* Users & reviews – This is a one to many relationship, because one user can write multiple reviews but one review can only belong to one user. The parent in this relationship is clearly a users and the child is the reviews. And that’s why I choose to model this relationship using parent referencing, and that’s because a user can write a lot of reviews and also because we might also query only for the reviews on there own. So the data access pattern is very important to take into consideration in this particular relationship , so the review keeping the reference of the user, keeping an id basically. And because we do not want to allow an array to grow definitely.
* Tours and reviews – so again it’s a one to many relationship, where one tour can have multiple reviews, but one review can only be about one tour, so we are going to model it the exact same way as the users and reviews relationship. So again a reviews ending with a tourid and a userid, so once we query reviews we always known exactly.
* Tours and locations – each tour can have couple of locations, so for example a park camper basically have a 3 or 4 national parks so each of this national parks is Gona be a one location, so an each tour can have few locations now following that example one of these national park might also a part one of the other tours and so basically this relationship here is a few to few relationship, we can call this as a few to few or a ton to ton, I call few to few because each tour is only gona have 3,4 locations and not really like a hundred and again each of the locations can also be a part of another locations, now this could be a good example of implementing two way referencing so basically normalising the locations into its own datasets, but instead I am actually doing de normalise the locations, so to embed then into the tours, and that’s actually for multiple reasons

1. Because they actually have so few locations
2. Also we are not going to access the locations on there own.
3. And finally these locations are intrinsically related to the tours because really without locations there couldn’t be any tours.

And that is why I choose embed locations into the tours and not create yet another collection for these.

So, we will have one collection for tours, one for users, and a bit later we also create a new collection for the reviews. But not for locations again because this will be embedded into the tours.

* Tours and users – and that’s because we’re gonna have tour guides in the tours, and these guides will actually be users. So remember how we actually gave users a role, in our mongoose schema and the possibilities there contained the guide and lead guide, remember? And so there’s gonna be a relationship between these types of users and the tours. And this relationship is again a few to few relationship because one tour can have only a few users, so a few tour guides, but at the same time each tour guide can also be guiding a few tours. Alright and so again there a many to many relationship here, which I simply called here few to few.

Now about actually modelling this relationship, we could do it in two ways. We could use referencing or embedding and actually I am going to implement both child referencing and embedding using mongoose throughout this section. And the argument for embedding is that in this case we could then have all the information about each tour containing the information about tour guides right on each tour document, but in the other hand that would then create some extra information in the database because we will still need to have the users as a separate collection simply because we need to access them all the time for user authentication and authorization and all that stuff so usually users are always an entity on their own in each database. But we could still embed some of the users into the tours, so basically when the user is a tour guide for a specific tour, we could then copy all this data into the tour document. But also we would then have to update the user on the tour each time that the underlying user itself changes. So lets say the role of the user changes, so lets say that the role of a user changes from guide to a lead guide. And that case we will have to go the tour and also update that role information right there on the embedded data. And so that’s not ideal and so we’re actually also goanna then implement chid referencing and so with that we can still keep basically the information about the tour guides on the users but simply in a referenced form. So basically, keeping the id’s there which are then gonna point to the users. And of course we could also use two-way referencing so also keeping an ID of the tour right on the user, but I think that’s a bit too much for this kind of small example. Because not all users will actually need an id of the tour because all users are tour guides. And so this relationship here is a bit tricky to model I think but I believe that in the end child referencing is gonna be the best way to go.

* Next we have a booking, and basically a new booking will be created each time that a user purchases a tour. So this is still kind of a relationship between users and tours because again it’s a user who is gonna buy a tour. But we also want to store some data about that relationship itself, so this case about the purchase itself in our database. For example, the price or the date when the purchase happened or something like that. So it’s a good idea to create an extra dataset, which in this case is the bookings and so of course there will be a relationship between tours and bookings and also users and bookings and again because basically the booking connects tours with users but kind of with an intermediate step.

so one tour can have many bookings, but one booking can only belong to one tour, and the same thing with the users. So one user can book many tours, but one booking can only belong to one of the users. And so off course we have a one-to-many relationship in both cases and also in both cases, we’re gonna use parent referencing. And so that means on each booking we are gonna keep an id of both tour that was purchased and also of the user who actually purchased the tour. And in this case I am doing in this way basically I don’t want to pollurte the tour data with information about who actually bought the tour. It would’nt be really relevant to the tour data itself. And the same thing with users. So we also don’t want to pollute the users object with all of the bookings that they did. And instead we’re gonna create an intermediate object or an intermediate dataset that’s going to stand between users and tours whenever they create a new purchase.

