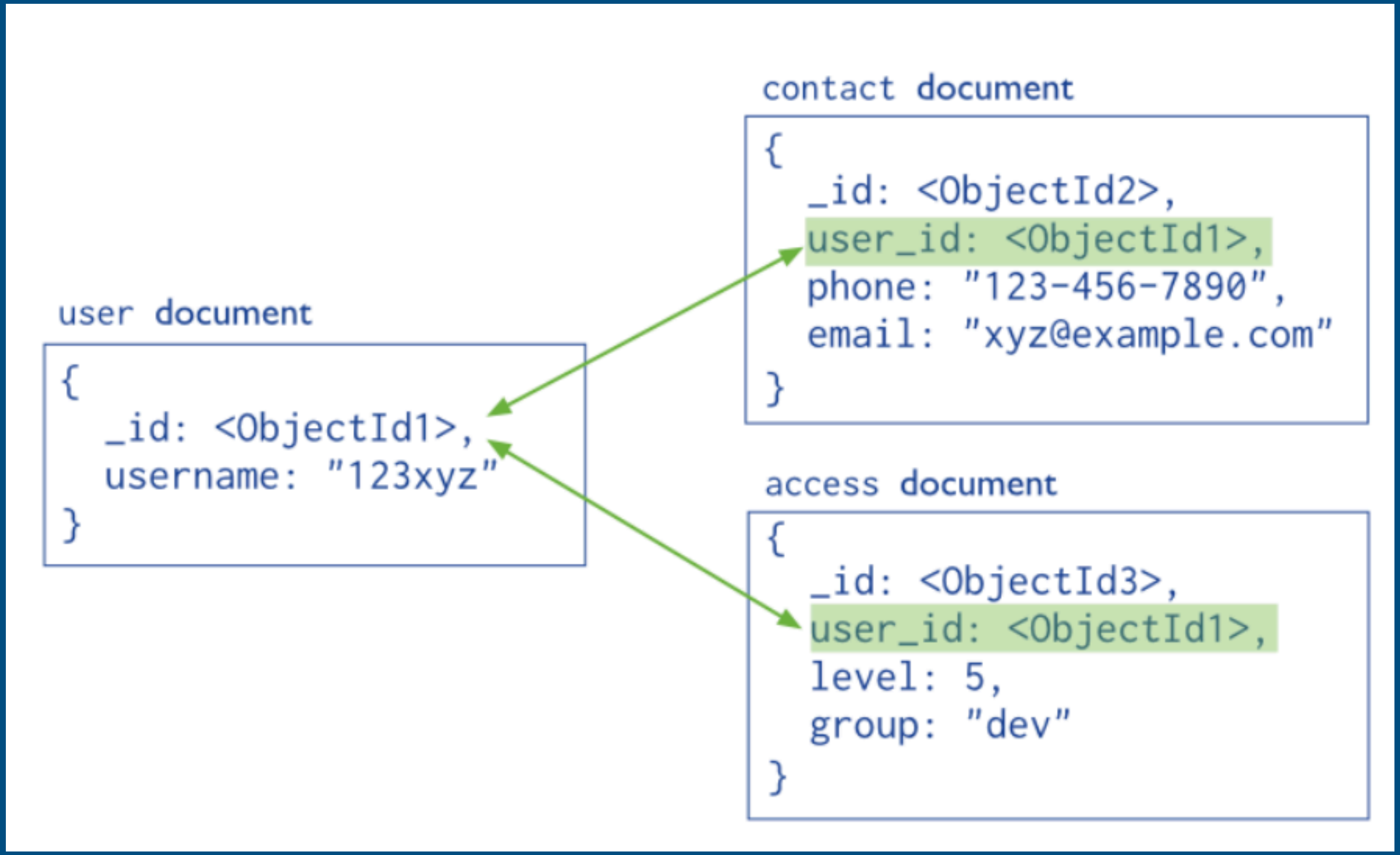
Mongo References



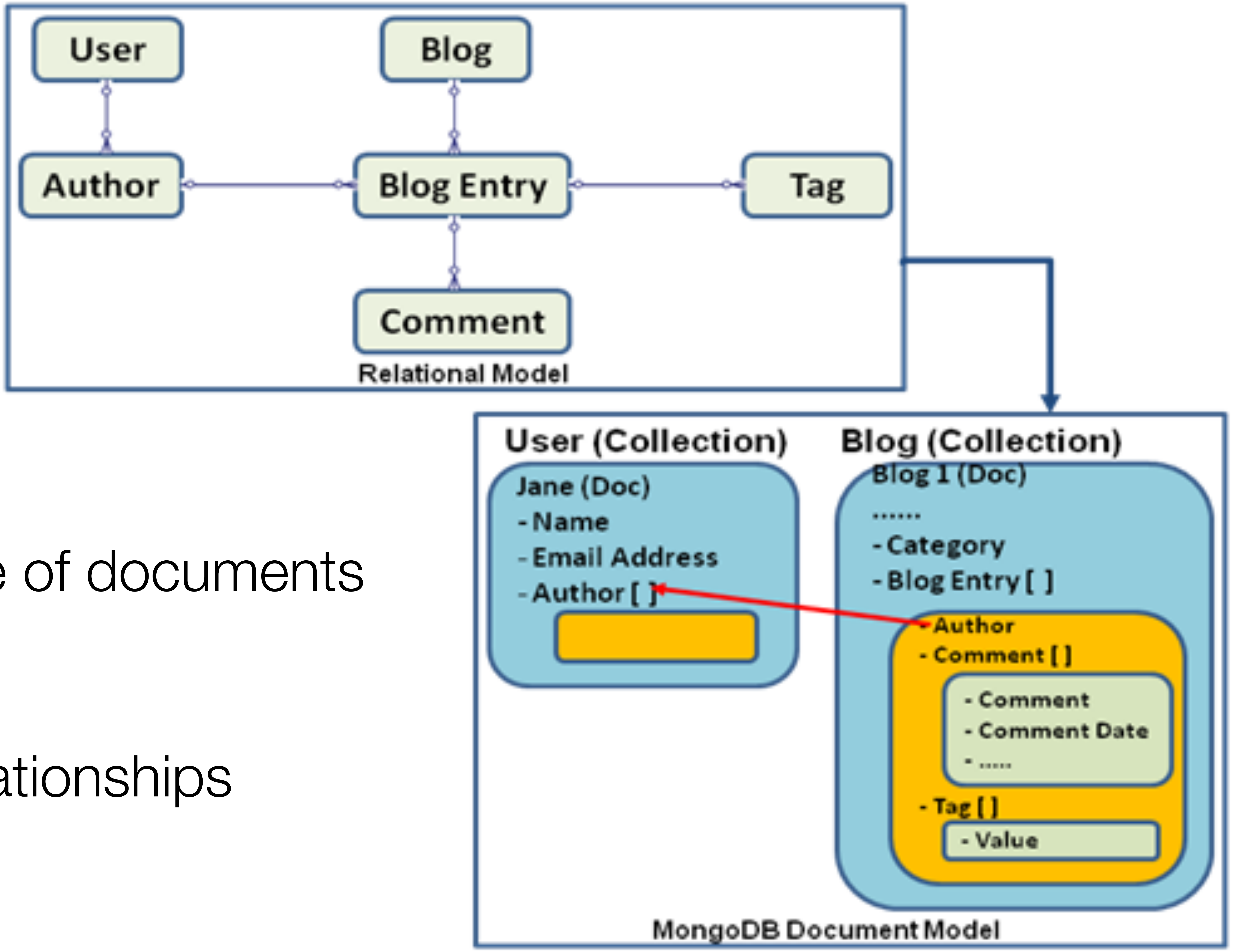
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Full Stack Web Development

