TOPIC G01

UNDERSTANDING NOSQL DATABASE

UNDERSTAND NOSQL DATABASE

CONTENT

- Explain advantages and disadvantages of relational database
- Explain NoSQL and type of NoSQL databases
- Use MongoDB
- Explain NoSQL data modeling

Reference:

1. Dan Sullian, Advanced NoSQL for Data Science (2017), LinkedIn Course
2. MongoDB documentation, https://docs.mongodb.com/manual/core/data-modeling-introduction/

RELATIONAL DATABASE - RECAP

- Records are organized into tables.
- Rows of tables are identified by keys.
- Data spans multiple tables, which are linked by a join operation.
- Transactions are ACID-compliant.
- Data models are normalized.

Atomicity

- Each transaction is 'all or nothing'.
- If part of the transaction fails, then the entire transaction fails.
- The database state is left unchanged.

Consistency

- Any transaction will bring the database from one valid state to another.
- It satisfies all the rules and constraints that database.

Isolation

- The concurrent execution of transactions results in a system state that would be obtained if the transactions were executed serially.

Durability

- Once transaction has been committed, it will remain so.

RELATIONAL DATABASE

Advantages

- Comprehensive querying
- Normalized data
- Widely supported

Disadvantages

- Fixed schema
- Costly to join
- Limited data structures
- Difficult to scale

RELATIONAL DATABASE – MINIMIZE THE CONS

Denormalization

- Expand the number of columns in a single table

Partition

- Breaking up the database and store pieces of database in different servers

Replication

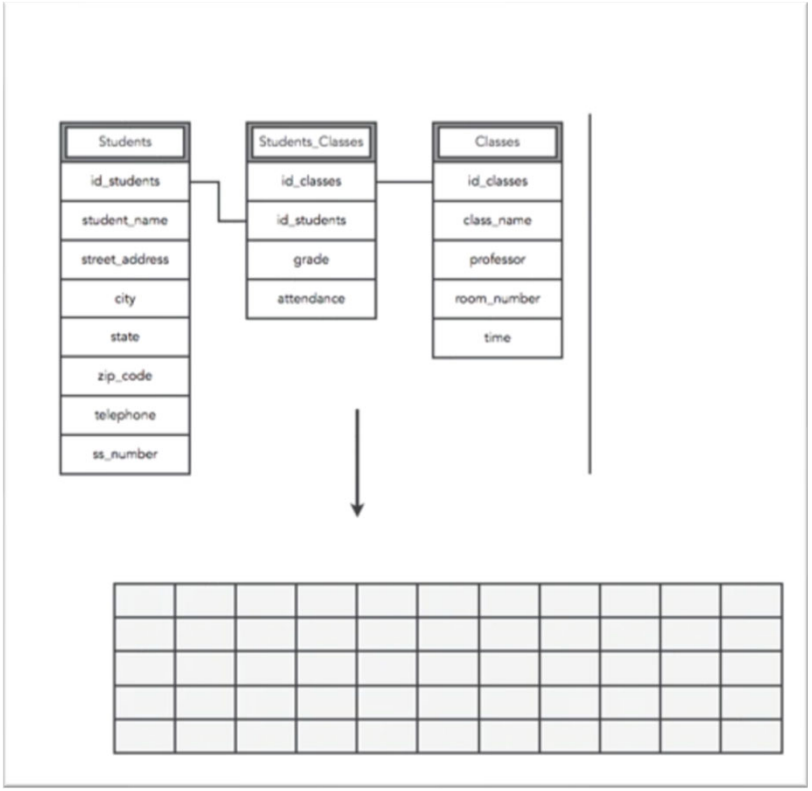
- Making copies of the database, store the copies in different servers

GI

RELATIONAL DATABASE – MINIMIZE THE CONS

Denormalization

- Avoid joins
- Expand the number of columns
- Design table to include related data
- Query a single table
- **Improve read performance**



WHAT IS NOSQL DATABASE?

- Not using SQL
- Flexible Schema
  - No fixed-table definition
  - Fields are not standardized between records
- Support complex data structures
  - Nested values are common in NoSQL database
- Design with 'partition' in mind (Scalability)

NOSQL DATABASE TYPE

Key-value (eg Amazon S3)

- Based on key-value or dictionary data structures

Document (eg CouchDB, MongoDB)

- Multiple key-value pairs in a document

Wide Column (eg Cassandar, Google Bigtable)

- Organized in tables, rows and columns
- Columns are not fixed, can be different between rows and can be changed

Graph (Neo4j)

- Network of connected entities
- Entities are linked by edges

SHORT QUIZ

- Which type of NoSQL database is based on sets of nodes and edges between nodes?

