Populate

Pentru a înțelege `populate()` trebuie să înțelegem că este răspunsul Mongoose la operatorul de agregare al MongoDB $lookup. Documentația MongoDB aduce câteva lămuriri necesare pentru a înțelege modul de operare a lui `populate()` .

> Face un **left outer join** pe o colecție `unsharded` , care este în *aceeași* bază de date pentru a căuta în documentele din colecția constituită (*joined*), care va fi supusă procesării. Pentru fiecare document care intră în faza de `$lookup` se adaugă un nou câmp de array al cărui elemente sunt documentele care s-au potrivit criteriilor aflate în colecția *joined*. Faza `$lookup` pasează aceste documente remodelate către etapa următoare (lookup | docs.mongodb.com) [https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/].

Definiția lui `populate()` din documentația originală:

> Populate este procesul prin care se înlocuiesc automat căile specificate în document cu documentele aflate în alte colecții.

Este posibilă popularea unui singur document, a mai multor documente, a unor simple obiecte, a mai multor obiecte simple sau a tuturor obiectelor care au fost returnate ca urmare a unei interogări.

Documentele returnate atunci când se face popularea, pot fi `remove` able, `save` able, cu singura mențiune că nu trebuie să fie specificată opțiune `lean` . Documentele returnate nu trebuie confundate niciun moment cu sub-documentele. Pentru că atunci când se face o operațiune de ștergere, aceasta se va răsfrânge direct asupra bazei de date, fii foate atentă cu `remove` în scenariul `populate` . Pentru mai multe detalii privind metoda în sine, citește și documentația de la `Query.prototype.populate()` .

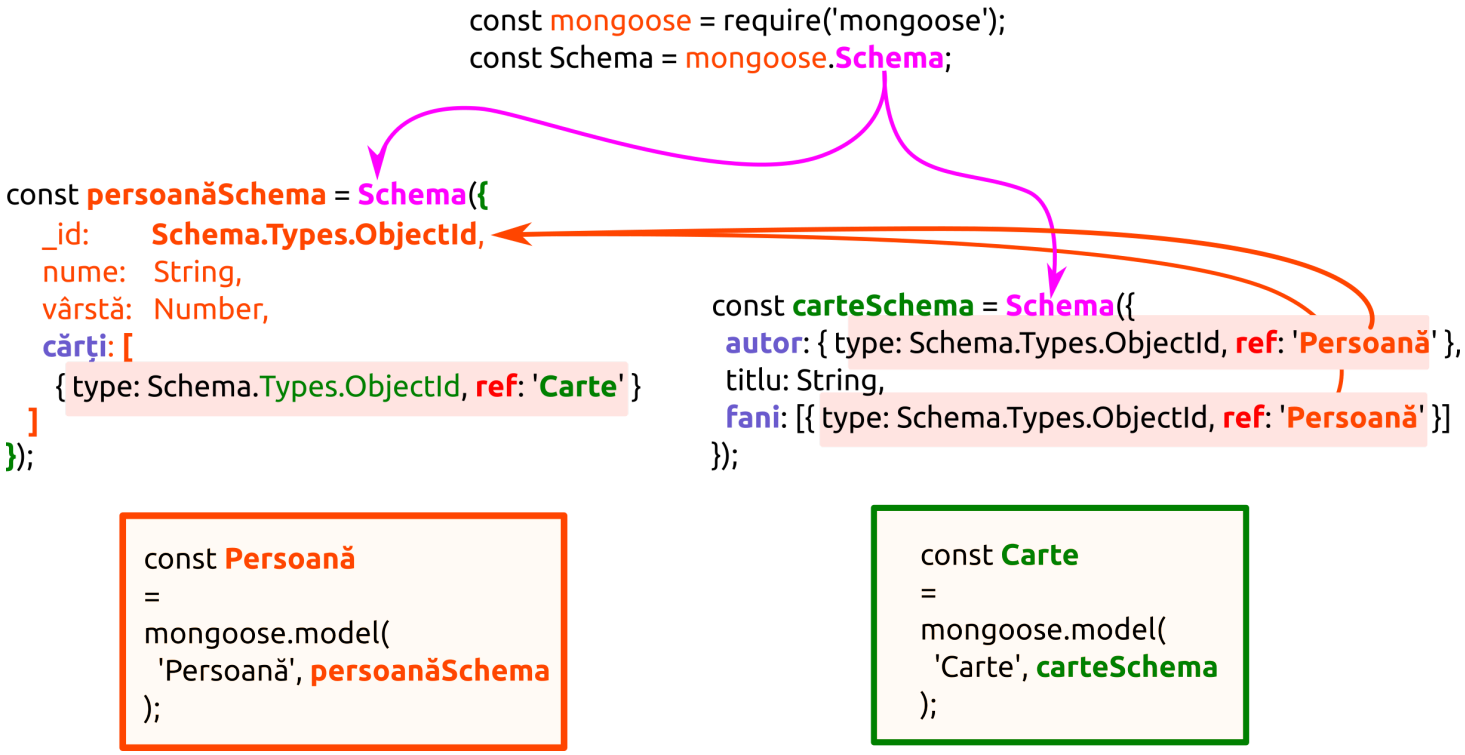
```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const persoanăSchema = Schema({
  _id: Schema.Types.ObjectId,
  nume: String,
  vârstă: Number,
  cărți: [{
    type: Schema.Types.ObjectId,
    ref: 'Carte'
  }]
});

const carteSchema = Schema({
  autor: {
    type: Schema.Types.ObjectId,
    ref: 'Persoană'
  },
  titlu: String,
  fani: [{
    type: Schema.Types.ObjectId,
    ref: 'Persoană'
  }]
});

const Carte = mongoose.model('Carte', carteSchema);
const Persoană = mongoose.model('Persoană', persoanăSchema);
```