Embed vs References



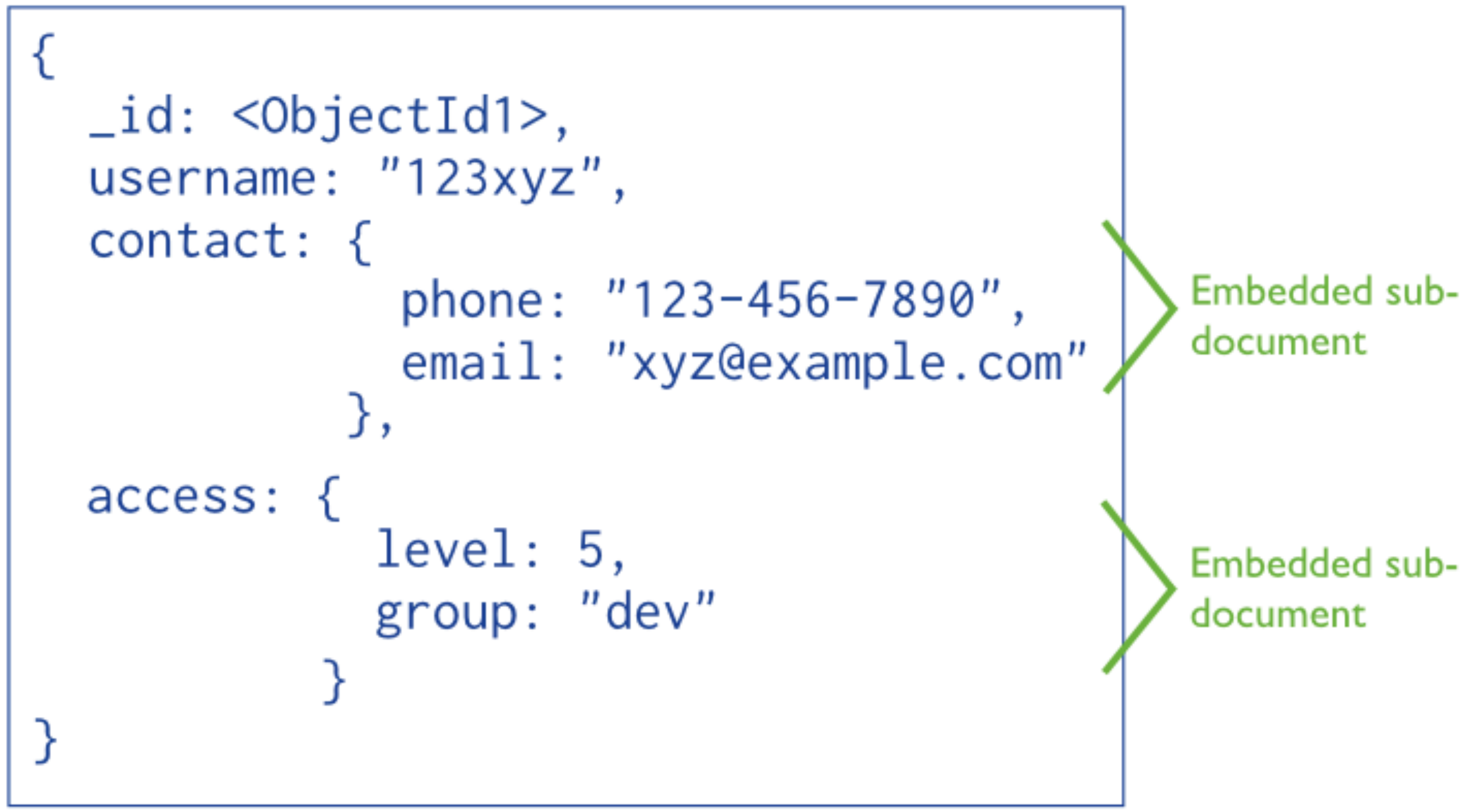
- A key consideration for the structure of documents is the decision to :
  - Embed objects to encapsulate relationships

