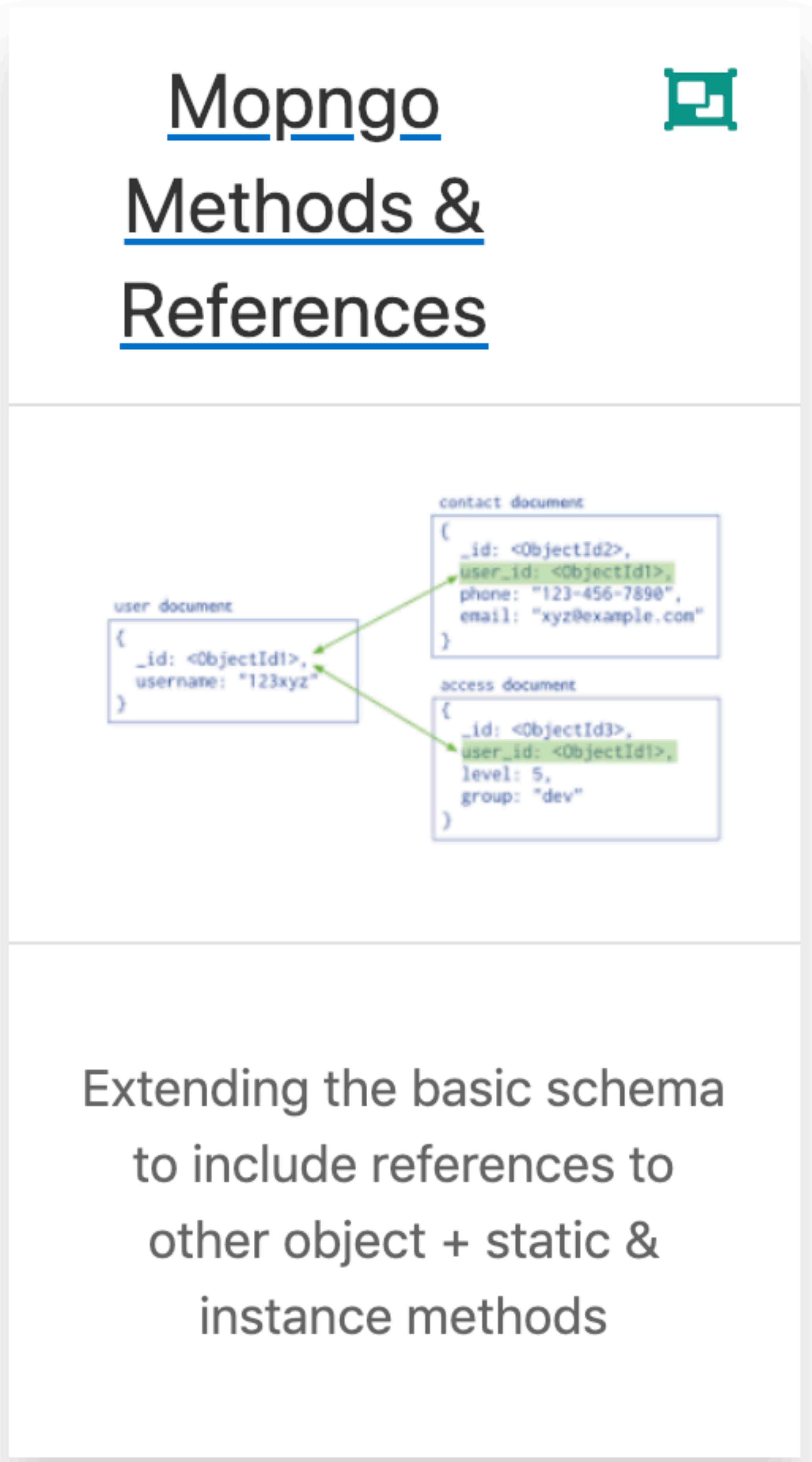
Mongo Methods & References



Mopngo Methods & References

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

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

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

Extending the basic schema to include references to other object + static & instance methods

- Instance Methods
- Object References
- Query Population

Mongoose Models:

- Instance Methods	→	*Methods associated with a Schema*
- Statics	→	*Methods associated with an Object*

Instance Methods

Methods

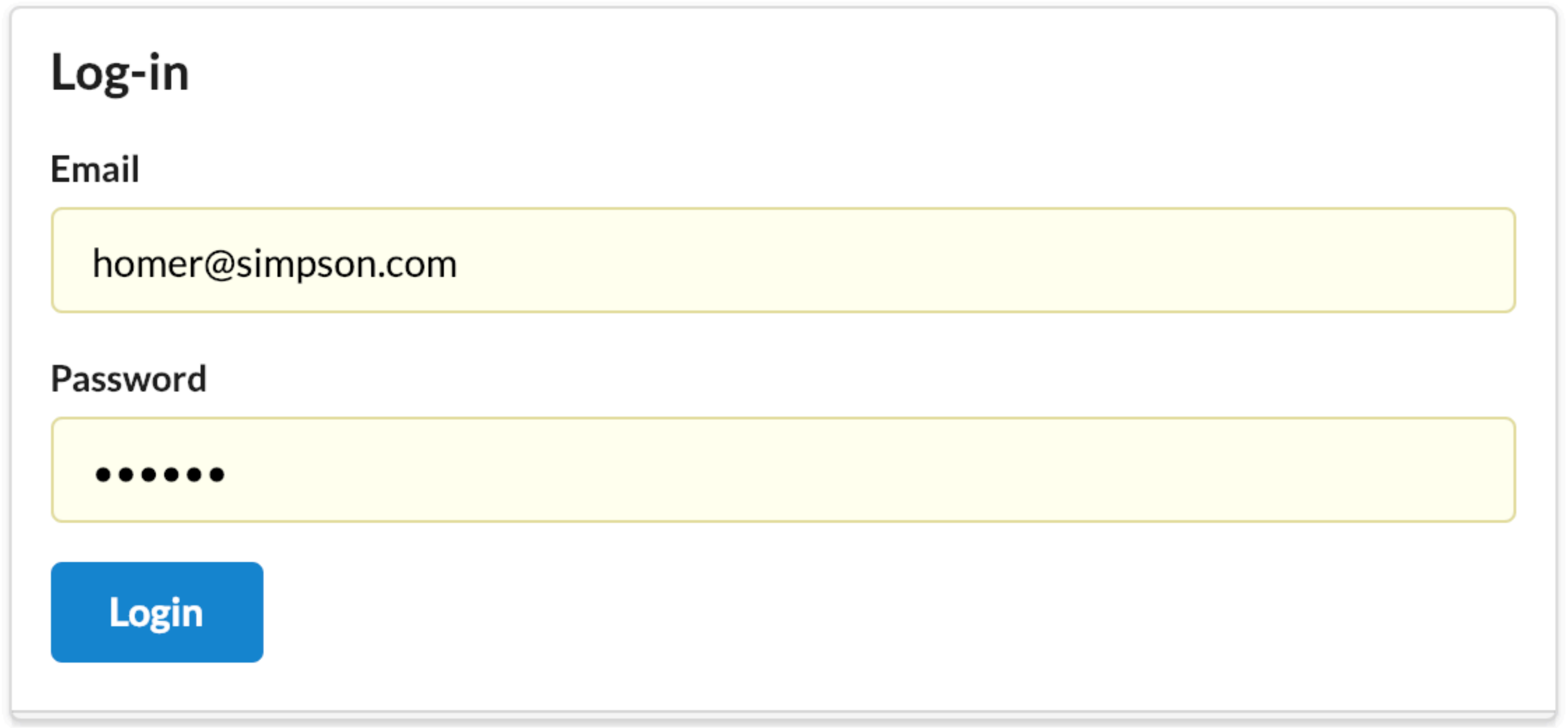
Schema

```
const userSchema = new Schema({
  firstName: String,
  lastName: String,
  email: String,
  password: String
});
```