În exemplul dat, o persoană poate scrie mai multe povești. Opțiunea `ref` îi spune lui Mongoose, care este colecția din care vor proveni documentele la momentul populării. În cazul nostru este schema `persoanăSchema`.



Id-urile care vor fi culese, trebuie să fie id-uri de documente (`_id`) din modelul de `Carte` .

Atenție, `ref` -urile pot primi următoarele valori: `ObjectId` -uri, `Number` , `String` sau `Buffer` .

Salvarea referințelor către celelalte documente

Atunci când este nevoie, poți salva `ref` -uri către alte documente. Pur și simplu trebuie să atribui valoarea `_id` -ul.

```
const autor = new Persoană({
  _id:    new mongoose.Types.ObjectId(),
  nume:   'Ian Fleming',
  vârstă: 50
});

autor.save(function (err) {
  if (err) throw err;
  // salvează autorul cu o primă operă a sa:
  const carte1 = new Carte({
    titlu: 'Casino Royale',
    autor: autor._id    // atribuie _id din modelul Persoană
  });

  carte1.save(function (err) {
    if (err) throw err;
    // am salvat cartea si apoi autorul
  });
});
```

Ceea ce am realizat este salvarea unui autor cu o primă operă/carte a sa dintr-o-o singură mișcare.

Populatea unui singur câmp cu date

Scenariul este următorul: ai nevoie să afișezi datele unei cărți. O carte poate fi scrisă de o singură persoană sau de mai multe. Pentru că anterior am introdus un singur autor cu o singură operă.

Să inițiem popularea câmpului `autor` cu datele ce reprezintă cărțile sale. Pentru că anterior am introdus doar una, datele acesteia vor popula câmpul `autor` al înregistrării de tip carte pentru un titlu ales.

```
Carte.
  findOne({ titlu: 'Casino Royale' }).
  populate('autor').
  exec(function (err, carte) {
    if (err) throw err;
    console.log('Autorul este %s', carte.autor.nume);
    // afișează "Autorul este Ian Fleming"
  });
```

În baza de date, în câmpul `autor` avem un array de id-uri ale autorilor care au scris opera. În momentul în care facem interogarea, în cazul nostru o carte, acele id-uri ale autorilor vor fi înlocuite cu documentele ce reprezintă fiecare dintre autori.

În exemplul de mai sus a fost folosită metoda `exec()` pentru a afișa punctual datele.