OR

  - Use object references to encapsulate relationships

Embedded Data Models

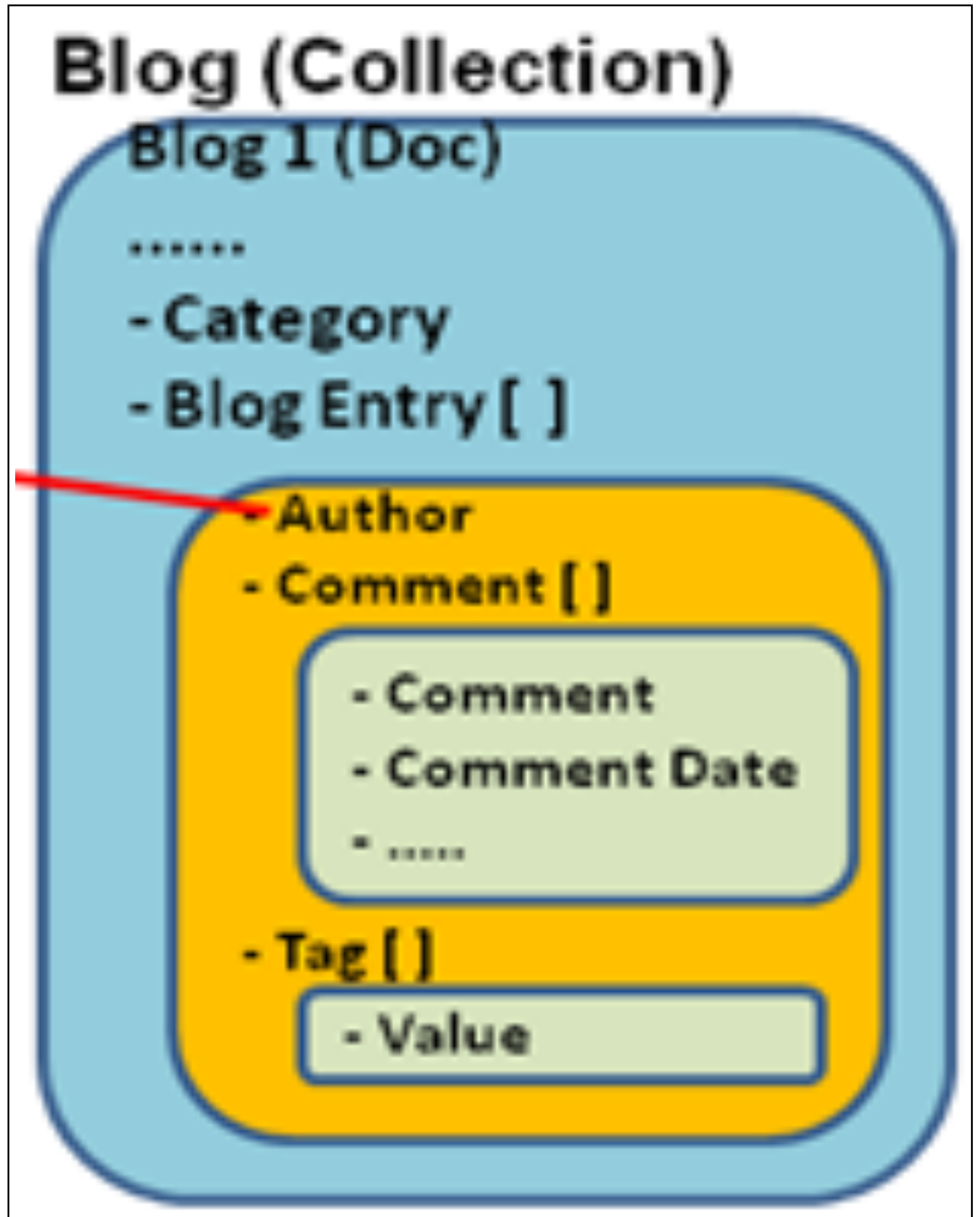
- Embed related data in a single structure or document.
- Generally known as "denormalized" models
- Allow applications to store related pieces of information in the same database record.
- Applications may need to issue fewer queries and updates to complete common operations.