'Static' method: defined independently of any object

instance method: associated with an object

```
userSchema.statics.findByEmail = function(email) {
  return this.findOne({ email : email});
};

userSchema.methods.comparePassword = function(candidatePassword) {
  const isMatch = this.password === candidatePassword;
  if (!isMatch) {
    throw new Boom('Password mismatch');
  }
  return this;
};
```


Log-in

Email

homer@simpson.com

Password

••••••

```
handler: async function(request, h) {
  const { email, password } = request.payload;
  let user = await User.findByEmail(email);
  if (!user) {
    return h.redirect(‘/');
  }
  user.comparePassword(password);
  request.cookieAuth.set({ id: user.id });
  return h.redirect('/home');
}
```

Invoke static method

Invoke static method

Built in Static methods (mongoose)

https://mongoosejs.com/docs/api.html#document-js

- Mongoose.prototype.createConnection()
- Mongoose.prototype.deleteModel()
- Mongoose.prototype.disconnect()
- Mongoose.prototype.get()
- Mongoose.prototype.model()
- Mongoose.prototype.modelNames()
- Mongoose.prototype.mongo
- Mongoose.prototype.mquery
- Mongoose.prototype.now()
- Mongoose.prototype.plugin()
- Mongoose.prototype.pluralize()
- Mongoose.prototype.set()
- Mongoose.prototype.startSession()
- Mongoose.prototype.version

- Mongoose()
- Mongoose.prototype.Aggregate()
- Mongoose.prototype.CastError()
- Mongoose.prototype.Collection()
- Mongoose.prototype.Connection()
- Mongoose.prototype.Decimal128
- Mongoose.prototype.Document()
- Mongoose.prototype.DocumentProvider()
- Mongoose.prototype.Error()
- Mongoose.prototype.Mixed
- Mongoose.prototype.Model()
- Mongoose.prototype.Mongoose()
- Mongoose.prototype.Number
- Mongoose.prototype.ObjectId
- Mongoose.prototype.Promise
- Mongoose.prototype.PromiseProvider()
- Mongoose.prototype.Query()
- Mongoose.prototype.STATES
- Mongoose.prototype.Schema()
- Mongoose.prototype.SchemaType()
- Mongoose.prototype.SchemaTypes
- Mongoose.prototype.Types
- Mongoose.prototype.VirtualType()
- Mongoose.prototype.connect()
- Mongoose.prototype.connection
- Mongoose.prototype.connections

Object References

Donor References

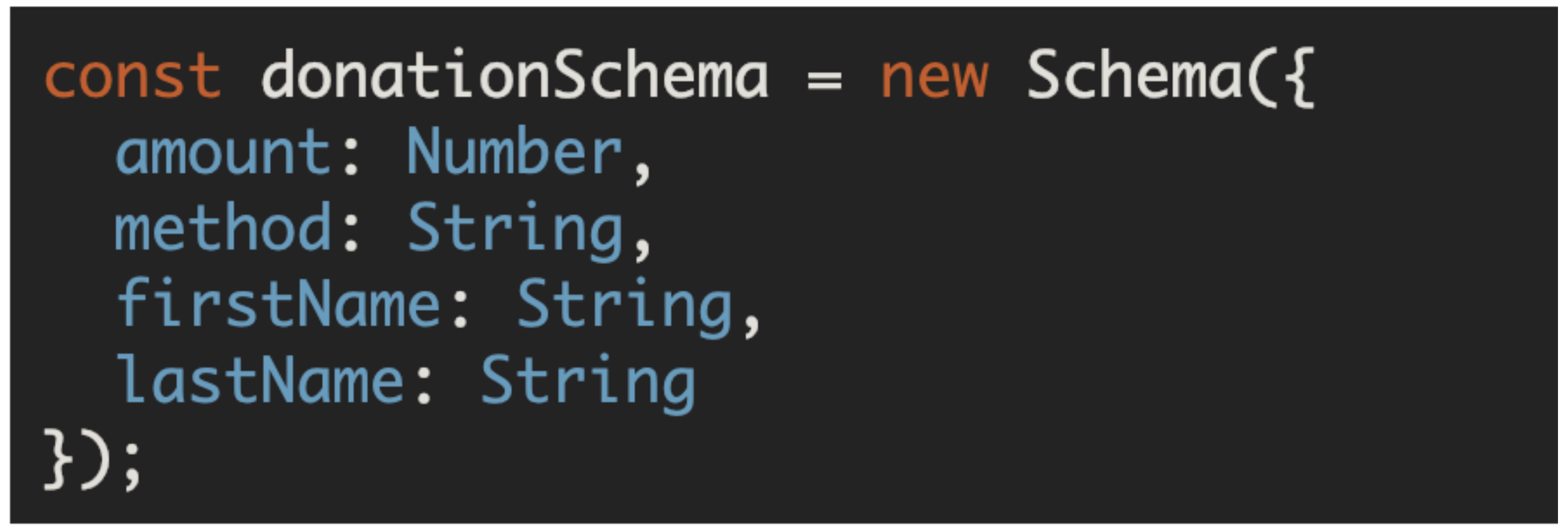
```
const donationSchema = new Schema({
  amount: Number,
  method: String,
  firstName: String,
  lastName: String
});
```

```
donate: {
  handler: async function(request, h) {
    const id = request.auth.credentials.id;
    const user = await User.findById(id);
    const data = request.payload;
    const newDonation = new Donation({
      amount: data.amount,
      method: data.method,
      firstName: user.firstName,
      lastName: user.lastName
    });
    await newDonation.save();
    return h.redirect('/report');
  }
}
```

