CSCU9YQ - NoSQL Databases
Lecture 7.a: Data Models in MongoDB

Gabriela Ochoa

Data Modelling

- A challenge in data modelling is to balance
  - the needs of the application
  - the performance characteristics of the database engine
  - the retrieval patterns
- When designing data models, we should consider
  - the application usage of the data (i.e. queries, updates, and processing of the data)
  - the inherent structure of the data itself.

https://docs.mongodb.com/manual/core/data-modeling-introduction/

Flexible Schema

- Documents in MongoDB collections are not required to have the same schema
  - Documents do not need to have the same fields and data type for field across the collection
  - The structure of the document can be changed (add new fields, remove existing fields, or change the type)
- This flexibility facilitates the mapping of documents to an entity in your application (even if they have variations)
- In practice, however, the documents in a collection share a similar structure
- You can enforce document validation rules for a collection, during updated and insert operations

Document Structure: how to represent relationship between the data

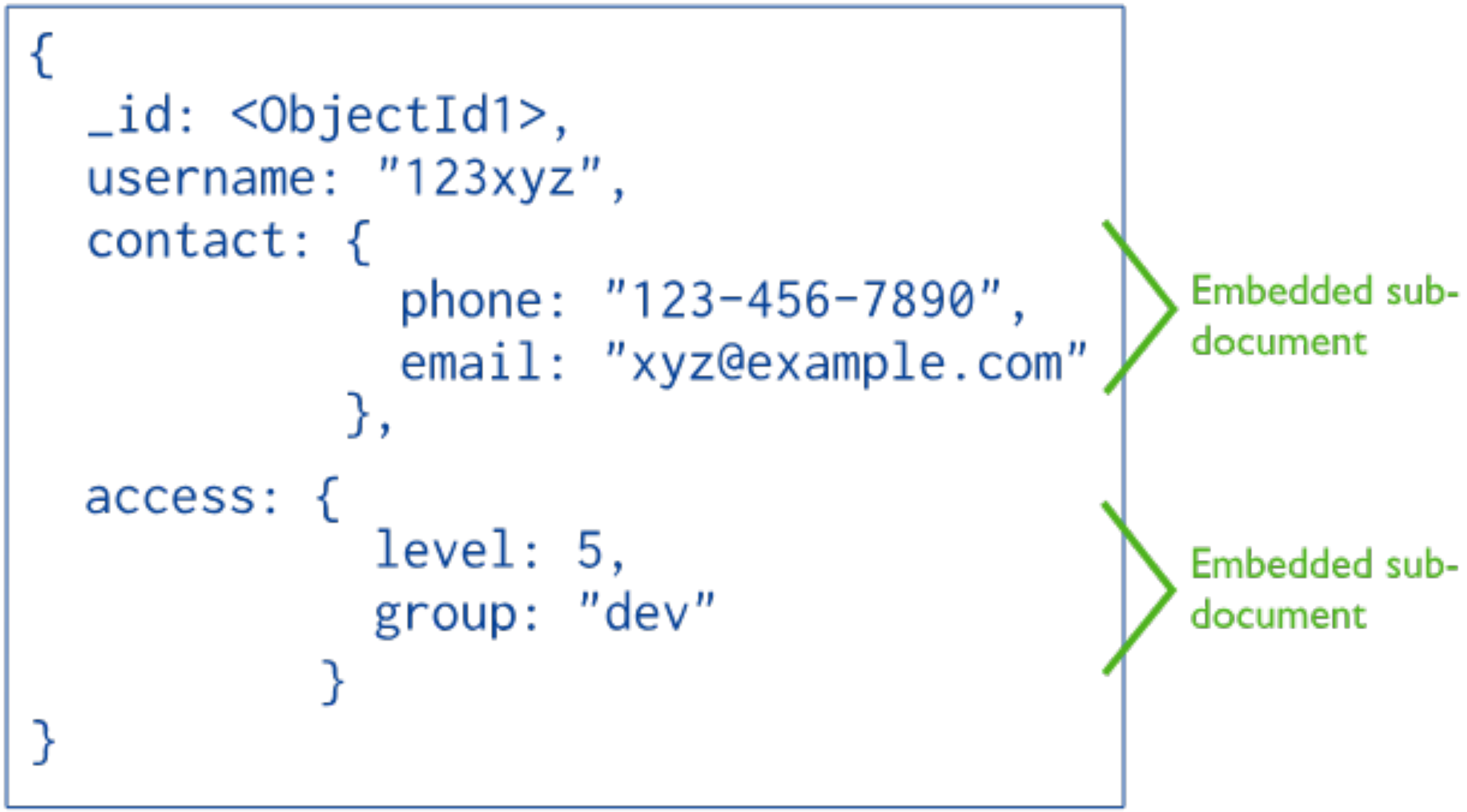
Embedded data

- Store the relationships between data by storing related data in a single document
- Documents and arrays can be embedded within a document
- *Denormalised* data models
- Related data can be retrieved and manipulated in a single operation

References

- Store the relationships between data by including links or references from one document to another
- Applications resolve these references to access the related data
- *Normalised* data models
- More than one operation required to access data.

