

Fullstack-разработка

Mongo

Отношения

Часть 1

Цель урока:

- Разобраться, как данные в БД связаны между собой
- Попробовать 2 способа создания связей

Теория

Отношения

Мы уже умеем выполнять
простейшие операции над
данными в БД:

- Находить
- Добавлять
- Удалять
- Изменять

Вот только все эти операции мы
делали, работая с одной
единственной коллекцией...

и очень простой структурой
документа.

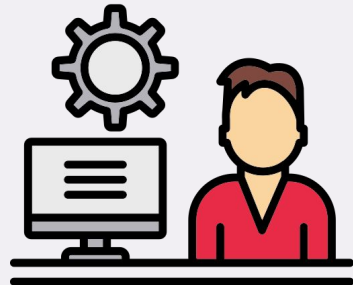


В больших приложениях участвуют большие данные.

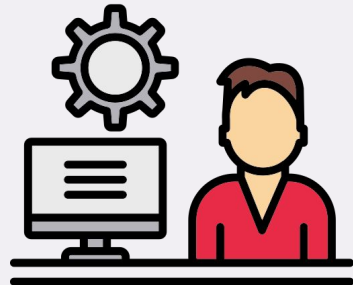
Архитектор БД – это человек, который продумывает:

1. **Какие данные нужны?**
2. **Как организовать структуру БД?**

Какие данные нужны
для онлайн-кинотеатра?

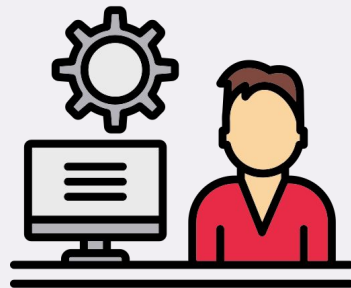


Какие способы
организовать хранение
данных мы знаем?



1. JSON
2. Коллекции

Для выбора правильного
способа нужно понять,
как данные связаны
между собой.

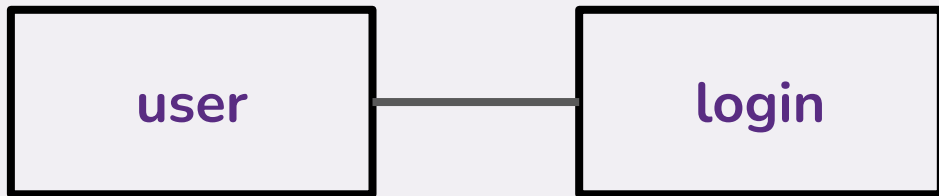


Связи между данными
называются **отношениями**.

1. Один к одному
2. Один ко многим
3. Многие ко многим

Один к одному

Пример: пользователь и его логин

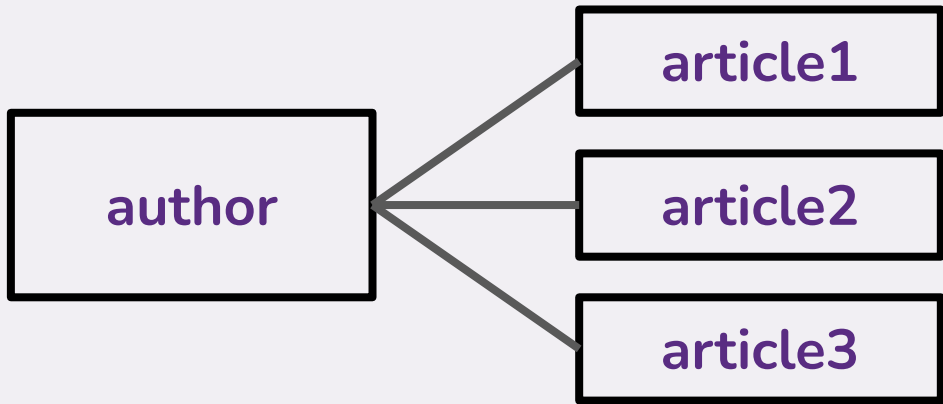


У пользователя может быть только 1 логин.

Логин может быть указан только у одного пользователя.

Один ко многим

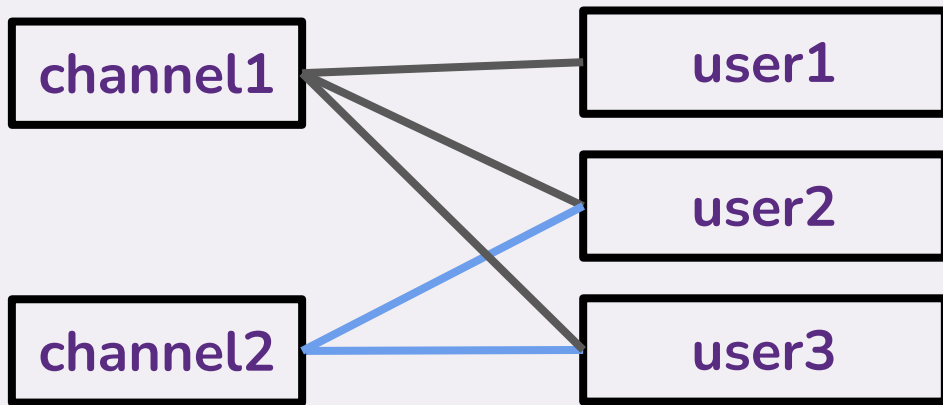
Пример: **блогер** и его статьи



*Блогер может написать множество разных статей.
У каждой статьи будет только один автор.*

Многие ко многим

Пример: каналы и их подписчики

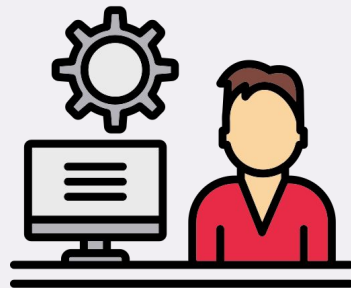


У одного канала может быть много подписчиков

Один подписчик может читать много каналов

На платформе есть
фильмы, зрители и
комментарии.