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

When to use Embedded Models?

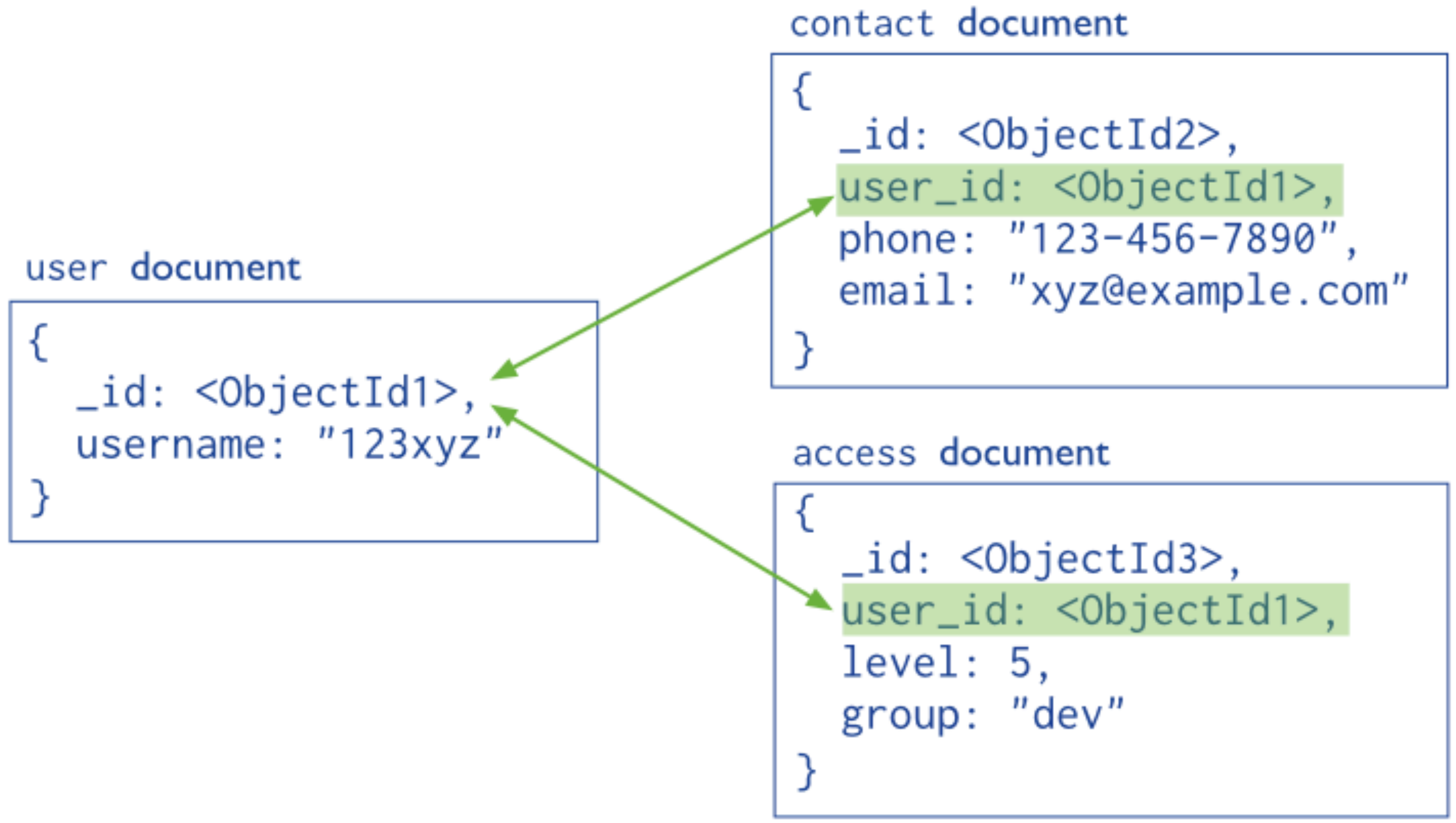
- The "contains" relationships between entities (One-to-One Relationship)
- Some one-to-many relationships between entities - particularly where the "many" (the child document) always appears in the context of the "one" or parent documents.
- Advantages:
  - Provides better performance for read operations i.e. a request and retrieve related data in a single database operation.
  - Possible to update related data in a single atomic write operation.
- Disadvantage:
  - May lead to situations where documents grow uncontrollably.