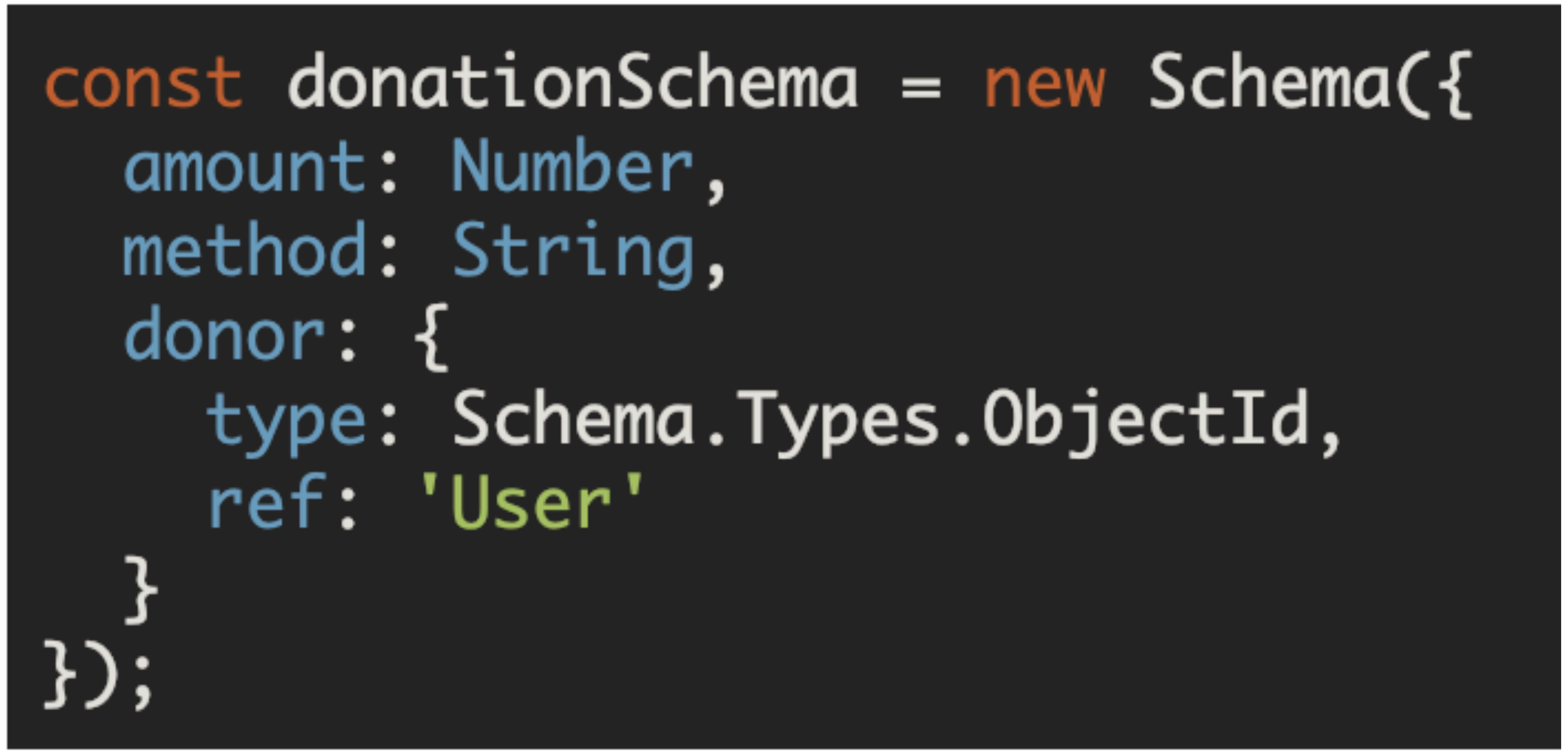
```
{{#each donations}}
  <tr>
    <td> {{amount}} </td>
    <td> {{method}} </td>
    <td> {{firstName}} {{lastName}} </td>
  </tr>
{{/each}}
```

Amount	Method donated	Donor
50	paypal	
100	paypal	
100	paypal	
50	direct	aaa aaa

Donor References: ObjectId

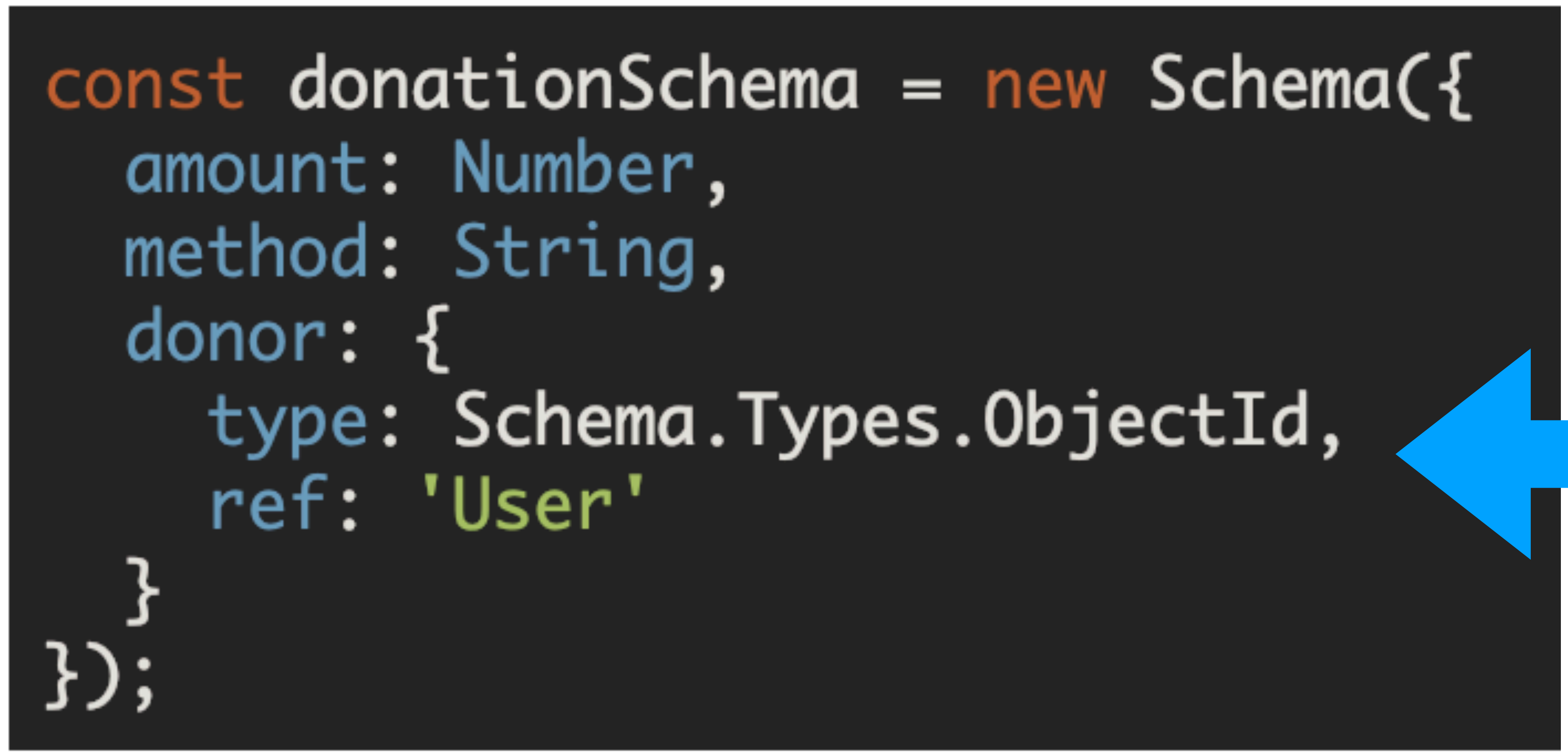


```
const donationSchema = new Schema({
  amount: Number,
  method: String,
  firstName: String,
  lastName: String
});
```



