort or query on that subset of fields
- The fields we need to sort on are only found in a small subset of documents
- Both of the above conditions are met within the documents.

Movie
title
director
release_US
release_France
release_Italy
release_UK
[…]

Attribute Pattern

Implementation:

- Implement an array with the attributes you want to query
- Implement an index on the array property

Pros: Fewer indexes are needed.

Cons: Limitations of the multikey indexes.

Document:

```
{
   title: "Star Wars",
   director: "George Lucas“,
   releases: [
      {
         location: “US”,
         date: ISODate("1977-05-20T01:00:00+01:00")
      },
      {
         location: “France”,
         date: ISODate(" 1977-10-19T01:00:00+01:00 ")
      }, ….
   ]
…
}
```

Index:

{releases. location : 1, releases. date : 1}

4. MongoDB database schema patterns

Use case - Data access pattern

Use Case:

- Data coming in as a stream over a period of time (time series data)
- We have a sensor taking the temperature and saving it to the database every minute.

Measurement
sensor_id
timestamp
temperature

4. MongoDB database schema patterns

Bucket Pattern

Implementation:

- Bucket the date at the smallest interested timespan
- Implement pre-computed aggregations

Pros:

- Fewer documents in the collection
- Smaller indexes
- Usage of the aggregated data
- Archive possibility

Document:

```
{
   sensor_id: 12345,
   start_date: ISODate("2019-01-31T10:00:00.000Z"),
   end_date: ISODate("2019-01-31T10:59:59.000Z"),
   measurements: [
     {
     timestamp: ISODate("2019-01-31T10:00:00.000Z"),
     temperature: 40
     }, {
     timestamp: ISODate("2019-01-31T10:01:00.000Z"),
     temperature: 40
     },
     ...
  ],
  transaction_count: 42,
  sum_temperature: 2413
}
```

4. MongoDB database schema patterns

Use case - Data access pattern

Use Case:

- We design a book store application
- It's required to store the customers of each book
- Few books are best seller having many customers

Assumptions:

- Only a low percentage of the documents have the different behavior
- Changing the schema could affect performance

Outlier Pattern

Implementation:

- Save only a limited number of ids into the array
- Add a flag to save that more data exists
- Save the remaining array in another document with a *parent_id* property

Pros:

- Usual data is queried with high performance

Cons:

- Outlier document needs aggregation at the application level

Book:

```
{
  _id : 1234
  title : “Harry Potter and the
          Philosopher's Stone”,
  author : “J.K. Rowling”,
  customers : ["user01", "user02", ... , "user99"],
  has_extras : true,
  ...
}
{
  _id : 1235
  parent_id : 1234,
  customers : ["user100", ..., "user1M", ...],
}
```

Use case - Data access pattern

Use Case:

- We have screening information of movies
- How many viewers watched the latest blockbuster movie?

Other use cases:

- Time series data
- Product catalogs
- Single view applications
- Event sourcing

Screening
theater
location
movie_title
num_viewers
revenue

Computed Pattern

Implementation:

- Compute the data together with any update of the source data and store it
- Compute the data at defined intervals
- Update the computed data based on update timestamp
- Implement a queue for computations which need to be done

Pros:

- Fewer CPU cycles
- Significantly lower number of reads

Movie:

```
{
  title : ”Jack Ryan: Shadow Recruit”
  total_viewers : 2800
  total_revenue : NumberDecimal(“33550.00”)
}
```

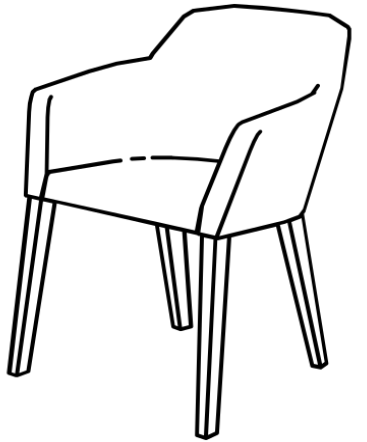
Use case - Data access pattern

Use Case:

- An e-commerce site has reviews for products
- The product document's size increases because of many reviews
- Not all reviews are displayed to the user
- The working set's size exceeds RAM

Other use case:

The document has many properties but only few are usually used.



User55 1 day ago

Functional and beautiful

I love it.

