Use mongoDB `firstDB` separately. Means clear DB before using any of the file i.e. ‘embedding.js’ or ‘population.js’

Trade off between query performance vs consistency

1 - Using reference (Normalization) -> Consistency

```
let author = {
 name: "Mutahhir",
 //50 other properties
}


let course = {
 author: 'id',
 name: "Learn JavaScript",
}
```

2 - Using Embedded documents (Denormalization) -> Performance

```
let course = {
 name: "Learn JavaScript",
 author: {
  name: "Mutahhir",
  //50 other properties
 }
}
```

3 - Hybrid

```
let course = {
name: "Learn JavaScript",
author: {
 id: 'ref',
 name: "Mutahhir",
}
}
```

Population

```
const course = await Course.find().populate('author', 'name -_id').select('name
author');
```

Here minus sign before the attribute points out that those fields have to be filtered out.
As we don't have relationships in Document based databases, we use populate.
Before this we have the reference field to the parent object. E.g.

```
const Course = mongoose.model('Course', new mongoose.Schema({
name: String,
 author: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Author'
}
}));

const Author = mongoose.model('Author', new mongoose.Schema({
 name: String,
 bio: String,
 website: String
}));
```

Embedded

Embedded documents are sub-documents or normal documents.
These documents can not be saved on their own but they can only be saved on the context of their parents.

```
/**
* @NOTICE here author document is embedded in the course document.
* it can only be save through the course document. can not access or save it
directly.
*/
//for single author document
const authorSchema = new mongoose.Schema({
name: String,
bio: String,
website: String,
});

//for single author document
const Author = mongoose.model("Author", authorSchema);

//for single author document
const Course = mongoose.model(
"Course",
new mongoose.Schema({
 name: String,
 author: authorSchema, //directly embedding the complete authorSchema
})
);
```