(Choose: Document, Wide-Column, Graph)

- Which is not an advantage of NoSQL database?

(Choose: Support 'Joins', Support for large dataset, Support distributed database)

mongodb.com

mongoDB.

The database for modern applications

MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era. No database makes you more productive.

Try MongoDB free in the cloud!

Get Started

DOCUMENT DATA MODEL – DOCUMENT (~ RECORD)

Documents

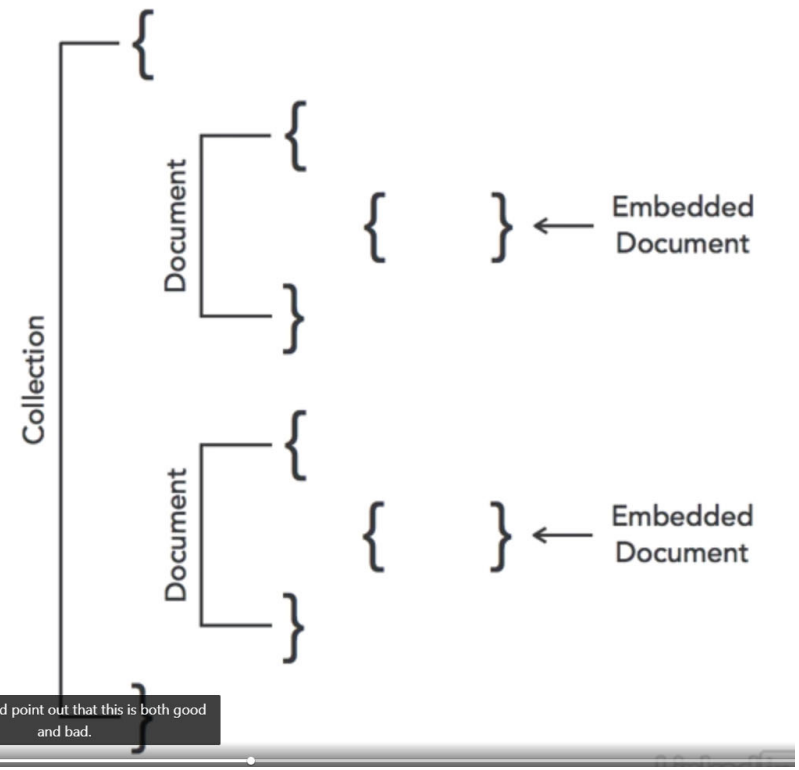
- Set of key-values
- Optional indexes
- Attributes vary across documents.
- Values may be complex structures.

```
{
  orderID: 165345,

  customer:{ FullName:'Morgan James',
             Address: '321 First St',
             City: 'Portland'},

  orderItem:{ SKU: 5638126,
              Desc: 'Small Pen'},

  orderItem:{ SKU: 5639444,
              Desc: 'Notepad'},

  orderItem:{ SKU: 5627734,
              Desc: 'Scissors'}

}
```

DOCUMENT DATA MODEL – COLLECTION (~ TABLE)

Collections

- Set of 0 or more documents
- Each document has a unique ID.
- Schema inferred
- New attributes added by creating a document with attributes



Now, I should point out that this is both good and bad.