Popularea a mai multor câmpuri deodată

Pentru a popula mai multe câmpuri deodată, se va apela metoda `populate` pentru fiecare din câmpuri.

```
Carte.
  findOne({ titlu: 'Casino Royale' }).
  populate('autor').
  populate('fani').
  exec(function (err, carte) {
    if (err) throw err;
    console.log('Fanii sunt %s', carte.fani.nume);
    // afișează toate înregistrările de persoane pentru care câmpul fani are id-uri.
  });
```

În cazul în care, din eroare, faci o populare de mai multe ori pe aceeași cale, doar ultima va fi luată în considerare.

```
Carte.
  findOne({ titlu: 'Casino Royale' }).
  populate({path: 'fani', select: 'nume').
  populate({path: 'fani', select: 'vârstă').
  exec(function (err, carte) {
    if (err) throw err;
    console.log('Fanii sunt %s', carte.fani.nume);
    // afișează toate întregistrările de persoane pentru care câmpul fani are id-uri.
  });
  // Fiind echivalent cu:
  Carte.find().populate({ path: 'fani', select: 'vârstă' });
```

Setarea explicită a câmpurilor pentru populare

O proprietate poate fi populată manual prin setarea sa unui document. Documentul trebuie să fie o instanță a modelului la care proprietatea `ref` se referă.

```
Carte.findOne({ titlu: 'Casino Royale' }, function(error, carte) {
  if (error) {
    console.log(error);
  }
  carte.autor = autor;
  console.log(carte.autor.nume); // afișează "Ian Fleming"
});
```

Setarea explicită a câmpului populat

În cazul în care un anumit scenariu o cere, se poate atribui explicit valoarea câmpului ce va fi populat.

```
Carte.findOne({titlui: "Casino Royale"}, function (err, carte) {
  if (err) throw err;
  carte.autor = autor; // atribuirea într-o manieră explicită a valorii
  console.log(carte.autor.nume); // afișează "Ian Fleming"
});
```

Documentul corespondent nu există

În cazul în care nu avem un document corespondent din care proprietatea `ref` să poată aduce date, rezultatul `ref` va fi `null` . Este același comportament ca al unui LEFT OUTER JOIN.

```
await Persoană.deleteMany({ nume: 'Ian Fleming' });

const poveste = await Carte.findOne({ titlu: 'Casino Royale' }).populate('autor');
carte.autor; // `null`
```

În cazul în care în `carteSchema` ai avut un array de autori, `populate` , va returna un array gol.

Selectarea câtorva câmpuri, nu a întregii întregistrări

În cazul în care dorești să aduci doar câteva câmpuri din întreaga întregistrare care va popula câmpul, acestea pot fi menționate ca argumente metodei `populate()` .

```
Carte.
  findOne({ titlu: /casino royale/i }).
  populate('autor', 'nume'). // doar valoarea câmpului nume va fi adusă
  exec(function (err, carte) {
    if (err) throw err;

    console.log('Autorul este %s', carte.autor.nume);
    // afișează "Autorul este Ian Fleming"

    console.log('Vârsta autorului este %s', carte.autor.vârstă);
    // afișează "Vârsta autorului este null'
  });
```

Popularea mai multor căi deodată

În cazul în care dorești popularea mai multor căi, acest lucru este posibil:

```
Carte.
  find({ titlu: 'Casino Royale' }).
  populate('fans').
  populate('author').
  exec();
```

Popularea cu anumite condiții - `match`

Uneori dorești să faci o alegere a unor date cu care se va face popularea în funcție de anumite criterii. În exemplul de mai jos, vom aduce toate cărțile ale căror fani au vârsta peste 21 de ani.

```
Carte.
  find().
  populate({
    path: 'fani',
    match: { vârstă: { $gte: 18 }},
    // Excluzi `_id` în mod explicit, vezi http://bit.ly/2aEfTdB
    select: 'nume -_id',
    options: { limit: 5 }
  }).
  exec();
```