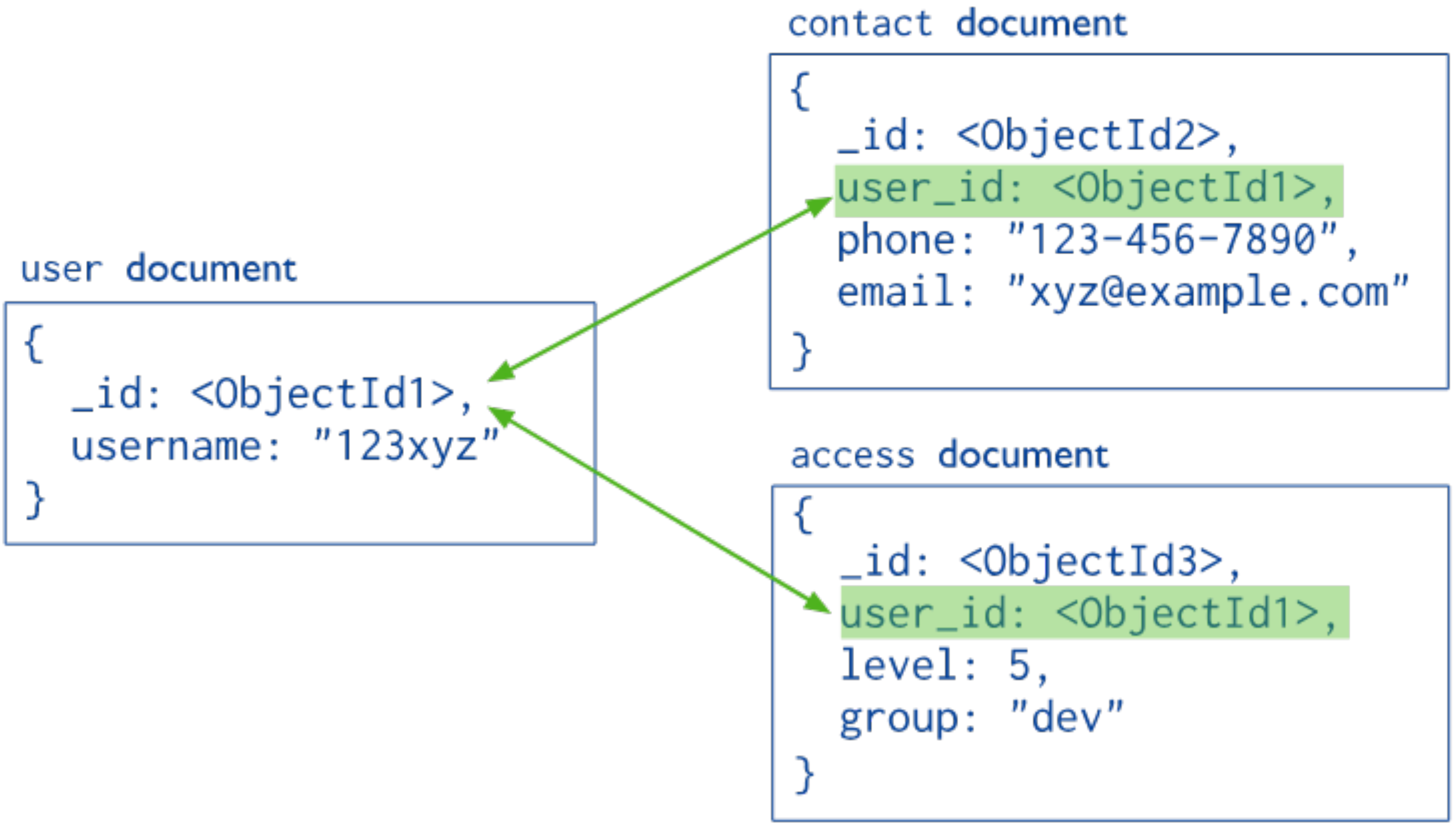
Embedded Data



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

https://docs.mongodb.com/manual/core/data-modeling-introduction/

References



contact document
```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

user document
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

access document
```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

https://docs.mongodb.com/manual/core/data-modeling-introduction/

Advantage of Embedded Data Models

- Take advantage of MongoDB's rich documents
- Allow applications to store related pieces of information in the same database record
- Applications may need to issue fewer queries and updates to complete common operations.
- Embedding provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation.
- Related data can be updated in a single atomic operation

When to Use Embedded Models

- When you have “contains” (one-to-one) relationships between entities.
- When you have one-to-many relationships between entities and
  - The “many” or child documents always appear with or are viewed in the context of the “one” or parent documents.

Example one-to-one relationship

Normalised (Reference) model

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

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

Denormalised (Embedded) model

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

Example one-to-many relationship

Normalised (Reference) model

```
{
  _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"
}
```

Denormalised (Embedded) model

```
{
  _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"
          }
         ]
 }
```

Comparing Models

Normalised (Reference) model

- The address document contains a reference to the patron document.
- If the address data is frequently retrieved with the name information, then your application needs to issue multiple queries to resolve the reference

Denormalised (Embedded) model

- A better data model would be to embed the address data in the patron data
- With the embedded data model, your application can retrieve the complete patron information with one query.

When to use Normalised Models

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

