Mongo

Embedded Documents

Like RDBMS

- Easy enough to set up two collections in a one to one relation, or even a one to many
- But not always the best thing in Mongo
- simpler to just embed the address document in the user document
  - address is only ever accessed in relation to this one user
  - often needed at the same time as the user

Create DB is if does not exist

Equivalent statements

Create a collection, if it does not exist

```
> use people
> db = db.getSiblingDB("people")

> db.user1.insertOne(
{
    _id: 111111,
    email: "email@example.com",
    name: {given: "Jane", family: "Han"},
}

db.address1.insertOne({
     _id: 121212,
     user_id: 111111,
     street: "111 Elm Street",
     city: "Springfield",
     state: "Ohio",
     country: "US",
     zip: "00000"
 })
```

```
> db.address.find({user_id: 111111})
{
    _id: 121212,
    user_id: 111111,
    street: "111 Elm Street",
    city: "Springfield",
    state: "Ohio",
    country: "US",
    zip: "00000"
}
```

Projection

what columns do you want

- In SQL, the list of columns immediately following SELECT
- MQL: a second optional argument to find
  - List all the fields you want from returned documents
  - Inevitably, list in JSON-like format
    - {fieldA:1, fieldB:1, ...}
      - value can be any non-zero **number**
  - **0** only allowed to de-select the _id field

```
people==> db.address1.find()
[
  {
    _id: 121213,
    user_id: 111111,
    location: 'work',
    street: '111 Maple Street',
    city: 'Lima',
    state: 'Ohio',
    country: 'US',
    zip: '00000'
  },
  {
    _id: 121212,
    user_id: 111111,
    street: '111 Elm Street',
    city: 'Springfield',
    state: 'Ohio',
    country: 'US',
    zip: '00000',
    location: 'home'
  }
]
people==> db.address1.find({}, {street:1, city:1})
[
  { _id: 121213, street: '111 Maple Street', city: 'Lima' },
  { _id: 121212, street: '111 Elm Street', city: 'Springfield' }
]
people==> db.address1.find({}, {street:1, number:"aaa", _id:0})
[
  { street: '111 Maple Street', number: 'aaa' },
  { street: '111 Elm Street', number: 'aaa' }
]
```

Often better to Embed

- Simpler (and faster) to just embed the address document in the user document
- Especially if:
  - address is only ever accessed in relation to this one user
  - often needed at the same time as the user

```
> db.user2.insertOne(
{
    _id: 111111,
    email: "email@example.com",
    name: {given: "Jane", family: "Han"},
    address:[{
     _id: 121212,
     user_id: 111111,
     street: "111 Elm Street",
     city: "Springfield",
     state: "Ohio",
     country: "US",
     zip: "00000"
 },{
     _id: 121214,
     user_id: 111111,
     street: "111 Maple Street",
     city: "Lina",
     state: "Ohio",
     country: "US",
     zip: "00000"
 }]})
```

```
> db.user2.findOne()


> db.user2.find({_id:111111})
```

```
db.user2.find({address: {$elemMatch: {_id:121214}}})
```

Search embedded Documents

- Query for document that has a field address and within address is an document with the _id 121214
  - returns the entire containing document
    - If you just want the address
      - Use Projection
      - Note the "address.$"

```
>db.user2.find({address: {$elemMatch: {_id:121214}}})
[
  {
    _id: 111111,
    email: 'email@example.com',
    name: { given: 'Jane', family: 'Han' },
    address: [
      {
        _id: 121212,
        user_id: 111111,
        street: '111 Elm Street',
        city: 'Springfield',
        state: 'Ohio',
        country: 'US',
        zip: '00000'
      },
      {
        _id: 121214,
        user_id: 111111,
        street: '111 Maple Street',
        city: 'Lina',
        state: 'Ohio',
        country: 'US',
        zip: '00000'
      }
    ]
  }
]
```

```
> db.user2.find({address: {$elemMatch:
{_id:121214}}}, {"address.$":1})
[
  {
    _id: 111111,
    address: [
      {
        _id: 121214,
        user_id: 111111,
        street: '111 Maple Street',
        city: 'Lina',
        state: 'Ohio',
        country: 'US',
        zip: '00000'
      }
    ]
  }
]
```

Update Embedded Documents

- Standard idea
  - identify containing document
    - then identify contained document in array (using dot notation)
    - Then update (overwrite)
      - field.$.cField

```
people==> db.user2.updateOne({_id:111111, "address._id":121214},
                                {$set: {"address.$.street":"111 Maple Lane"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

people==> db.user2.find()
[
  {
    _id: 111111,
    email: 'email@example.com',
    name: { given: 'Jane', family: 'Han' },
    address: [
     {
        _id: 121214,
        user_id: 111111,
        street: '111 Maple Lane',
        city: 'Lina',
        state: 'Ohio',
        country: 'US',
        zip: '00000'
      }, ...
    ]
  }
]
```

Embedded Subset Pattern

- A one to many relation but many is large and possibly changing frequently
- Idea, give queryer of the One side a view of some of the many
  - do this efficiently
  - need to occasionally the "some"
  - Consider new article and commentary on the article.
    - Show viewers of the article 5 comments with a "more" button
    - Embed those 5 comments into the article document
    - Other comments in a separate collection

```
people==> db.news.insertOne({_id:1, text:"I am done"})
{ acknowledged: true, insertedId: 1 }
people==> db.comment.insertOne({_id:1000, newsID:1, text:"No"})
{ acknowledged: true, insertedId: 1000 }
people==> db.comment.insertOne({_id:1001, newsID:1, text:"Yes"})
{ acknowledged: true, insertedId: 1001 }
people==> db.comment.insertOne({_id:1002, newsID:1, text:"Maybe"})
{ acknowledged: true, insertedId: 1002 }


people==>db.news.update({_id:1},{$set:
{comment:db.comments.find({}).limit(2).toArray()}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
people==> db.news.find({})
[
  {
    _id: 1,
    text: 'I am done',
    comments: [
      { _id: 1000, newsID: 1, text: 'No' },
      { _id: 1001, newsID: 1, text: 'Yes' }
    ]
  }
]
```

Extended Reference Pattern

- Show a little bit of information about the topic
  - that little bit is often enough
  - Have the ID reference so can get to the rest
- In Sakila, for films, rather than just storing actor ID, store actor name and ID

```
{  _id:1,
 title: "The Big Lebowski",
actors:[{_id:1001, name:"Jeff Bridges",},
        {_id:1002, name: "John Goodman"}
       ]
}
```