RUN MONGODB COMMUNITY EDITION

Set up Mongo Environment

- Create \data\db

Start MongoDB

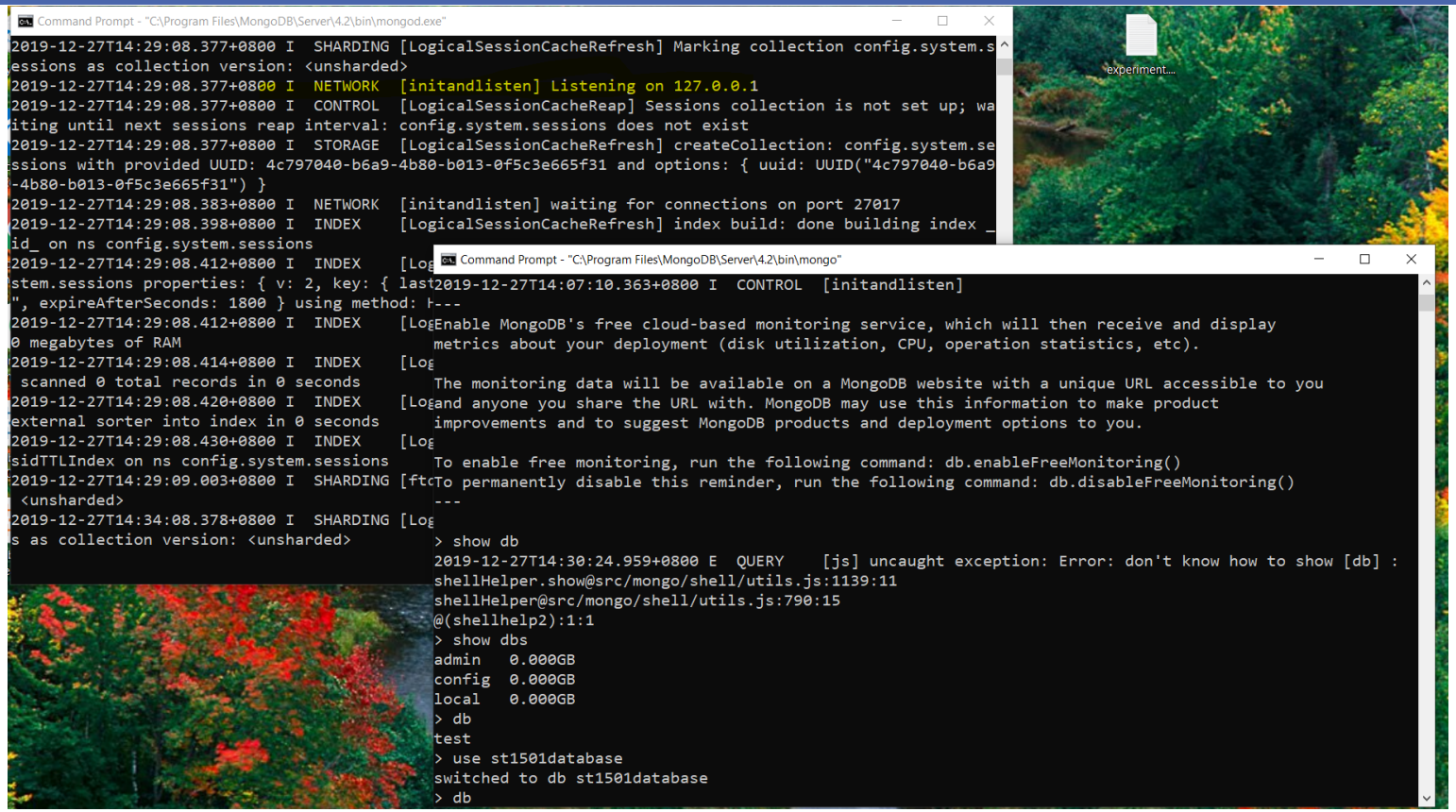
- Run **mongod.exe**
- If not using default dbpath, specify --dbpath actualpath