Normalised Data Models

- References provides more flexibility than embedding.
- However, client-side applications must issue follow-up queries to resolve the references.
- In other words, normalised data models can require more round trips to the server.

One-to-many Relationships with References

- Let us consider an example that maps publisher and book relationships.
- The example illustrates the advantage of referencing over embedding to avoid repetition of the publisher information.

```
{
  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"
            }
}
```

Embedding the publisher document inside the book document would lead to **repetition** of the publisher data

One-to-many relationships with References

- To avoid repetition of the publisher data, use references and keep the publisher information in a separate collection from the book collection.
- The way in which the relationship data grows will determine where to store the reference.
  - Store the book references in the publisher document : If the number of books per publisher is small with limited growth
  - Store the publisher reference inside the book document: if the number of books per publisher is large and can grow (to avoid growing mutable arrays)

```
{
  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"
}
```

If the number of books per publisher is unbounded, this data model would lead to mutable, growing arrays.

```
{
  _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"
}
```

To avoid mutable, growing arrays, store the publisher reference inside the book document

Summary

- When designing data models we should consider the structure and usage of the data (queries, updates)
- Two options
  - Embedded data
    - Related pieces of information stored in the same DB record
    - Preferred for one-to-one relationships, or one-to-many when the “many” are always viewed in the context of the “one”
    - Fewer queries required to complete an operation
  - References
    - Related pieces of information stored in different DB records
    - Preferred when embedding would result in duplication of data
    - To represent more complex many-to-many relationships
    - More queries required to complete an operation