Eliminarea înregistrărilor folosind `virtual()` și `match()`

Să presupunem că o anumită resursă are mai multe comentarii. Unele dintre acestea sunt șterse de administrator. Pentru a filtra documentele comentariu care au setat un boolean ce indică ștergerea, ne putem ajuta de un posibil model, care folosește o zonă de documente găsite ca tampon de date implicând `virtual()` .

```
// Creează schema resursei
const resursaSchema = new mongoose.Schema({
  titlu:   String,
  autorId: Number
});
// creează virtualul
resursaSchema.virtual('comentarii', {
  ref:        'Comentariis',
  localField: '_id',
  foreignField: 'resursaId',
  // În momentul în care populezi comentariile, exclude-le pe cele care au fost șterse`;
  options: { match: { sters: { $ne: true } } }
});

const comentariuSchema = new mongoose.Schema({
  _id:       Number,
  resursaId: mongoose.ObjectId,
  authrId:   Number,
  sters:     Boolean
});
```

În materialul dedicat metodei `match()` este detaliat modul în care poți combina `populate` cu `match` pentru a aduce rezultate.

```
const authorId = 1
post = await BlogPost.find().populate({
  path: 'comments',
  match: doc => (doc.authorId === authorId ? {} : { deleted: { $ne: true } })
}); // după Valeri Karpov - vezi Aggregate.prototype.match
```

Cum facem legăturile între colecții

Există posibilitatea ca atunci când folosim obiectul `autor` , să descoperim că nu se populează lista cărților. Acest lucru este pentru că încă nu au fost introduse obiecte `carte` în `autor.carti` .

Aici există două abordări:

- lași `autor` -ul să-și cunoască cărțile. De regulă schema ar trebui să rezolve relații de tipul one-to-many prin existența unui pointer către părinte în zona de many (în cazul nostru o înregistrare carte să aibă id-ul autorului).
- dacă există un motiv întemeiat să constitui un array de pointeri către documentele copil (carti), poți face `push()` documentelor în array precum în următorul exemplu:

```
// mai întâi constitui schema autor
autor.carti.push(cartea1);
autor.save(callback);
```

Acest mod permite combinarea căutări cu popularea.

```
Persoana.
  findOne({ nume: 'Ian Fleming' }).
  populate('carti'). // merge doar dacă am făcut push la ref-uri către copil
  exec(function (err, persoana) {
    if (err) return handleError(err);
    console.log(persoana);
  });
```

În cazul în care setăm pointeri și în `Persoana` către `Carte`, dar și în `Carte` către `Persoana`, fie o parte, fie cealaltă se va desincroniza, adică va trebui să găsim o cale prin care ambele colecții să fie actualizate. În loc de populare, mai eficient ar fi să se facă un `find()` pe cărți.

```
Carte.
  find({ autor: autor._id }).
  exec(function (err, carti) {
    if (err) throw err;
    console.log('Cărțile sunt un array ce conține: ', carti);
  });
```

Popularea unui document

În cazul în care deja au un document și dorești să populezi unele căi ale sale, vei folosi metoda `populate()`, pe care `Document` o pune la dispoziție. Pentru mai multe detalii utile, vezi documentația pentru `Document.prototype.populate()`.

Popularea unui model

Pentru a face experimentele mai simple, mai întâi vom crea două colecții introducându-le și referințele necesare.

```
const mongoose = require('mongoose');
// MONGOOSE - Conectare la MongoDB
mongoose.set('useCreateIndex', true); // Deprecation warning
mongoose.connect('mongodb://localhost:27017/lucru', {useNewUrlParser: true, useUnifiedTo
    if (error) throw error;
});

var Schema = mongoose.Schema;

// 2 models: Book and Author
var bookSchema = new Schema({
    title: String,
    author: {
        type: mongoose.ObjectId,
        ref: 'Author'
    }
});
var Book = mongoose.model('Book', bookSchema);

var authorSchema = new Schema({
    name: String
});

var Author = mongoose.model('Author', authorSchema);

// Creează cărțile cu autorii lor
var arr = [
    { name: 'Michael Crichton' },
    { name: 'Ian Fleming' }
];

Author.create(arr, (err, a) => {
    if (err) throw err;
    const [author1, author2] = a;
    var arr2 = [
        { title: 'Jurassic Park', author: author1._id },
        { title: 'Casino Royale', author: author2._id }
    ];
    Book.create(arr2, (err, b) => {
        if (err) throw err;
    });
});
```

