You need to consider two factors:

* Will the entities on the “N” side of the One-to-N ever need to stand alone?
* What is the cardinality of the relationship: is it one-to-few; one-to-many; or one-to-squillions?

Based on these factors, you can pick one of the three basic One-to-N schema designs:

* Embed the N side if the cardinality is one-to-few and there is no need to access the embedded object outside the context of the parent object
* Use an array of references to the N-side objects if the cardinality is one-to-many or if the N-side objects should stand alone for any reasons
* Use a reference to the One-side in the N-side objects if the cardinality is one-to-squillions

I’ve covered the additional choices that you have past the basics of embed, child-reference, or parent-reference.

* You can use bi-directional referencing if it optimizes your schema, and if you are willing to pay the price of not having atomic updates
* If you are referencing, you can denormalize data either from the “One” side into the “N” side, or from the “N” side into the “One” side
* When deciding whether or not to denormalize, consider the following factors:
* You cannot perform an atomic update on denormalized data
* Denormalization only makes sense when you have a high read to write ratio

Here are some “rules of thumb” to guide you through these indenumberable (but not infinite) choices

**One:** favor embedding unless there is a compelling reason not to

**Two:** needing to access an object on its own is a compelling reason not to embed it

**Three:** Arrays should not grow without bound. If there are more than a couple of hundred documents on the “many” side, don’t embed them; if there are more than a few thousand documents on the “many” side, don’t use an array of ObjectID references. High-cardinality arrays are a compelling reason not to embed.

**Four:** Don’t be afraid of application-level joins: if you index correctly and use the projection specifier (as shown in part 2) then application-level joins are barely more expensive than server-side joins in a relational database.

**Five:** Consider the write/read ratio when denormalizing. A field that will mostly be read and only seldom updated is a good candidate for denormalization: if you denormalize a field that is updated frequently then the extra work viof finding and updating all the instances is likely to overwhelm the savings that you get from denormalizing.

**Six:** As always with MongoDB, how you model your data depends – entirely – on your particular application’s data access patterns. You want to structure your data to match the ways that your application queries and updates it.

# Functies

## Coa

function insertCoa ( naam, commandant, starttijd, eenheid, lat, long ) {   
db.coa.insert({   
coa\_naam: naam,   
commandant\_naam: commandant,   
start\_tijd: starttijd,   
eenheidsnaam: eenheid,   
scenario\_lat: lat,   
scenario\_long: long, });   
}

> insertCoa("Test COA 2", "Peter John", new Date(), "Test Unit", 1.3264, 5.3321)  
> insertCoa("Test COA 1", "Pete Johnson", new Date(), "Test", 0.1234, 4.3211)

## Hoofdvak

function insertHoofdvak(lat, long, titel, risiconiveau, modus) {  
db.hoofdvak.insert({  
hoofdvak\_lat: lat,   
hoofdvak\_long: long,   
hoofdvak\_titel: titel,   
risico\_niveau: risiconiveau,   
modus: modus}) ;   
}

> insertHoofdvak(1.12313, 2.1312, "Test hoofdvak 1", 0, 1)  
> insertHoofdvak(11.11213, 24.11231, "Test hoofdvak 2", 4, 0)  
> insertHoofdvak(9.11213, 5.11231, "Test hoofdvak 3", 4, 0)

## Hoofdvak in COA

function addHoofdvakToCoa(coa, hoofdvak) {   
db.coa.update( {"\_id":ObjectId(coa)}, { $push: {  
"hoofdvak": {$ref: "hoofdvak", $id:hoofdvak}}});  
}

addHoofdvakToCoa("5eb5b3a276bf3e7ac02f2a07","5eb5b4a176bf3e7ac02f2a09")  
addHoofdvakToCoa("5eb5b34276bf3e7ac02f2a06","5eb5b49276bf3e7ac02f2a08")  
addHoofdvakToCoa("5eb5b3a276bf3e7ac02f2a07","5eb5b61e9d42a5b422b0bc5c")

## Focusvak in hoofdvak

function insertFocusvak(hoofdvak, id, lat, long, titel, risiconiveau) {   
db.hoofdvak.update( {"\_id": ObjectId(hoofdvak)}, {$push: {  
"focusvak": {   
focusvak\_id: id,   
focusvak\_lat: lat,   
focusvak\_long: long,   
focusvak\_titel: titel,   
risico\_niveau: risiconiveau}}});   
}

> insertFocusvak("5eb5b61e9d42a5b422b0bc5c", 1, 0.1231, 1.1211, "focusvak 1 van hoofdvak 3", 0)  
> insertFocusvak("5eb5b61e9d42a5b422b0bc5c", 2, 0.1231, 1.1211, "focusvak 2 van hoofdvak 3", 0)  
> insertFocusvak("5eb5b4a176bf3e7ac02f2a09", 1, 0.1231, 1.1211, "focusvak 1 van hoofdvak 2", 0)

## Drone

function insertDrone(type) {   
db.drone.insert({   
drone\_type: type,   
});}

> insertDrone("helicopter")  
> insertDrone("quadcopter")

## Drone to COA

function addDroneToCoa(coa, drone, hoofdvak, start\_lat, start\_long) { db.coa.update( {"\_id":ObjectId(coa)}, {$push: {"drone": { drone\_id: {$ref: "drone", $id:drone}, start\_positie\_lat: start\_lat, start\_positie\_long: start\_long, hoofdvak: {$ref: "hoofdvak", $id:hoofdvak} }}});}

> addDroneToCoa("5eb5b34276bf3e7ac02f2a06", "5eb5b7669d42a5b422b0bc5d", "5eb5b49276bf3e7ac02f2a08", 0.0011, 1.0111)  
> addDroneToCoa("5eb5b34276bf3e7ac02f2a06", "5eb5b76d9d42a5b422b0bc5e", "5eb5b49276bf3e7ac02f2a08", 1.0011, 1.0121)

## Uitvoering

function insertUitvoering(drone, lat, long, batterij) { db.uitvoering.insert({ tijd: new Date(), drone\_id: {$ref:"drone", $id:"5e8e1be07260724202cbf387"}, drone\_lat: 0.1213, drone\_long: 3.3123, batterij\_duur: 100 } );}