# Relationships

MONGOOSE

# Using References (Normalization)

```
let author = {
 name: "Usman Akram"
}
let course = {
 title: 'Web Technologies',
 author: 'id'
}
```

# Using Embedded Documents (Denormalization)

```
let course1 = {
 title: 'Web Technologies',
 author: {
  name: "Usman Akram"
 }
}
```

# Trade Off between Query Performance Vs Consistency

## NORMALIZATION

A change in author would reflect every where

More Consistent but need extra query to get child records

## DE NORMALIZATION

If you need to change the author you will have to modify in multiple records

Not Consistent but More Performance

# Hybrid Approach

```
let author = {
name: "Usman Akram"
// 50 More properties
}
```

```
let course = {
 title: 'Web Technologies',
 author: {
  name: "Usman Akram",
  id: 'reference to author'
 }
}
```

// Copy id and some of specific properties. Like facebook top comment should be beside post

# Mongoose Recap and Code for This section

https://1drv.ms/f/s!AtGKdbMmNBGd0ia66DfXulxFIM6m

```
const mongoose = require('mongoose');
//npm init –yes
//npm install mongoose
mongoose.connect('mongodb://localhost/playground')
.then(() => console.log('Connected to MongoDB...'))
.catch(err => console.error('Could not connect to MongoDB...', err));
```

# Author

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

# Course

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

# Create Author

```
async function createAuthor(name, bio, website) {
  const author = new Author({
    name,
    bio,
    website
  });
  const result = await author.save();
  console.log(result);
}
```

# Create Course

```
async function createCourse(name, author) {
  const course = new Course({
    name,
    author
  });
  const result = await course.save();
  console.log(result);
}
```

# List Courses

```
async function listCourses() {
  const courses = await Course
    .find()
    .select('name');
  console.log(courses);
}
```

# Actually Create an Author

createAuthor('Mosh', 'My bio', 'My Website');

createCourse('Web Technologies', '5c3e1c1d8b3a1c1d0c1ac03d')

// whatever id the create course has given to new doc put that in create Course

# Author Doc


```
{
 "_id":"5c3e1e0734ea7010842c96df",
 "name":"Web Technologies",
 "author":"5c3e1c1d8b3a1c1d0c1ac03d",
 "__v":0
}
```

# Population

```
async function listCourses() {
const courses = await Course
.find()
.populate('author')
.select('name author');
console.log(courses);
}
```

# Author populated

```
{ _id: 5c3e1e0734ea7010842c96df,
   name: 'Web Technologies',
   author:
    { _id: 5c3e1c1d8b3a1c1d0c1ac03d,
      name: 'Usman',
      bio: 'My bio',
      website: 'usmanlive.com',
      __v: 0 }
}
```

# Population only get author name

```
async function listCourses() {
const courses = await Course
.find()
.populate('author', 'name')
.select('name author');
console.log(courses);
}
```

# Population only get author name -id

```
async function listCourses() {
const courses = await Course
.find()
.populate('author', 'name -_id')
.select('name author');
console.log(courses);
}
```

# Multiple Populations

```
async function listCourses() {
const courses = await Course
.find()
.populate('author', 'name -_id')
.populate('category', 'name')
.select('name author');
console.log(courses);
}
```

# What if the author ID is changed

E.g. Course has an author ID which Actually doesn’t exist in author collection

-Will Mongo Complain –No

-What will return –Null

# Embedding Documents (Course has One Author)

```
//Define a Schema
 const authorSchema = new mongoose.Schema({
  name: String,
  bio: String,
  website: String
 });
```

# Embed Auth Schema in Course Schema

```
const Course = mongoose.model(
"Course",
 new mongoose.Schema({
 name: String,
 author: authorSchema
 })
);
```

# Mongo Data Shape


{

"_id":"5c51d826e53bf91d5c6c3560",

"name":"Node Course 1",

"author":

**{"_id":"5c51d826e53bf91d5c6c355f","name":“Usman"},**

"__v":0

}

# Updating Embedded Doc

```
async function updateCourse(id) {
  const course = await Course.findById(id);
  course.author.name = "Hareem";
  course.save();
  //course.author.save();
}
```

# Update Embedded Doc with Single Query

```
async function updateCourseSingleQuery(id) {
const course = await Course.update({ _id: id }, {
 $set: {
  "author.name": "Hareem Fatima"
 }
});
course.save();
}
```

# Removing Embedded Doc

```
async function removeAuthor(id) {
const course = await Course.update({ _id: id }, {
 $unset: {
 "author":""
 }
});
course.save();
}
```

# Embedded Doc is a Doc ☺

```
new mongoose.Schema({
 name: String,
 author: {
  type: authorSchema,
  required: true
 }
})
```

# An Array of Sub Docs (Many Authors)

```
new mongoose.Schema({
  name: String,
  authors: [authorSchema]
})
```

# Create Course With Many Authors

```
createCourse("Web Tech", [
 new Author({ name: "Usman" }),
 new Author({ name: "Hareem" })
]);
//course.authors.push(author) to add
//No Change in Create Course Function
```

# Add Author

```
async function addAuthor(courseID,author){
  const course = await
Course.findById (courseID);
  course.authors.push(author);
  course.save();
}
```

# Remove Author

```
async function removeAuthor(courseID,authorID){
const course = await Course.findById(courseID);
const author = course.authors.id(authorID);
author.remove();
course.save();
}
```

# Project Movies API Exercise

https://1drv.ms/f/s!AtGKdbMmNBGd0ja8VzM2bdZ43qtK

# Movies Rental API Exercise

https://1drv.ms/f/s!AtGKdbMmNBGd0iWySSLjdDZA565v

Create a New Rental

// POST  /api/rentals

Get the List of Rentals

//GET /api/rentals

# Transactions in Mongo DB ? ☹

Two Phase Commit

npm install fawn

# Object ID (24 char or 12 Bytes)

5c3e1e0734ea7010842c96df

16 Million docs per millisecond per machine per process

4-bytes: timestamp

3-bytes: machine identifier

2-bytes: process identifier

3-bytes: counter

# Performance ?

Mongo DB Driver (mongoose) can generate the object ID

No need to consult Mongo DB for id

```
const id = mongoose.Types.ObjectId()
console.log(id.getTimestamp())
```

//mongoose.Types.ObjectId.isValid(456); ?

# ReCAP

https://1drv.ms/b/s!AtGKdbMmNBGd0wuTPreLe8KsrlNR