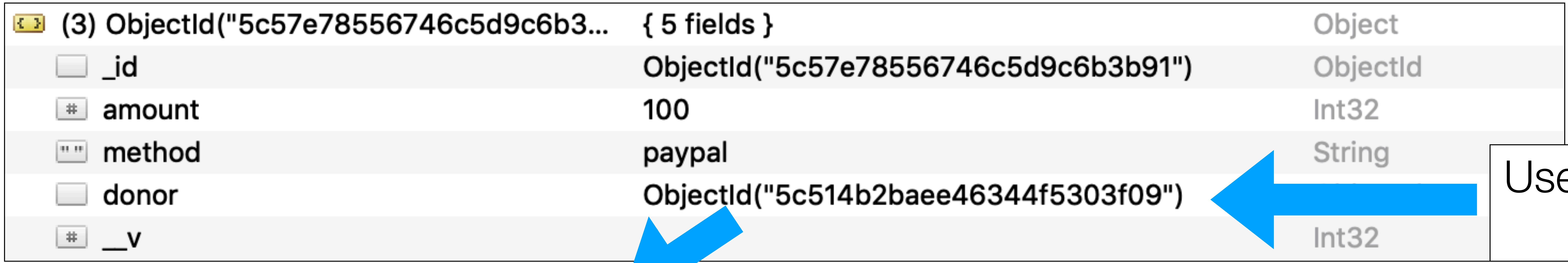
```
const donationSchema = new Schema({
  amount: Number,
  method: String,
  donor: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  }
});
```

donor: an object reference to a User object



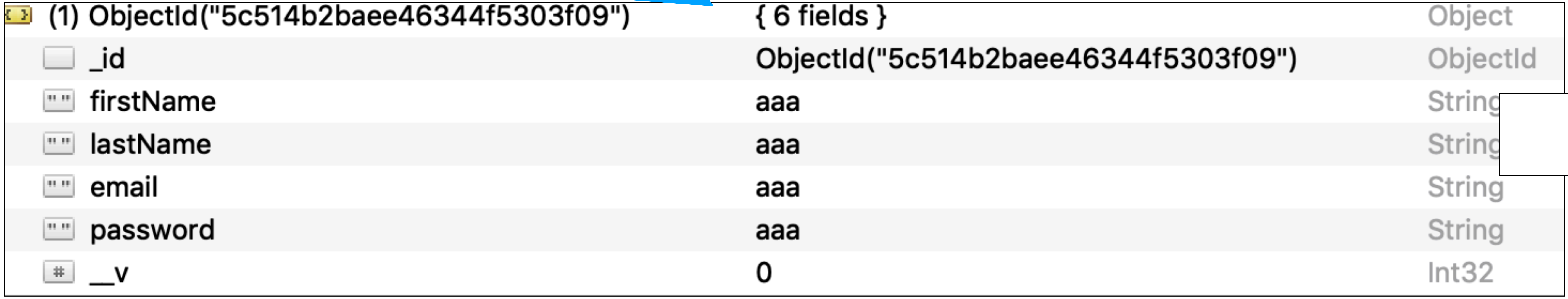
```
const donationSchema = new Schema({
  amount: Number,
  method: String,
  donor: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  }
});
```

User Reference



(3) ObjectId("5c57e78556746c5d9c6b3...	{ 5 fields }	Object
_id	ObjectId("5c57e78556746c5d9c6b3b91")	ObjectId
amount	100	Int32
method	paypal	String
donor	ObjectId("5c514b2baee46344f5303f09")	
__v		Int32

User Reference instance



(1) ObjectId("5c514b2baee46344f5303f09")	{ 6 fields }	Object
_id	ObjectId("5c514b2baee46344f5303f09")	ObjectId
firstName	aaa	String
lastName	aaa	String
email	aaa	String
password	aaa	String
__v	0	Int32

User

Query Population

Query Population

Population is the process of automatically replacing the specified paths in the document with document(s) from other collection(s).