4. MongoDB database schema patterns

Subset Pattern

Implementation:

- The Product collection will have the latest 10 reviews
- The most used data is saved in the Product document
- Create another collection for Reviews.

Cons:

- If all reviews need to be loaded then additional trips to the database are needed.

Product:

```
{
  _id : 1234
  name : “Chair”,
  last_10_reviews : [
        ….
  ]
}
```

Review:

```
{
   rating : 4,
   title : “Functional and beautiful”,
   content : “I love it”,
   user : “User55”
   date : ISODate("2019-02-10T11:00:00.000Z")
   product_id : 1234
 }
```

Use case - Data access pattern

Use Case:

- We implement an ordering application
- Each order is given by a customer
- A customer could have many orders

Assumptions:

- The use case involves a one to many relationship
- Part of the document from the one side is needed often

Extended Reference Pattern

Implementation:

- Copy the most queried properties from the *Customer* collection as subdocument in the *Order* collection
- Usable for N – 1 relationship
- The copied data should not change often

Pros:

- Reduces the JOINs needed

Cons:

- Data duplication which needs handling

Order:

```
{
  _id : 1234,
  ordeder_date :
      ISODate("2019-02-10T11:00:00.000Z"),
  customer : {
    customer_id : 10,
    name : "Big Customer"
  },
  products : [ … ],
  value : NumberDecimal(“100.15”)
}
```

Customer:

```
{
  _id : 10,
  name : "Big Customer",
  address : "Main street 10"
}
```

Use case - Data access pattern

Use Case 1:

- We have a city with approximately 39000 residents
- The population is changing daily
- We need the city's population for the city's planning strategy

Use Case 2:

- We need the number of views a website had.

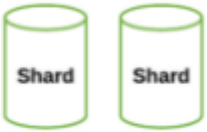
Approximation Pattern

Implementation:

- Choose an approximation factor. E.g.: 100
- Write to the database only that occurrence of the event increasing with the factor.
  - Use a counter
  - Use a random number generation

Pros: Less writes to the database.

Cons: Implementation needed at the application level.



*

- Register budget changes
- Decision for new public transport
- Other
- Population changes

* Source: Approximation Pattern - Daniel Coupal and Ken W. Alger

4. MongoDB database schema patterns

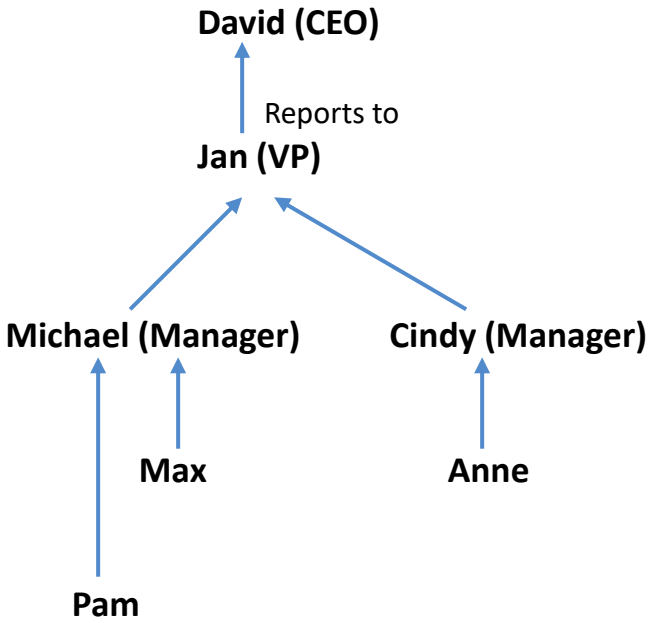
Use case - Data access pattern

Use Case:

- Implement an employee management system
- Model the reporting chain from an employee to CEO

Other use case:

- Product catalog where products belong to hierarchical categories
- Genealogy management system
- Any hierarchy modeling application



Tree and Graph Pattern

Implementation:

- Add an array property having the report chain
- If needed, add a property for direct report
- Define the needed indexes

Pro:

- Easy to query the reporting chain

Cons:

- Greater difficulties for updates

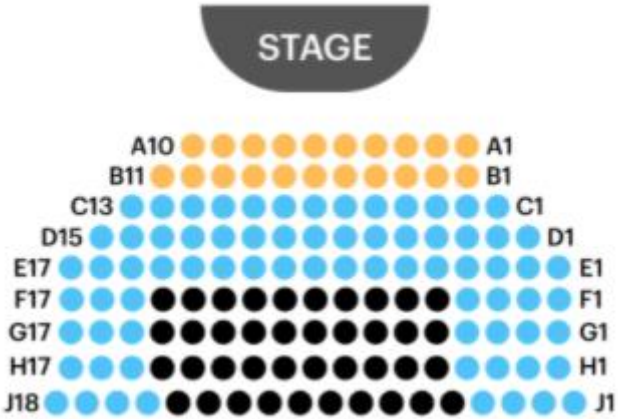
Employee:

```
{
  _id : 7
  name : “Anne”,
  direct_reports_to : “Cindy”,
  reports_to : [
      “Cindy”,
      “Jan”,
      “David”
  ]
}
```

Use case - Data access pattern

Use Case:

- A theatre seat reservation system is needed
- The rows have different number of seats
- Some seats are accessible



Other use case:

- A reservation system where the object is reserved on a daily bases.

Preallocation Pattern

Implementation:

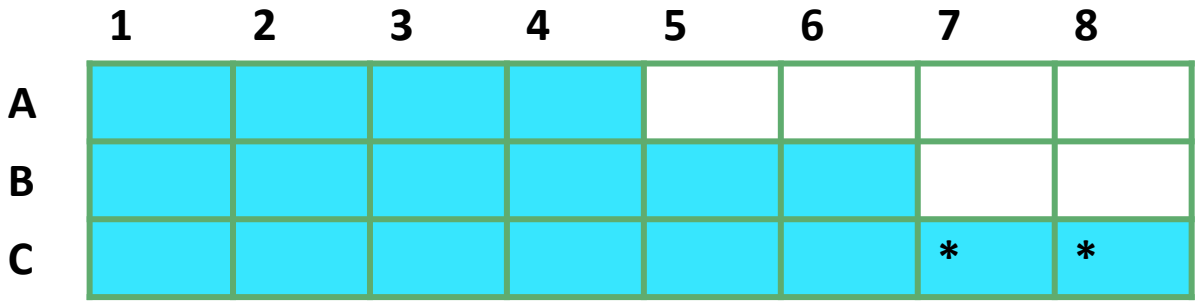
- Two dimensional array of objects
- If the seat doesn't exist then the cell is null
- Flag for accessible seats

Pros:

- Easy finding of seats

Cons:

- More memory is used



Preallocation Pattern

Use in the early versions of MongoDB

- It was helpful when *MMAPv1* storage engine was used *(prior to MongoDB version 3.2)*
- Pre allocates the necessary memory for each document
- Used to prevent document moving during updates
- *MMAPv1* storage engine is deprecated
- *WiredTigre* storage engine does not need this approach.

Article:

```
{
  _id : 1234
  title : “The WiredTigre Storage Engine”,
  author : “MongoDB”,
  content : “The engine’s description”,
  ...
  memory_allocation : “**********”
}
```

Use case - Data access pattern

Use Case:

- We need to estimate a project
- Few revisions of the same estimation is needed
- Each revision should be saved for history

Assumptions:

- Each document has few revisions.
- Not many documents need versioning.
- The most current version of the document is mostly used.

Other use cases:

- Insurance versions
- Healthcare industry applications

Document Versioning Pattern

Implementation:

- Add a version number property to the document
- Save the history and the current version in different collections

Pros:

- Queries for the latest version are performant

CurrentEstimation:

```
{
  project_name : "Bridge construction",
  estimation_date : ISODate("2019-02-10T11:00:00.000Z"),
  revision : 2,
  breakdown : [ Material, Labor ],
  value : NumberDecimal(“ 110.00”)
}
```

EstimationHistory:

```
{
  project_name : "Bridge construction",
  estimation_date : ISODate("2019-01-31T10:00:00.000Z"),
  revision : 1,
  breakdown : [ Material ],
  value : NumberDecimal(“ 100.00”)
}
```

Use case - Data access pattern

Use Case:

- A contact management application saves contact information
- At the beginning only home and work phone numbers were saved
- Now it is needed to support different contact information
- The attribute pattern is implemented to save different contact details
- How to handle the schema change?

