**Model One-to-One Relationships**

Assume that we have 2 entity: *Identifier* and *Customer*. For example:

// Identifier

{ \_id: "12345xyz",

cardCode: "BKD2019",

}

// Customer

{ \_id: "cus123",

name: "innouser",

age: 29,

gender: "male"

}

We want to map *Identifier* and *Customer* relationship in that:  
– One *Customer* has only one *Identifier*.  
– One *Identifier* belongs to only one *Customer*.

This is called One-to-One Relationship.  
Now let’s take a look at 2 ways to design schema for this kind of model.

**Reference Data Models (Normalization)**

In this model, an object A connects to the object B by reference to object B id or a unique identification field.

For example, *Identifier* has a customer\_id field which value is equal to *Customer* object’s unique \_id value.

// Identifier

{

\_id: "12345xyz",

cardCode: "BKD2019",

customer\_id: "cus123",

}

// Customer

{

\_id: "cus123", // equal Identifier[customer\_id]

name: "innouser",

age: 29,

gender: "male"

}

**Embedded Data Models (Denormalization)**

It’s easy to understand with ‘Embedded’ word. Instead of using a reference, Object A contains the whole object B, or object B is embedded inside object A.

You can see the example below, *Identifier* will have a nested field customer.

// Identifier

{

\_id: "12345xyz",

cardCode: "BKD2019",

customer: {

\_id: "cus123", // Identifier[customer\_id]

name: "innouser",

age: 29,

gender: "male"

}

}

You’ve known 2 ways to make One-to-One Relationships. Let’s implement each of them in a Node.js app using Mongoose. After that, I will tell you which model should be used for implementing One-to-One relationship between collections in MongoDb Database.

**Mongoose One-to-One relationship example**

**Setup Nodejs App**

First we need to install **mongoose**, so run the command:

npm install mongoose

Next, we create project structure like this:

 src

 models

 Customer.js

 Identifier.js

 server.js

 package.json

Open *server.js*, we import mongoose to our app and connect to MongoDB database.

const mongoose = require("mongoose");

mongoose

.connect("mongodb://localhost/innouser\_db", {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => console.log("Successfully connect to MongoDB."))

.catch(err => console.error("Connection error", err));

The first step is done, we’re gonna create appropriate models and use mongoose to interact with MongoDB database in two ways:

* Referencing
* Embedding

**Mongoose One-to-One relationship: Referencing**

**1. Define One-to-One models using Mongoose**

Let’s create 2 main model with mongoose.Schema() construtor function.

In *models/Customer.js*, define Customer with 3 fields: name, age, gender.

const mongoose = require("mongoose");

const Customer = mongoose.model(

"Customer",

new mongoose.Schema({

name: String,

age: Number,

gender: String

})

);

module.exports = Customer;

Identifier object will have cardCode field and a reference customer.  
So open *models/Identifier.js*, define Identifier like this:

const mongoose = require("mongoose");

const Identifier = mongoose.model(

"Identifier",

new mongoose.Schema({

cardCode: String,

customer: {

type: mongoose.Schema.Types.ObjectId,

ref: "Customer"

}

})

);

module.exports = Identifier;

In the code above, we add customer field, set its type to ObjectId and ref to Customer. What does it help?

Now if we save an Identifier to MongoDB database, a document will be added like this:

{

\_id : ObjectId("5da000be062dc522eccaedeb"),

cardCode : "5DA000BC06",

customer : ObjectId("5da000bc062dc522eccaedea"),

\_\_v : 0

}

Let’s test it, and I will show you how to get an Identifier object with full-fields Customer in this approach.

**2. Test with MongoDB database**

Open *server.js*, add the code below:

const Customer = require("./models/Customer");

const Identifier = require("./models/Identifier");

const createCustomer = function(name, age, gender) {

const customer = new Customer({

name,

age,

gender

});

return customer.save();

};

const createIdentifier = function(cardCode, customer) {

const identifier = new Identifier({

cardCode,

customer

});

return identifier.save();

};

createCustomer("innouser", 29, "male")

.then(customer => {

console.log("> Created new Customer\n", customer);

const customerId = customer.\_id.toString();

return createIdentifier(customerId.substring(0, 10).toUpperCase(), customerId);

})

.then(identifier => {

console.log("> Created new Identifier\n", identifier);

})

.catch(err => console.log(err));