În momentul în care facem popularea, vom obține un rezultat foarte interesant în sensul în care vor fi aduse și înregistrările `null` precum în cazul unui **LEFT OUTER JOIN**.

```
Book.find().populate({
    path: 'author',
    match: { name: 'Ian Fleming' }
}).exec(function (err, books) {
    if (err) return handleError(err);
    books.forEach((book) => {
        console.log(book);
    });
});
/*
{
  _id: 5da42cba5edea641e4813518,
  title: 'Jurassic Park',
  author: null,
  __v: 0
}
{
  _id: 5da42cba5edea641e4813519,
  title: 'Casino Royale',
  author: { _id: 5da42cba5edea641e4813517, name: 'Ian Fleming', __v: 0 },
  __v: 0
}
*/
```

Este observabil faptul că în afară de înregistrarea care este utilă, mai este adusă una a cărui date de la câmpul `author` este `null` . De fapt, vor fi aduse multe alte documente pe lângă cel de interes pentru că în spate, ceea ce se petrece, este o operațiune în două faze:

1. Mai întâi se face o căutare globală cu

```
Book.find().populate({})
```

2. Apoi se filtrează folosind un scenariu de tipul:

```
Author.find({ _id: { $in: books.map(b => b.author) }, name: 'Ian Fleming' })
```

Pentru a filtra eficient, totuși calea corectă este de a introduce numele autorului în înregistrarea de carte.

```
// avem două modele: Book și Author
const Book = mongoose.model('Book', Schema({
  title: String,
  author: {
    type: mongoose.ObjectId,
    ref: 'Author'
  },
  authorName: String
}));

const authorSchema = Schema({ name: String });
// Adaugi middleware pentru a actualiza numele autorului dereferenced în câmpul `authorN
authorSchema.pre('save', async function() {
  if (this.isModified('name')) {
    await Book.updateMany({ authorId: this.author }, { authorName: this.name });
  }
});
const Author = mongoose.model('Author', authorSchema);
```

Mecanismul de mai sus implică faptul că un autor va salva numele său ca string în înregistrarea cărții, dar va fi preent și id-ul său ca referință către colecția autorilor. În cazul în care numele se schimbă, se va actualiza și numele ca string pentru operele sale. Această metodă se numește dereferențiere. Acest mecanism este preferabil lui `populate()` pentru că micșorează numărul de atingeri ale bazei prin query-uri complexe. În plus, nu este nevoie de mecanisme de caching.

Regula rapidă este `store what you query for` - înmagazineazp informația după care vei face căutări direct în înregistrare.

Popularea în adâncime

```
User.
  findOne({ name: 'Val' }).
  populate({
    path: 'friends',
    // Get friends of friends - populate the 'friends' array for every friend
    populate: { path: 'friends' }
  });
```

Referințe dinamice cu `refPath`

În baza valorii unei proprietăți a documentului, Mongoose poate popula cu date din mai multe colecții odată. Exemplul oferit în documentație este cel al unui user care poate comenta fie la un blogpost sau la un produs.

Pentru a aduce informație din mai multe colecții, în câmpul în care în mod obișnuit avem referința, în loc de a hardcoda numele colecției, vom face o redirectare către un câmp suplimentar, în care vor fi menționate colecțiile din care vor putea fi aduse informațiile în funcție de necesități.

```
const commentSchema = new Schema({
  body: { type: String, required: true },
  on: {
    type: Schema.Types.ObjectId,
    required: true,
    // În loc de numele hardcodat din `ref`, `refPath` face ca Mongoose
    // să se uite la proprietatea `onModel` pentru a găsi modelul corect.
    refPath: 'onModel'
  },
  onModel: {
    type: String,
    required: true,
    enum: ['BlogPost', 'Product']
  }
});

const Product  = mongoose.model('Product', new Schema({ name: String }));
const BlogPost = mongoose.model('BlogPost', new Schema({ title: String }));
const Comment  = mongoose.model('Comment', commentSchema);
```

Este important de subliniat că valorile din proprietatea `enum`, în cazul exemplului fiind `['BlogPost', 'Product']` sunt tot atâtea posibilități de a construi documente care să fie trimise în bază. Pe scurt, se realizează un necenism de reutilizare a schemei în funcție de cui va fi atașat comentariul. Dacă este un comentariu pe un produs, se va completa proprietatea `onModel` cu numele colecției căreia comentariul a fost atașat unuia dintre documentele întroduse.