E.g. Comment Embedded in BlogEntry

Object References -> ‘Normalized’ Data Model

- Normalized data models describe relationships using references between documents.



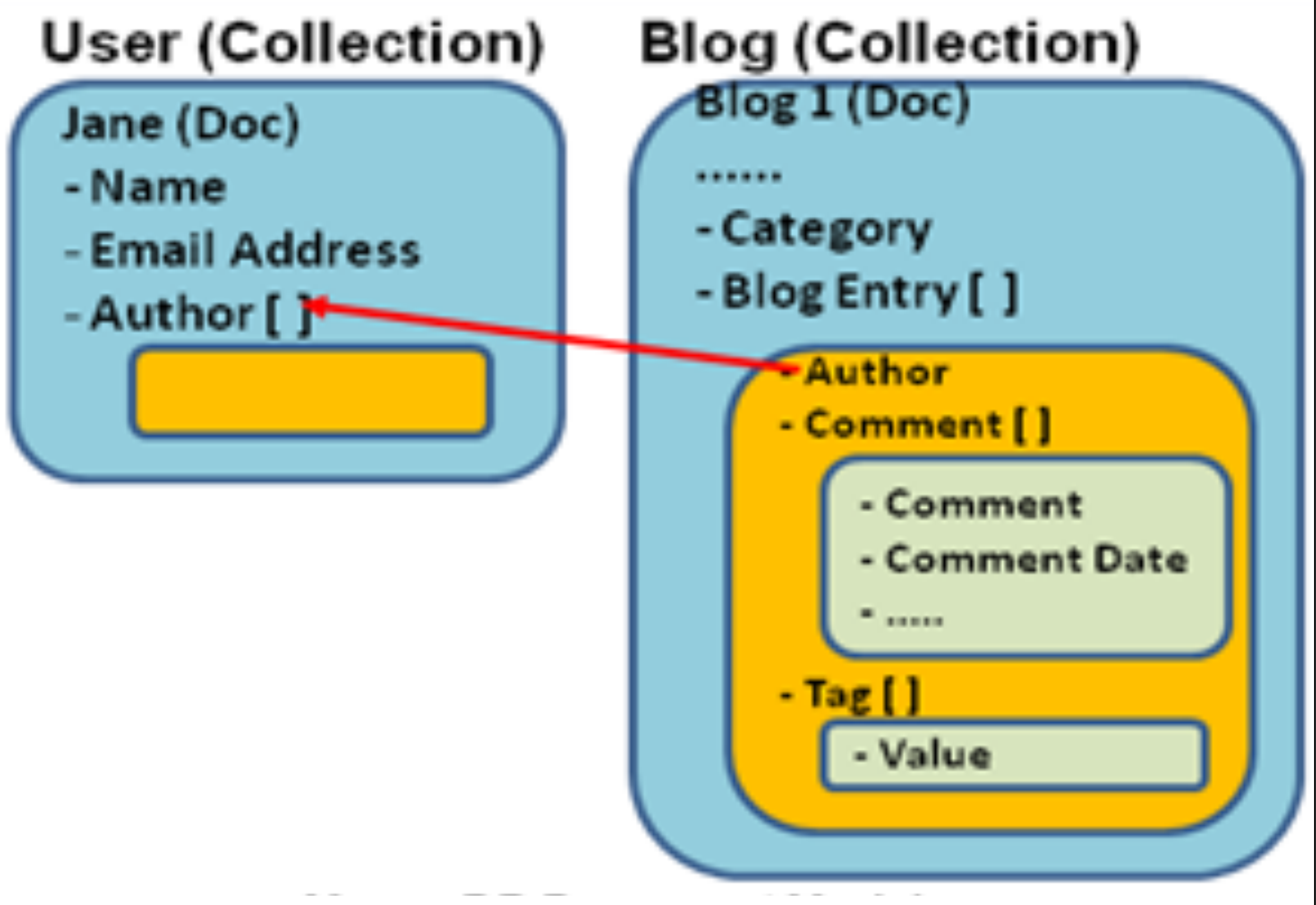
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

When to use Normalized Data Model?