Как они связаны между
собой?



Фильмы

Комментарии

Юзеры

Когда архитектор ответил на вопросы о связях, он может подумать о реализации.

Один к одному

Пример: **пользователь** и его **логин**

```
let userSchema = new mongoose.Schema({  
  login: String,  
  age: Number,  
  isBlogger: Boolean  
});
```

Это способ, которым мы пользовались уже
десяток раз – обычный JSON документ
с понятной схемой.

Один ко многим

Пример: блогер и его статьи

```
let authorSchema = new mongoose.Schema({  
  name: String,  
  surname: String  
});  
  
let Author = mongoose.Model(`author`,  
  authorSchema);
```

С автором всё просто – разместим его в отдельной коллекции.

Один ко многим

Пример: блогер и его статьи

```
let articleSchema = new mongoose.Schema({  
  title: String,  
  author_id: mongoose.ObjectId  
});  
  
let Article = mongoose.Model(`article`,  
  articleSchema);
```

А вот статья будет **ссылаться**
на своего автора по его **id**!

Один ко многим

Пример: **писатель** и его **книги**

Таким образом, если мы будем знать **_id** автора, то через метод **find** сможем найти все его книги.

```
Article.find({author_id: '12345678'})
```

Многие ко многим

Пример: каналы и их подписчики

Это самый сложный вид отношений, мы подробно рассмотрим его на следующем занятии.

А сейчас...

Практика

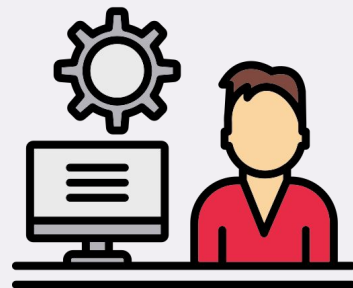
Фильмопоиск 1.0

Теория

Вложенные массивы

Как добавить в наше
приложение
жанр фильма?

Создавать новую
коллекцию каждый раз,
когда нужно "привязать"
новые данные –
не оптимально!



Вложенные массивы

В MongoDB внутри документа можно создавать поля, содержащие **массивы**.

```
{  
  username: 'Тестер',  
  age: 14,  
  colors: ['red', 'green', 'blue']  
}
```

Схема массива

Внутри схемы массивы описываются
как **[тип данных]**

```
let userSchema = new mongoose.Schema({  
  username: String,  
  age: Number,  
  colors: [String]  
});
```

Изменение массива

```
let user = await User.findOne(...);  
  
user.colors[0] = 'pink';  
user.colors.push('yellow');  
user.colors.splice(5, 1);  
  
// После изменений сохрани модель  
await user.save();
```

Поиск по массиву

С помощью простого условия поиска можно найти всех пользователей, у которых в массиве **colors** встречается строка **red**.

```
let users = await User.find({  
  'colors': 'red'  
});
```

Вложенные массивы

Массивы с простыми типами данных подходят для отношений **один и многие ко многим**.

```
{  
    // один ученик – много оценок  
    grades: [2, 3, 3, 4, 3, 5, 5],  
  
    // много учеников – много курсов  
    courses: ['front', 'back']  
}
```

Как добавить в наше
приложение
актёров, про каждого
из которых известны
имя и роль?

Вложенные массивы

Вместо отдельной коллекции можно создать **вложенный массив объектов**, где у каждого объекта есть свой **_id**.

```
{
  _id: '12345',
  name: 'IT блогер',
  articles: [
    {title: '...', _id: '...'},
    {title: '...', _id: '...'},
    {title: '...', _id: '...'}
  ]
}
```

Вложенные массивы

Сначала создадим схему, которая описывает объекты внутри массива.

```
let articleSchema = new mongoose.Schema({  
  title: String,  
  content: String  
});
```

Важно: нет коллекции = нет модели

```
let Article = mongoose.Model(`article`,  
  articleSchema);
```

Вложенные массивы

Затем используем схему в качестве
типа данных для элементов массива.

```
let authorSchema = new mongoose.Schema({  
  name: String,  
  articles: [articleSchema]  
});  
  
let Author = mongoose.Model(`author`,  
  authorSchema);
```

Изменение массива объектов

```
let author = await Author.findOne(...);  
  
author.articles[0].title = 'Статья 1';  
author.articles.push({  
  title: 'Статья 2'  
});  
author.articles.splice(5, 1);  
  
// После изменений сохрани модель  
await author.save();
```

Поиск по массиву объектов

```
let author = await Author.find({  
  'articles.title': 'Статья 1'  
});
```

С помощью такого условия можно найти **автора**, у которого в массиве **articles** есть объект со свойством **title = Статья 1**.

Найти саму статью через **find** нельзя!

Практика

Фильмопоиск 2.0

Практика

Итоги

Когда использовать связанные коллекции?

1. Если данных **много**
2. Если нужен доступ к каждой коллекции **по отдельности**
3. Если данные **часто** редактируют

Когда использовать вложенные массивы?

1. Если данных **немного**
2. Если они часто нужны **одновременно** с документом
3. Если их **редко** редактируют

Цель урока:

- ✓ Разобраться, как данные в БД связаны между собой
- ✓ Попробовать 2 способа создания связей

Домашнее задание