Connect to MongoDB via Mongo Shell

- Run **mongo.exe**

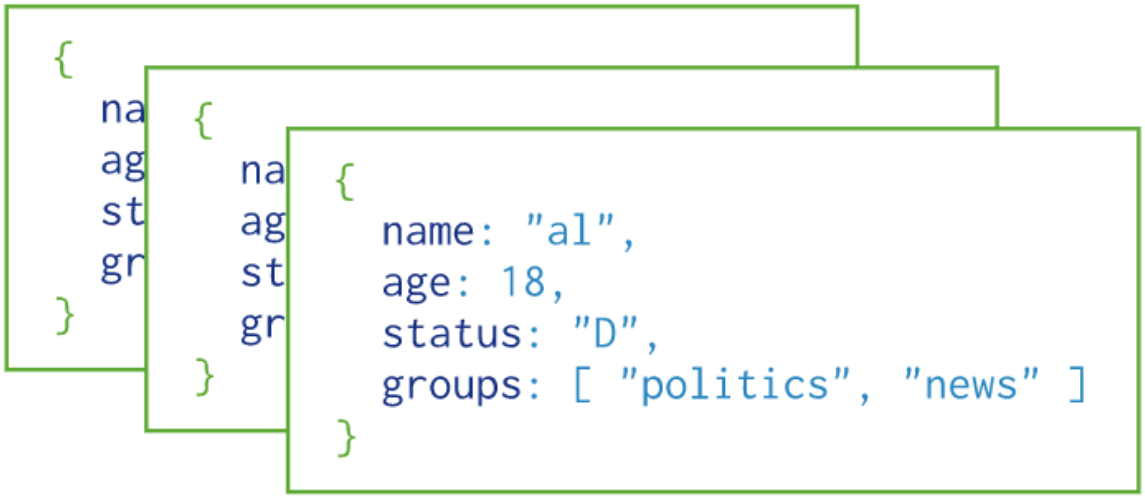
GI

RUN MONGODB COMMUNITY EDITION



MONGODB: DATABASE & COLLECTION

- MongoDB stores JSON **documents** in **collection**, collections in **database**.



Collection

- To display the database you are using, type db

```
db
```

```
> db
test
```

- To list the available databases, type 'show dbs'

```
show dbs
```

```
> show dbs
admin          0.000GB
local          0.000GB
myNewDatabase  0.000GB
```

Using NoSQL (MongoDB)

- To switch databases, issue the use command
- If the database does not exist, it will be created

```
use mymongodatabase
```

```
> use mymongodatabase
switched to db mymongodatabase
```

- To see the list of collections in the current database, use the 'show collections' command:

```
show collections
```

```
> use local
switched to db local
> show collections
startup_log
```

MONGODB
CRUD OPERATIONS

Using NoSQL (MongoDB)

- Create or insert operations, add new documents to a collection.
- If the collection does not currently exist, insert operations will create the collection.
- MongoDB provides the following methods to insert documents into a collection.

  - **db.collection.insertOne()**
  - **db.collection.insertMany()**

Using NoSQL (MongoDB)

```
db.users.insertOne(
  {
    name: "sue",
    age: 26,
    status: "pending"
  }
)
```

collection

field: value

field: value

field: value

document

```
> db.users.insertOne( {name: "sue",age: 26,status: "pending"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5a02d3541c1652812ae9874c")
}
```

Using NoSQL (MongoDB)

```
db.users.insertOne({name: "sue",age: 26,status: "pending"})
```

```
> db.users.insertMany([{name: "sue",age: 26,status: "pending"},{name: "zan",ag
e: 18,status: "pending"},{name: "don",age: 20,status: "pending"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5a03c2e11c1652812ae9874f"),
                ObjectId("5a03c2e11c1652812ae98750"),
                ObjectId("5a03c2e11c1652812ae98751")
        ]
}
```

Using NoSQL (MongoDB)

```
db.users.insertMany([{name: "sue",age: 26,status: "pending"},{name: "zan",age: 18,status: "pending"},{name: "don",age: 20,status: "pending"}])
```

```
> db.users.insertMany([{name: "sue",age: 26,status: "pending"},{name: "zan",ag
e: 18,status: "pending"},{name: "don",age: 20,status: "pending"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5a03c2e11c1652812ae9874f"),
                ObjectId("5a03c2e11c1652812ae98750"),
                ObjectId("5a03c2e11c1652812ae98751")
        ]
}
```

Using NoSQL (MongoDB)

- Read operations retrieves documents from a collection; i.e. queries a collection for documents.
- MongoDB provides the following methods to read documents from a collection
  - **db.collection.find()**
- You can specify query filters or criteria that identify the documents to return

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

Using NoSQL (MongoDB)

```
db.user.find({gender: "female"}, {name:1}).limit(5)
```

```
> db.user.find({gender: "female"}, {name:1}).limit(5)
{ "_id" : ObjectId("5b42144ad2963edac48e1997"), "name" : "Dora" }
{ "_id" : ObjectId("5b42154bd2963edac48e1999"), "name" : "Mary" }
```

```
db.users.find({age: {$gt:18}}, {name:1,address:1}).limit(5)
```

```
> db.users.find({age: {$gt:18}}, {name:1,address:1}).limit(5)
{ "_id" : ObjectId("5a02d3541c1652812ae9874c"), "name" : "sue" }
{ "_id" : ObjectId("5a03bf841c1652812ae9874e"), "name" : "ann" }
```

Using NoSQL (MongoDB)

- Update operations modify existing documents in a collection.
- MongoDB provides the following methods to update documents of a collection

  - **db.collection.updateOne()**
  - **db.collection.updateMany()**
  - **db.collection.replaceOne()**

- In MongoDB, update operations target a single collection.
- All write operations in MongoDB are atomic on the level of a single document.