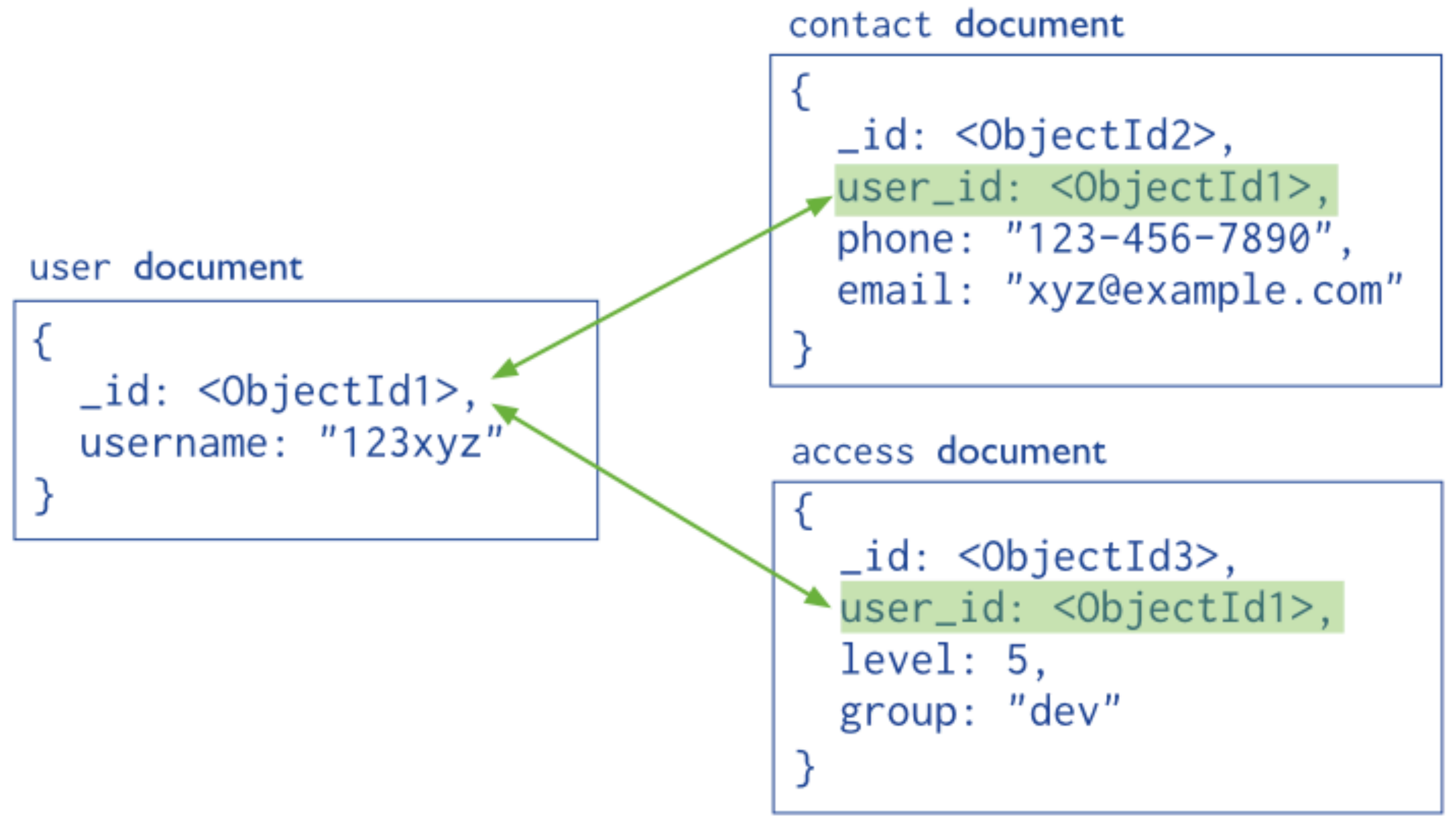
- When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication.
- To represent more complex many-to-many relationships.
- To model large hierarchical data sets



E.g. BlogEntry -> User

References can provide more flexibility than embedding. However, client-side applications must issue follow-up queries to resolve the references -> models may require more round trips to the server.

‘Normalized’