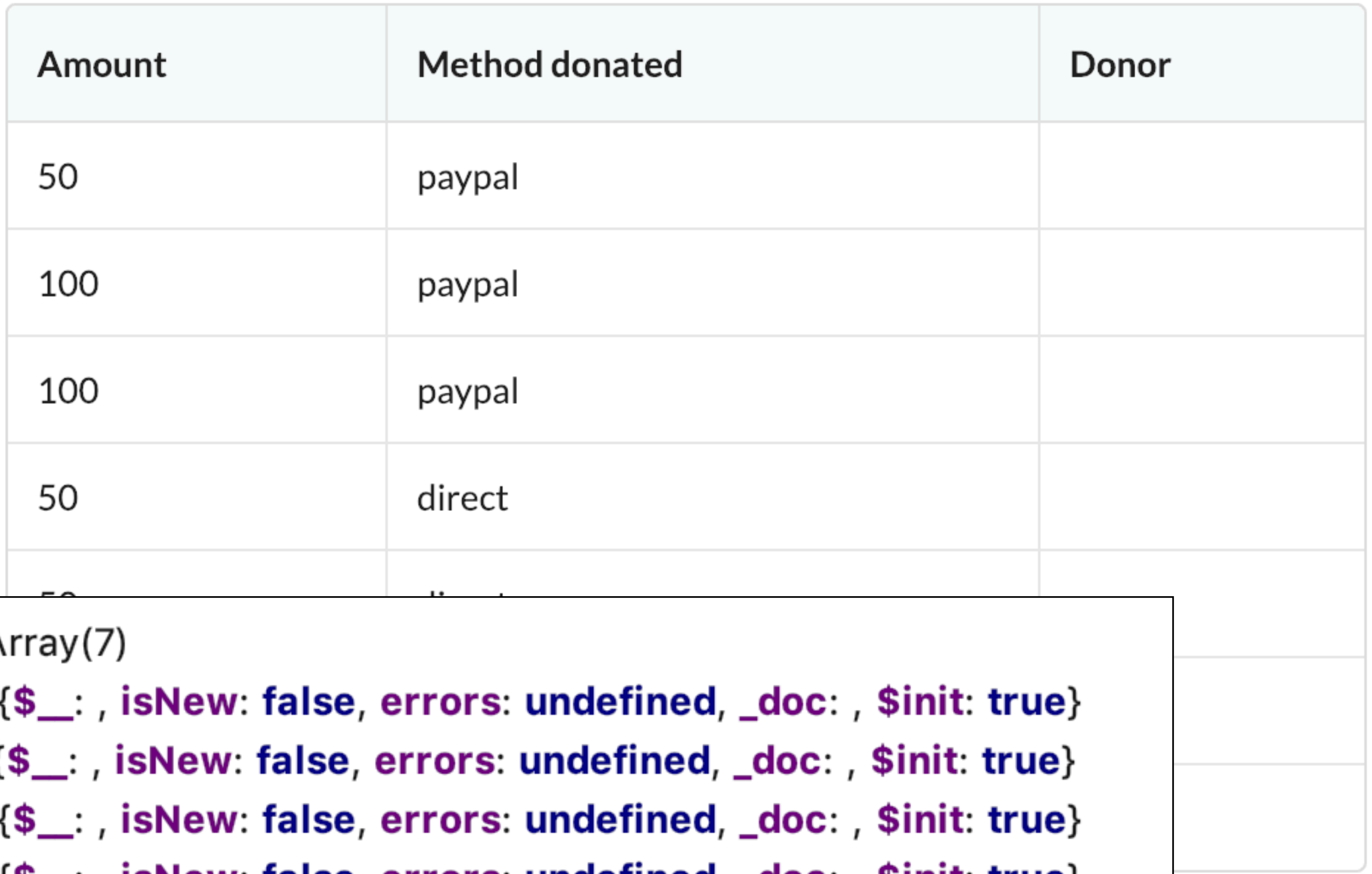
We may populate a single document, multiple documents, plain object, multiple plain objects, or all objects returned from a query.

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

const personSchema = Schema({
  _id: Schema.Types.ObjectId,
  name: String,
  age: Number,
  stories: [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});

const storySchema = Schema({
  author: { type: Schema.Types.ObjectId, ref: 'Person' },
  title: String,
  fans: [{ type: Schema.Types.ObjectId, ref: 'Person' }]
});

const Story = mongoose.model('Story', storySchema);
const Person = mongoose.model('Person', personSchema);
```

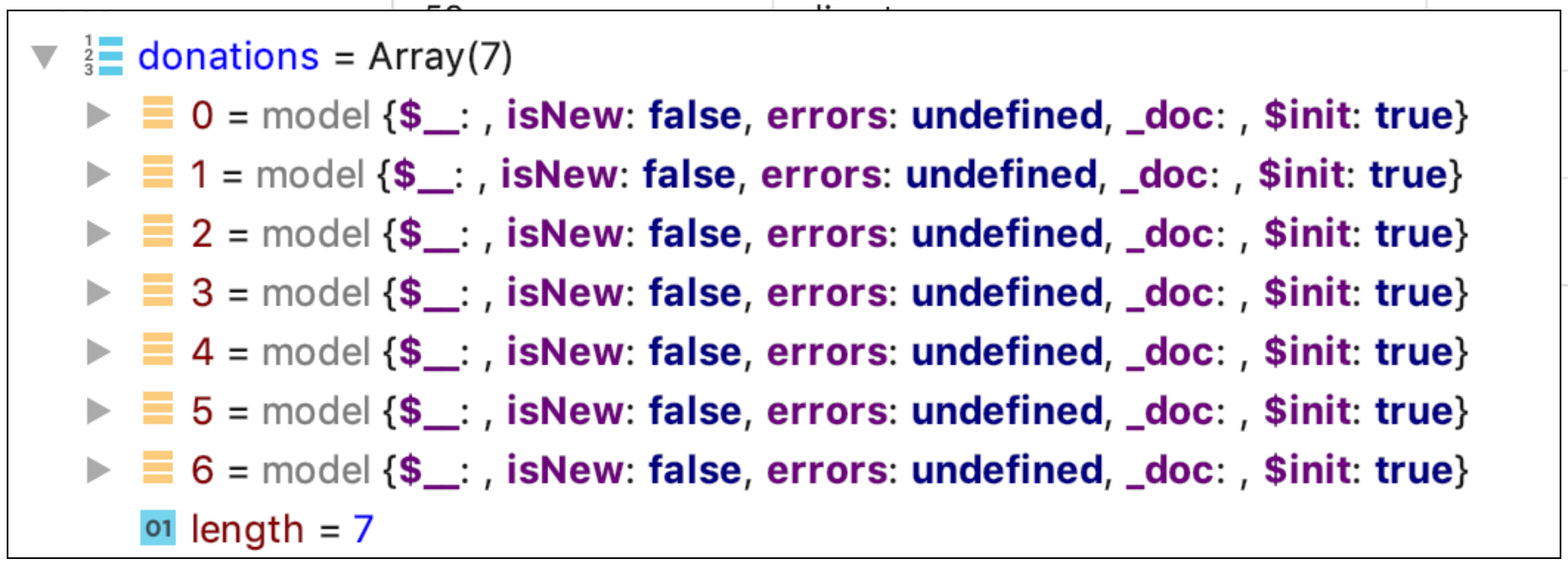
https://mongoosejs.com/docs/populate.htm