- You can specify criteria, or filters, that identify the documents to update.
- These filters use the same syntax as read operations.

Using NoSQL (MongoDB)

```
db.users.updateMany(                          ← collection
   { age: { $lt: 18 } },                      ← update filter
   { $set: { status: "reject" } }             ← update action
)
```

Using NoSQL (MongoDB)

- Delete operations remove documents from a collection.
- MongoDB provides the following methods to delete documents of a collection:
  - **db.collection.deleteOne()**
  - **db.collection.deleteMany()**
- In MongoDB, delete operations target a single collection.
- All write operations in MongoDB are atomic on the level of a single document.

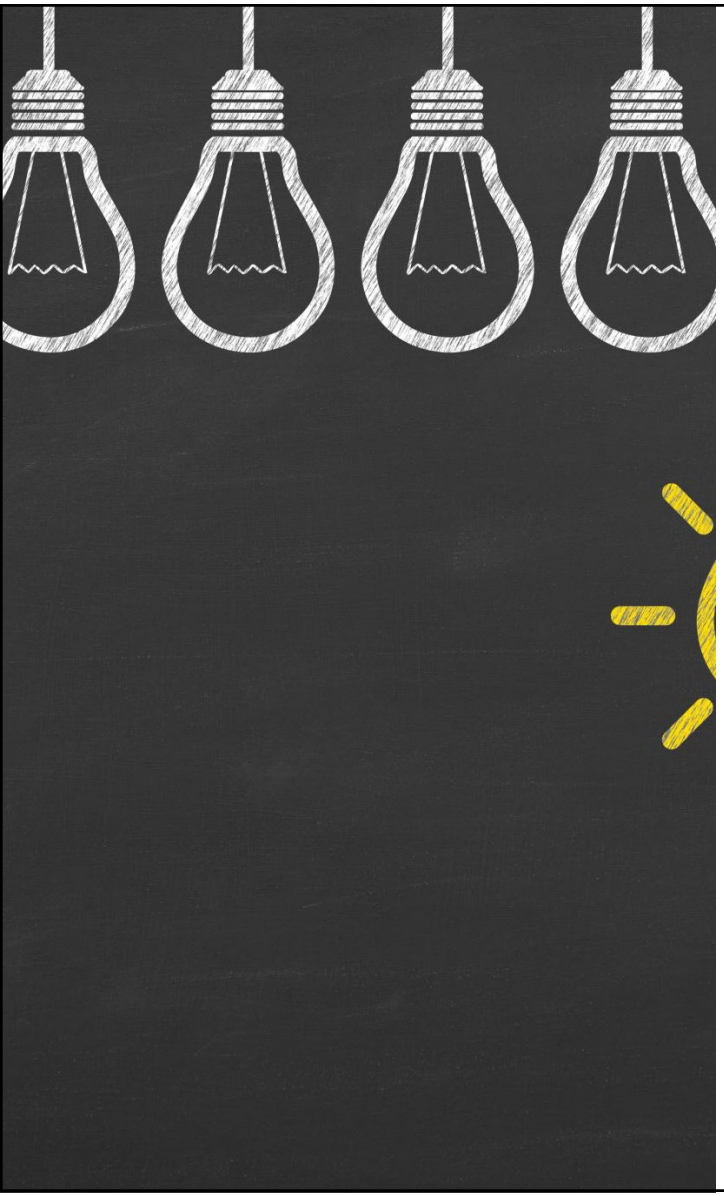
- You can specify criteria, or filters, that identify the documents to remove
- These filters use the same syntax as read operations.

```
db.users.deleteMany(          ← collection
   { status: "reject" }       ← delete filter
)
```

```
db.users.deleteOne({name:"sue"})
```

```
> db.users.deleteOne({name:"sue"})
{ "acknowledged" : true, "deletedCount" : 1 }
```

NOSQL DATA MODELING



MONGODB DATA MODELING

- MongoDB's document need not have the same schema.
- The document structure can be updated, such as add new fields, remove existing fields, or change the field values to a new type.
- Each field can be a complex structure.