Contact:

```
{
  _id : 1234
  name : “John”,
  home : “+40(123)456789”,
  work : “+40(456)789123”
}
```

Contact:

```
{
  _id : 1235
  name : “Cindy”,
  contact_details : [
    { work : “+40(789)456123” },
    { skype : “cindy456” }
  ]
}
```

Schema Versioning Pattern

Implementation:

- Add a *schema_version* property to the documents
- Save the new documents using the latest schema
- Decide how to upgrade the older documents' schema

Cons:

- In some cases doubled indexes are needed during the migration

Contact:

```
{
  _id : 1234
  name : “John”,
  home : “+40(123)456789”,
  work : “+40(456)789123”
},
{
  _id : 1235
  name : “Cindy”,
  contact_details : [
    { work : “+40(789)456123” },
    { skype : “cindy456” }
  ]
  schema_version : 2
}
```

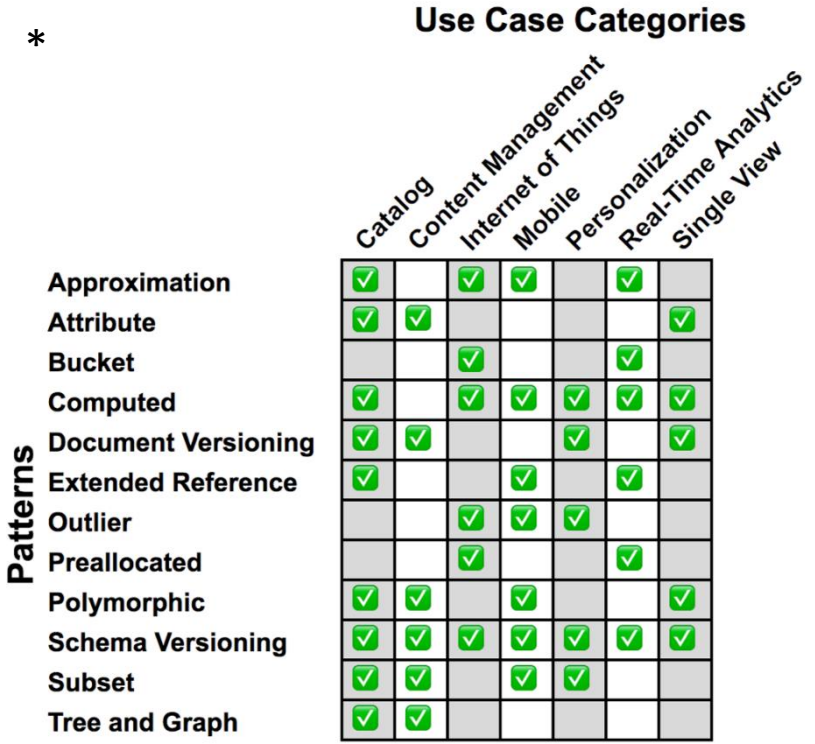
5. Summary

- You should consider patterns any time when you design an application
- Each new feature implemented could need database schema changes
- Database design patterns can be implemented any time during the application's lifecycle

Examples of which type of application could benefit of which pattern:

- These are not strong rules

*



* Source: Building with Patterns - Daniel Coupal and Ken W. Alger

Questions

1. MongoDB – A document database
2. Data modelling in MongoDB
3. Data access patterns
4. MongoDB database schema patterns
5. Summary

Slack PeakIT: https://peakit004.slack.com/
#workshop-23-oct-10_30am-mihaly-torok-mongodb-database-design-patterns

GitHub: https://github.com/MihalyTorok/PeakIT004-mongodb-design-patterns.git

MongoDB Database Design Patterns - More Info

PEAK IT 004

[What is a Document Database? – MongoDB]
[Data Modeling Introduction – MongoDB]
[MongoDB Schema Design Best Practices – Joe Karlsson]
[Building with Patterns - Daniel Coupal and Ken W. Alger – MongoDB]
[M320 – Data Modeling – MongoDB University]
[Multikey Indexes – MongoDB]
[$lookup (aggregation) Considerations – MongoDB]
[$graphLookup (aggregation) - MongoDB]
[Many to many relation]

AgileHub

Feedback

PEAK IT 004

http://bit.ly/peakit004-feedback

- Completați acum
- Durează 2-3 minute
- Feedback anonim - pentru formator si AgileHub

23 oct, 10:30 - Mihály Török- "MongoDB Database Design Patterns"

AgileHub