```
const donationSchema = new Schema({
  amount: Number,
  method: String,
  donor: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  }
});
```

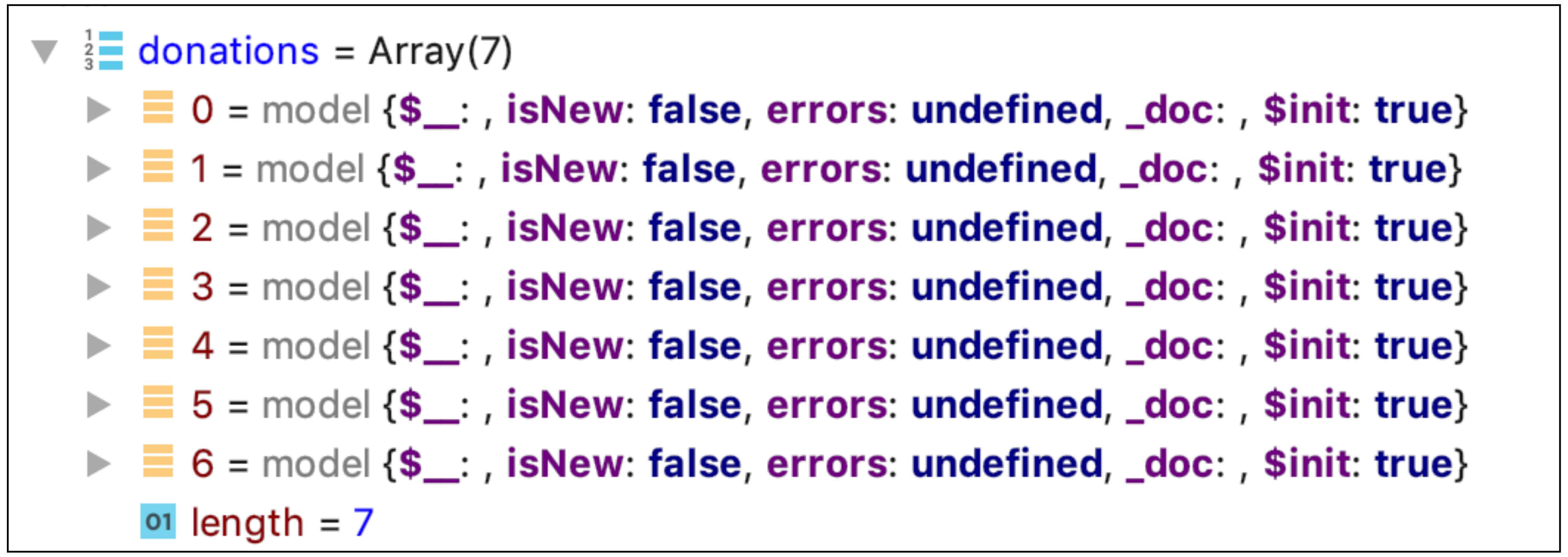
```
report: {
  handler: async function(request, h) {
    const donations = await Donation.find();
    return h.view('report', {
      title: 'Donations to Date',
      donations: donations
    });
  }
},
```



Amount	Method donated	Donor
50	paypal	
100	paypal	
100	paypal	
50	direct	

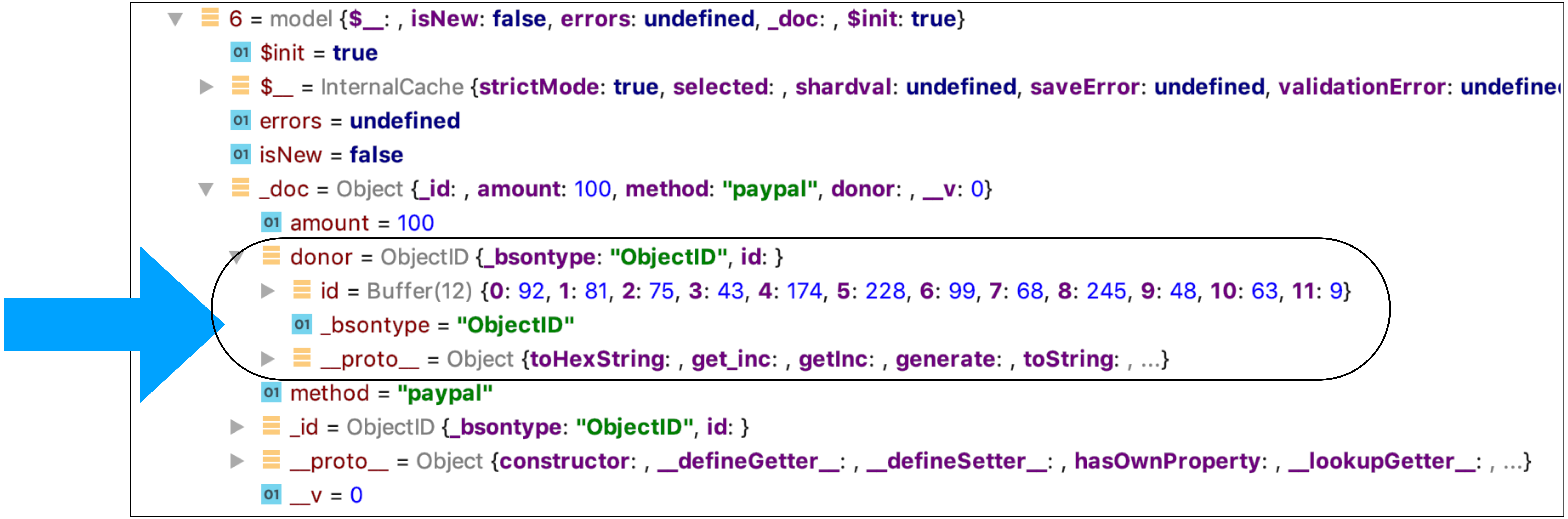


```
▼ donations = Array(7)
  ▶ 0 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 1 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 2 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 3 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 4 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 5 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 6 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
    length = 7
```