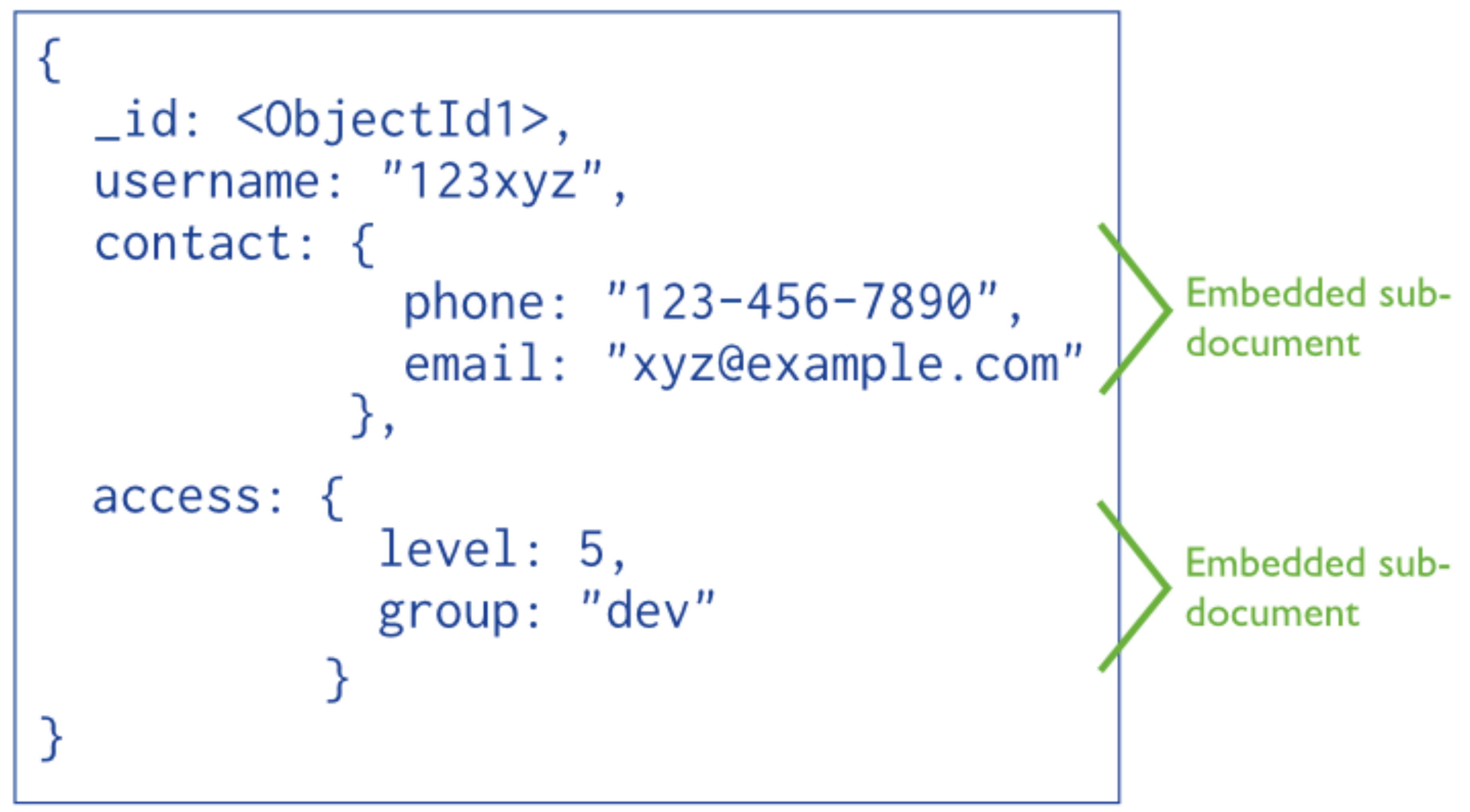


```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

‘Denormalized’

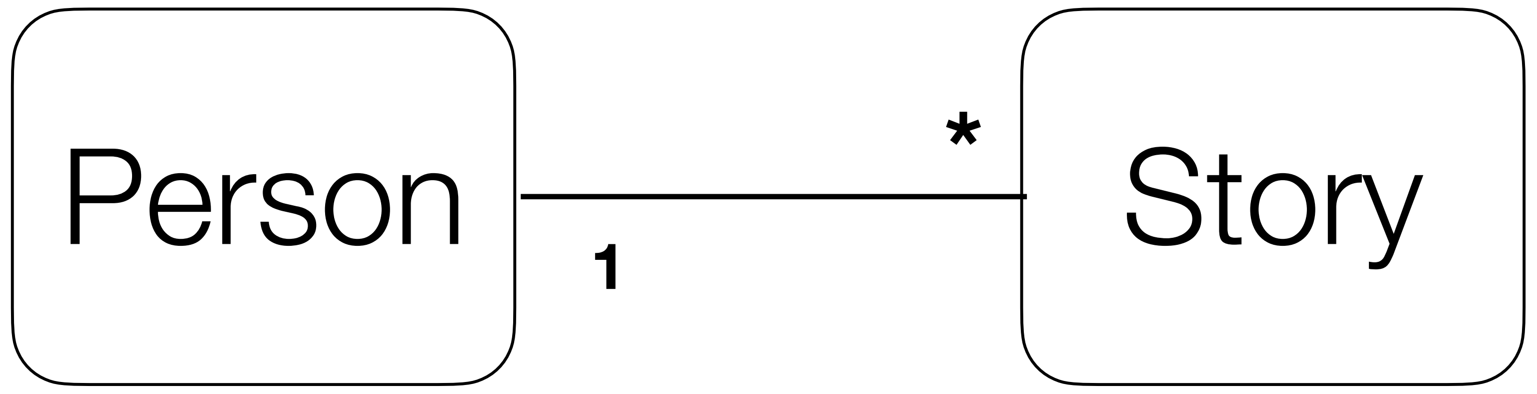