- Data models for MongoDB applications revolves around the structure of documents.
  - Embedded Data Model (Denormalized)
  - Reference Data Model (Normalized)

MONGODB DATA MODELLING : EMBEDDED DATA MODEL

- Store related piece of information in the same document, result in less quires.

- Appropriate:
  - we have "contains" relationships between entities.
  - we have one-to-many relationships between entities. In these relationships the "many" or child documents always appear with or are viewed in the context of the "one" or parent documents.

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

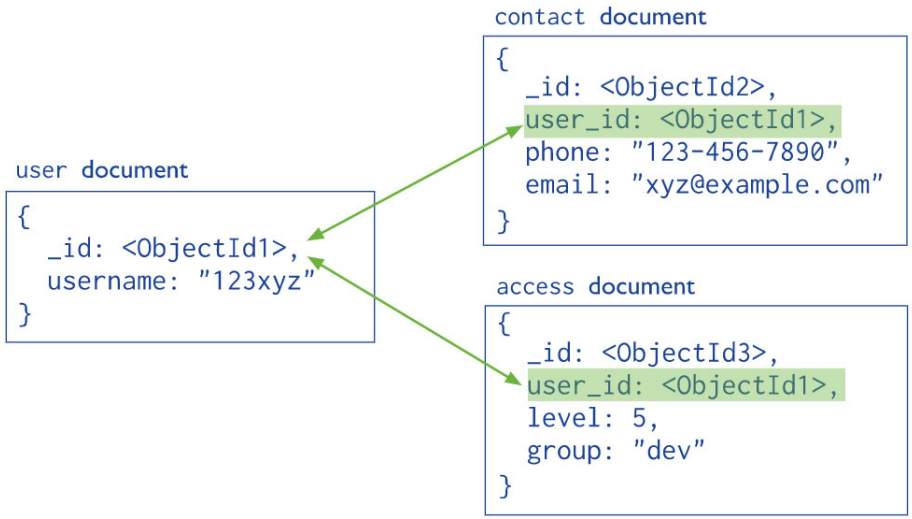
Embedded sub-document

GI

MONGODB DATA MODELLING : REFERENCE DATA MODEL

- Store the relationships between data by including links or references from one document to another.

- Appropriate:
  - when embedding would result in duplication of data but would not provide read performance advantages to outweigh the implications of the duplication.
  - to represent more complex many-to-many relationships.
  - to model large hierarchical data sets.



GI

CASE: ONE-TO-ONE MAPPING (EMBEDED)

```
// patron document
{
   _id: "joe",
   name: "Joe Bookreader"
}

// address document
{
   patron_id: "joe", // reference to patron document
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

```
{
   _id: "joe",
   name: "Joe Bookreader",
   address: {
            street: "123 Fake Street",
            city: "Faketon",
            state: "MA",
            zip: "12345"
         }
}
```

CASE: ONE-TO-MANY MAPPING (EMBEDED)

```
{
   _id: "joe",
   name: "Joe Bookreader"
}

{
   patron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}

{
   patron_id: "joe",
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
}
```

```
{
   _id: "joe",
   name: "Joe Bookreader",
   addresses: [
               {
                 street: "123 Fake Street",
                 city: "Faketon",
                 state: "MA",
                 zip: "12345"
               },
               {
                 street: "1 Some Other Street",
                 city: "Boston",
                 state: "MA",
                 zip: "12345"
               }
             ]
 }
```

CASE: ONE-TO-MANY

copy

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

{
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}
```

CASE: ONE-TO-MANY

PROPOSED SOLUTION A

copy

```
{
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA",
   books: [123456789, 234567890, ...]
}

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

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English"
}
```

CASE: ONE-TO-MANY

PROPOSED SOLUTION B

copy

```
{
   _id: "oreilly",
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA"
}

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

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher_id: "oreilly"
}
```

MONGODB DATA MODELING: EMBEDDED VS REFERENCE

- Data that is reference together should be embedded.
- Dependent entities can be embedded.
- Entities with one-to-one relationship can be embedded.
- Entities updated at the same time can be embedded.
- Independent entities should be reference.

THE END