```
▼ donations = Array(7)
  ▶ 0 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 1 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 2 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 3 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 4 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 5 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
  ▶ 6 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
    length = 7
```

- Default query returns donations with donor containing ObjectID only



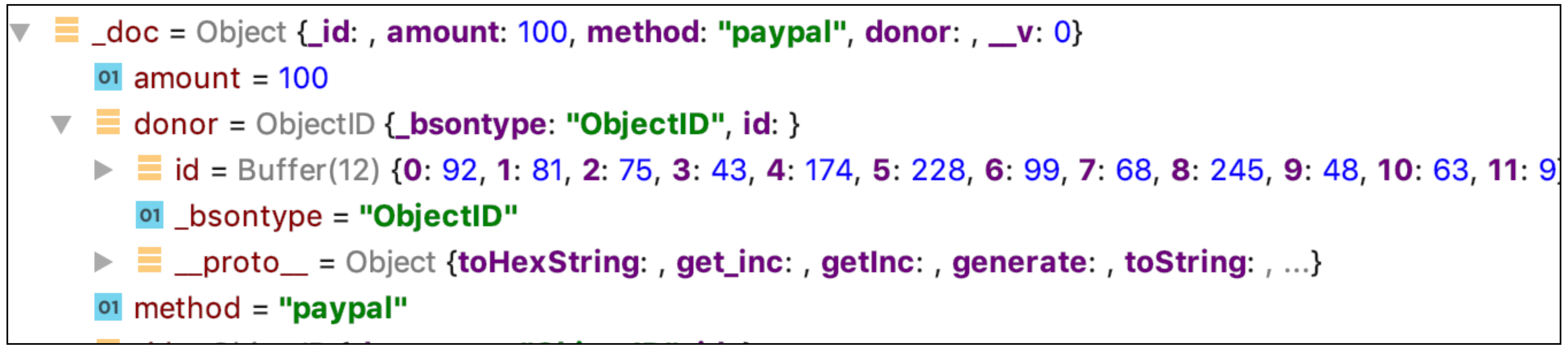
```
▼ 6 = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
    $init = true
  ▶ $__ = InternalCache {strictMode: true, selected: , shardval: undefined, saveError: undefined, validationError: undefine
    errors = undefined
    isNew = false
  ▼ _doc = Object {_id: , amount: 100, method: "paypal", donor: , __v: 0}
      amount = 100
    ▼ donor = ObjectID {_bsontype: "ObjectID", id: }
      ▶ id = Buffer(12) {0: 92, 1: 81, 2: 75, 3: 43, 4: 174, 5: 228, 6: 99, 7: 68, 8: 245, 9: 48, 10: 63, 11: 9}
        _bsontype = "ObjectID"
      ▶ __proto__ = Object {toHexString: , get_inc: , getInc: , generate: , toString: , ...}
      method = "paypal"
    ▶ _id = ObjectID {_bsontype: "ObjectID", id: }
    ▶ __proto__ = Object {constructor: , __defineGetter__: , __defineSetter__: , hasOwnProperty: , __lookupGetter__: , ...}
      __v = 0
```

```
report: {
  handler: async function(request, h) {
    const donations = await Donation.find();
    return h.view('report', {
      title: 'Donations to Date',
      donations: donations
    });
  }
},
```

Default Query

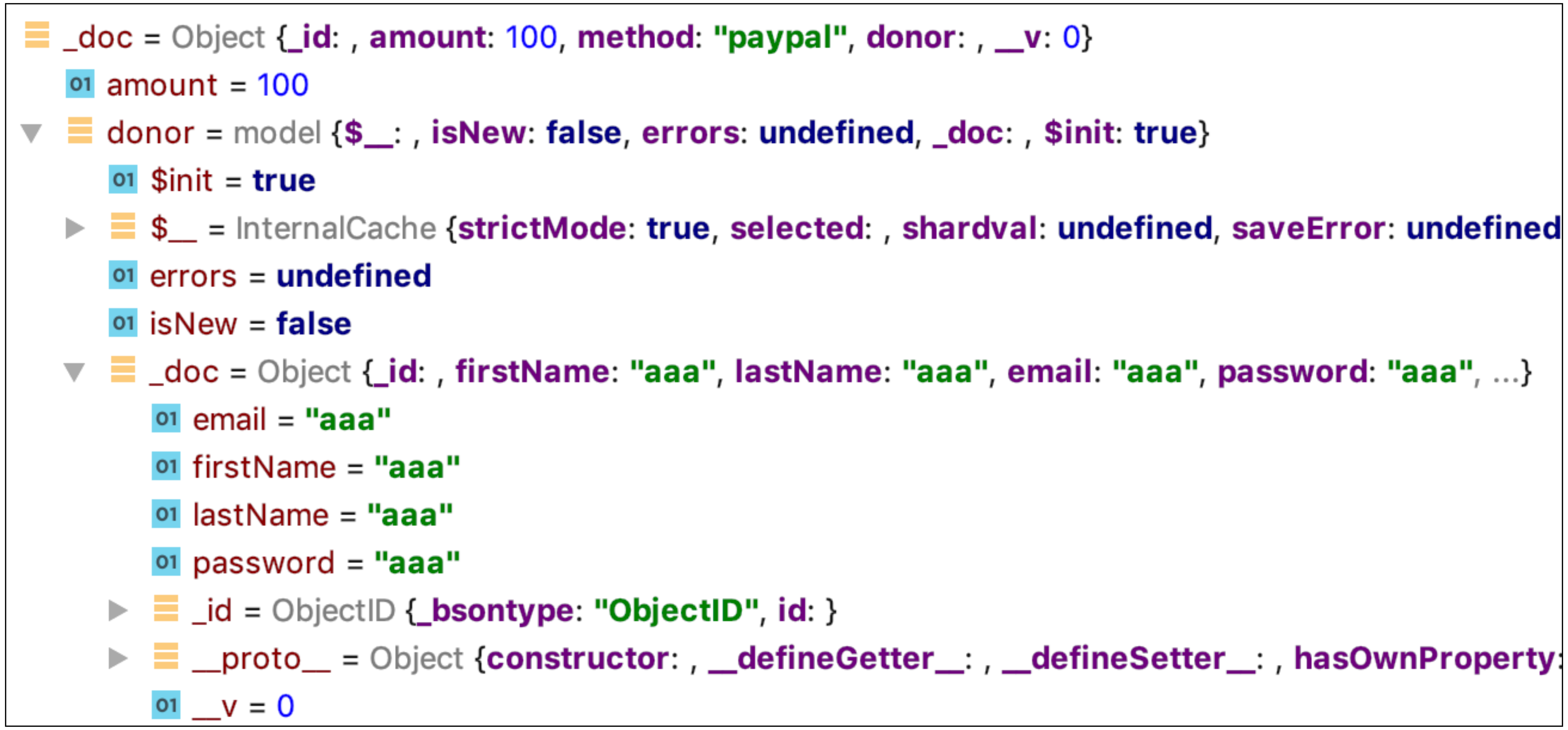
```
report: {
  handler: async function(request, h) {
    const donations = await Donation.find().populate('donor');
    return h.view('report', {
      title: 'Donations to Date',
      donations: donations
    });
  }
},
```

Populate Donor