Folosind `refPath`, poți configura ce model poate folosi Mongoose pentru fiecare document în parte.

```
// creezi două documente noi și în bază sunt trimise două înregistrări noi
const book = await Product.create({ name: 'The Count of Monte Cristo' });
const post = await BlogPost.create({ title: 'Top 10 French Novels' });

// creezi două comentarii noi: unul la înregistrarea de Product care tocmai a fost trimi
const commentOnBook = await Comment.create({
  body: 'Great read',
  on: book._id,
  onModel: 'Product'
});
// și altul la înregistrarea proaspătă `BlogPost`
const commentOnPost = await Comment.create({
  body: 'Very informative',
  on: post._id,
  onModel: 'BlogPost'
});

// Mai jos, `populate()` funcționează chiar dacă un comentariu se referă la colecția `Pr
// iar celălalt la `BlogPost`
const comments = await Comment.find().populate('on').sort({ body: 1 });
comments[0].on.name; // "The Count of Monte Cristo"
comments[1].on.title; // "Top 10 French Novels"
```

Alternativa ar fi definirea separată a proprietăților `blogPost` și `product` în `commentSchema` urmată de popularea deodată a acestora.

```
const commentSchema = new Schema({
  body: { type: String, required: true },
  product: {
    type: Schema.Types.ObjectId,
    required: true,
    ref: 'Product'
  },
  blogPost: {
    type: Schema.Types.ObjectId,
    required: true,
    ref: 'BlogPost'
  }
});
// The below `populate()` is equivalent to the `refPath` approach, you
// just need to make sure you `populate()` both `product` and `blogPost`.
const comments = await Comment.find().
  populate('product').
  populate('blogPost').
  sort({ body: 1 });
comments[0].product.name; // "The Count of Monte Cristo"
comments[1].blogPost.title; // "Top 10 French Novels"
```

În cazul în care sunt definite separat proprietățile `blogPost` și `product`, acest mecanism este acceptabil pentru cazurile simple. În cazul în care lucrurile devin ceva mai complicate, adică schema unui comentariu să fie folosită pentru a crea documente atașate mai multor altor documente ale mai multor colecții (comentarii făcute la alte comentarii), vei avea nevoie de câte un `populate()` pentru fiecare dintre documentele colecțiilor referite. Folosirea lui `refPath` este mai eficientă pentru că ai nevoie de doar două căi și un singur `populate()` indiferent de câte modele atinge `commentSchema`.

Popularea virtualelor

Popularea bazată pe câmpul `_id` este o soluție, dar pentru a realiza relații mai sofosticate între documente, se vor folosi *virtuals*.

```
const PersonSchema = new Schema({
  name: String,
  band: String
});

const BandSchema = new Schema({
  name: String
});
BandSchema.virtual('members', {
  ref: 'Person',        // Modelul care va fi folosit
  localField: 'name',   // Persoanele găsite vor fi în câmpul specificat la `localField`
  foreignField: 'band', // câmpul din documentele colecției cu care se leagă `foreignFie
  // Dacă `justOne` este true, 'members' va fi un singur document, altfel, un array
  // `justOne` este false din oficiu.
  justOne: false,
  options: { sort: { name: -1 }, limit: 5 } // pentru opțiuni, vezi: http://bit.ly/mongo
});

const Person = mongoose.model('Person', PersonSchema);
const Band = mongoose.model('Band', BandSchema);

/**
 * Suppose you have 2 bands: "Guns N' Roses" and "Motley Crue"
 * And 4 people: "Axl Rose" and "Slash" with "Guns N' Roses", and
 * "Vince Neil" and "Nikki Sixx" with "Motley Crue"
 */
Band.find({}).populate('members').exec(function(error, bands) {
  /* `bands.members` is now an array of instances of `Person` */
});
```

Resurse:

- Mongoose Design Pattern: Store What You Query For | The Code Barbarian
- LEFT JOIN vs. LEFT OUTER JOIN in SQL Server | stackoverflow
- Mongoose 4.5 Virtual Populate | http://thecodebarbarian.com