```
▼ _doc = Object {_id: , amount: 100, method: "paypal", donor: , __v: 0}
    amount = 100
  ▼ donor = ObjectID {_bsontype: "ObjectID", id: }
    ▶ id = Buffer(12) {0: 92, 1: 81, 2: 75, 3: 43, 4: 174, 5: 228, 6: 99, 7: 68, 8: 245, 9: 48, 10: 63, 11: 9
      _bsontype = "ObjectID"
    ▶ __proto__ = Object {toHexString: , get_inc: , getInc: , generate: , toString: , ...}
    method = "paypal"
```

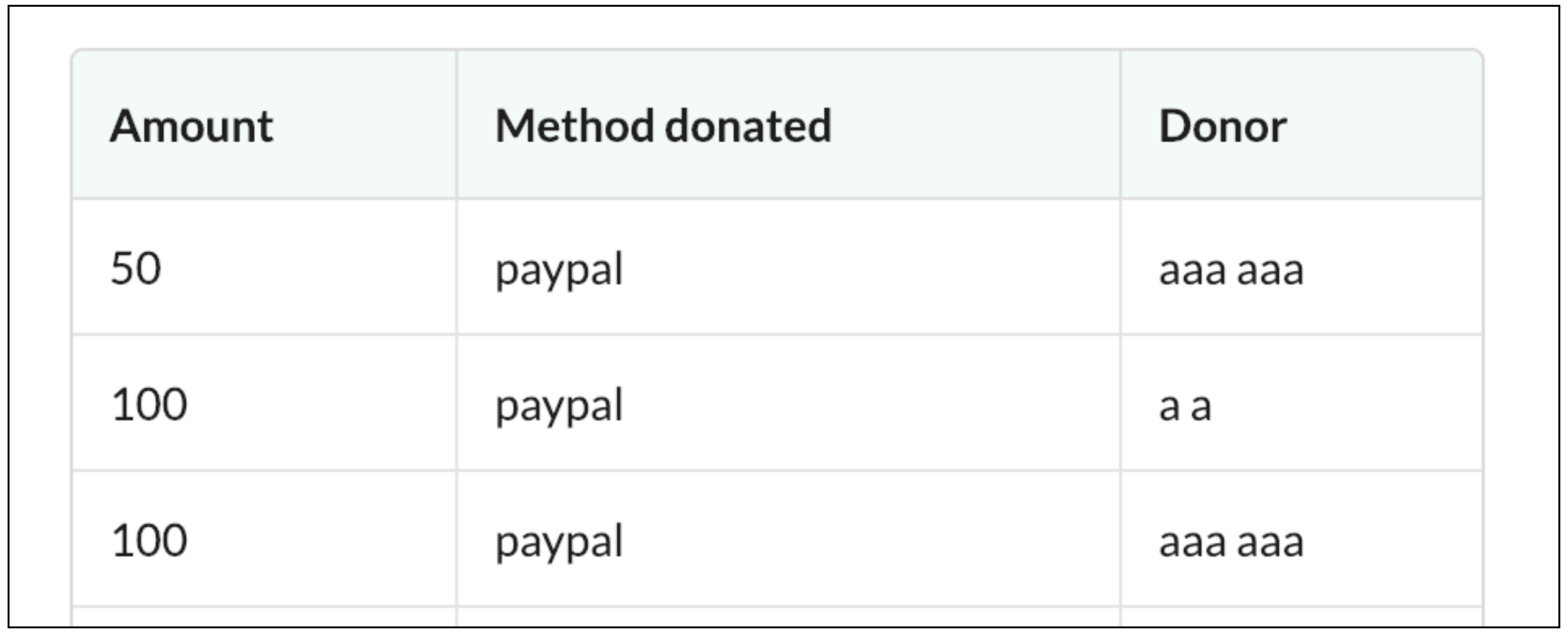
```
Donation.find().populate('donor');
```



```
_doc = Object {_id: , amount: 100, method: "paypal", donor: , __v: 0}
    amount = 100
▼   donor = model {$__: , isNew: false, errors: undefined, _doc: , $init: true}
        $init = true
    ▶   $__ = InternalCache {strictMode: true, selected: , shardval: undefined, saveError: undefined
        errors = undefined
        isNew = false
    ▼   _doc = Object {_id: , firstName: "aaa", lastName: "aaa", email: "aaa", password: "aaa", ...}
            email = "aaa"
            firstName = "aaa"
            lastName = "aaa"
            password = "aaa"
        ▶   _id = ObjectID {_bsontype: "ObjectID", id: }
        ▶   __proto__ = Object {constructor: , __defineGetter__: , __defineSetter__: , hasOwnProperty:
            __v = 0
```

- Donor has complete contents of user object

```
<tbody>
  {{#each donations}}
    <tr>
      <td> {{amount}} </td>
      <td> {{method}} </td>
      <td> {{donor.firstName}} {{donor.lastName}} </td>
    </tr>
  {{/each}}
</tbody>
```



Amount	Method donated	Donor
50	paypal	aaa aaa
100	paypal	a a
100	paypal	aaa aaa

- Subdocument directly available in template

```
'use strict';

require('dotenv').config();

const Mongoose = require('mongoose');

Mongoose.connect(process.env.db);
const db = Mongoose.connection;

db.on('error', function(err) {
  console.log(`database connection error: ${err}`);
});

db.on('disconnected', function() {
  console.log('database disconnected');
});

db.once('open', function() {
  console.log(`database connected to ${this.name} on ${this.host}`);
})
```

```
'use strict';

const Boom = require('boom');
const Mongoose = require('mongoose');
const Schema = Mongoose.Schema;

const userSchema = new Schema({
  firstName: String,
  lastName: String,
  email: String,
  password: String
});

userSchema.statics.findByEmail = function(email) {
  return this.findOne({ email : email});
};


userSchema.methods.comparePassword = function(candidatePassword) {
  const isMatch = this.password === candidatePassword;
  if (!isMatch) {
    throw new Boom('Password mismatch');
  }
  return this;
};

module.exports = Mongoose.model('User', userSchema);
```

Donation Models

```
'use strict';

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

const donationSchema = new Schema({
  amount: Number,
  method: String,
  donor: {
    type: Schema.Types.ObjectId,
    ref: 'User'
  }
});

module.exports = Mongoose.model('Donation', donationSchema);
```

Donation Controller

```
const Donations = {
  home: {
    handler: function(request, h) {
      return h.view('home', { title: 'Make a Donation' });
    }
  },
  report: {
    handler: async function(request, h) {
      const donations = await Donation.find().populate('donor');
      return h.view('report', {
        title: 'Donations to Date',
        donations: donations
      });
    }
  },
  donate: {
    handler: async function(request, h) {
      const id = request.auth.credentials.id;
      const user = await User.findById(id);
      const data = request.payload;
      const newDonation = new Donation({
        amount: data.amount,
        method: data.method,
        donor: user._id
      });
      await newDonation.save();
      return h.redirect('/report');
    }
  }
};
```