```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
          }
}
```

Model: One-to-Many

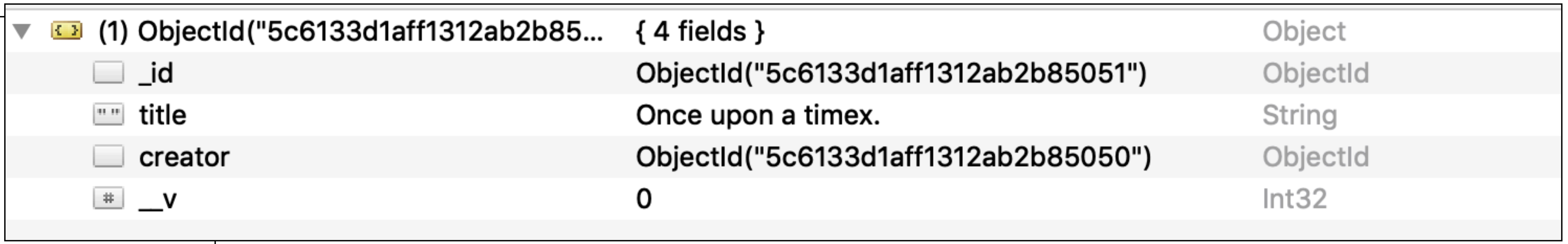
- Stories are written by Persons

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const personSchema = Schema({
  name: String,
  age: Number
});

const storySchema = Schema({
  creator: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Person'
  },
  title: String
});

const Story = mongoose.model('Story', storySchema);
const Person = mongoose.model('Person', personSchema);
```

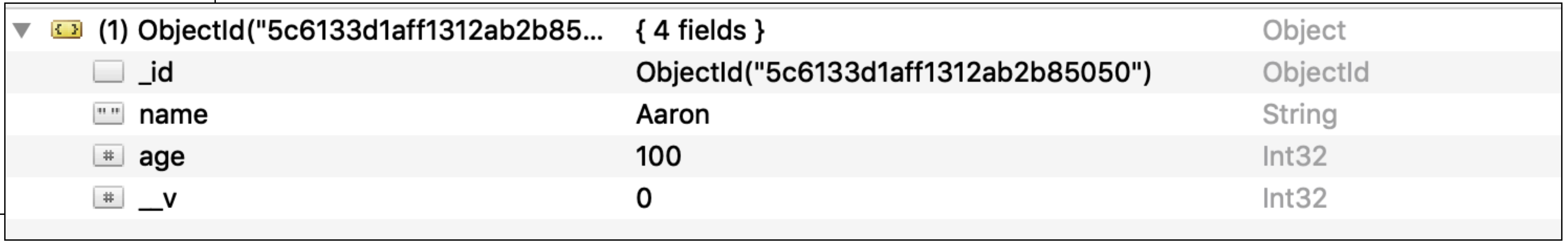


Creating the objects

```
var aaron = new Person({
  name: 'Aaron',
  age: 100
});

async function makeStory() {
  const newPerson = await aaron.save();
  const story1 = new Story({
    title: 'Once upon a timex.',
    creator: newPerson._id
  });
  await story1.save();
}

makeStory();
```

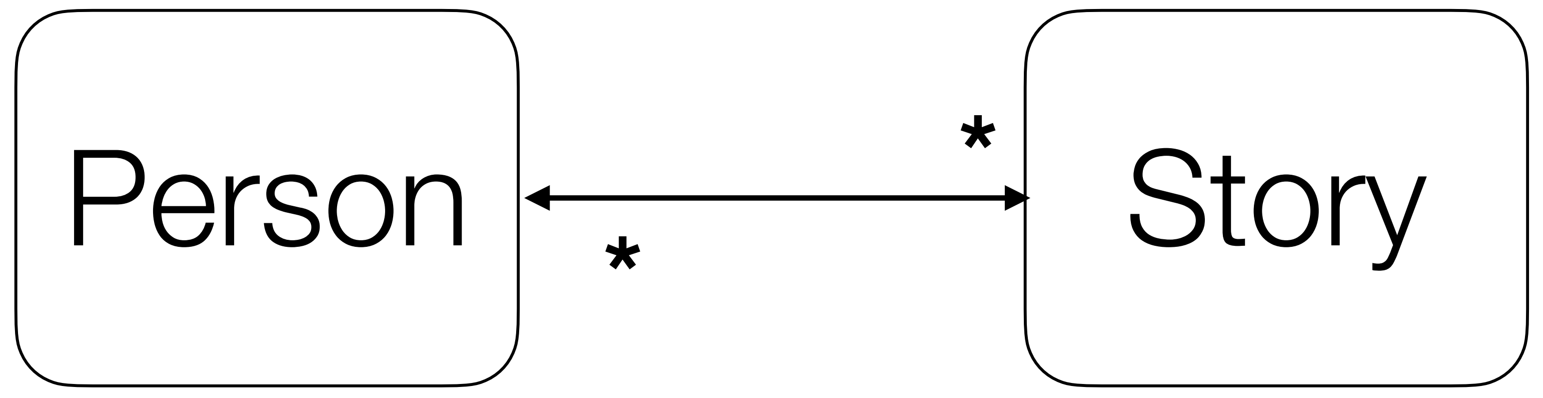


(1) ObjectId("5c6133d1aff1312ab2b85...	{ 4 fields }	Object
_id	ObjectId("5c6133d1aff1312ab2b85051")	ObjectId
title	Once upon a timex.	String
creator	ObjectId("5c6133d1aff1312ab2b85050")	ObjectId
__v	0	Int32

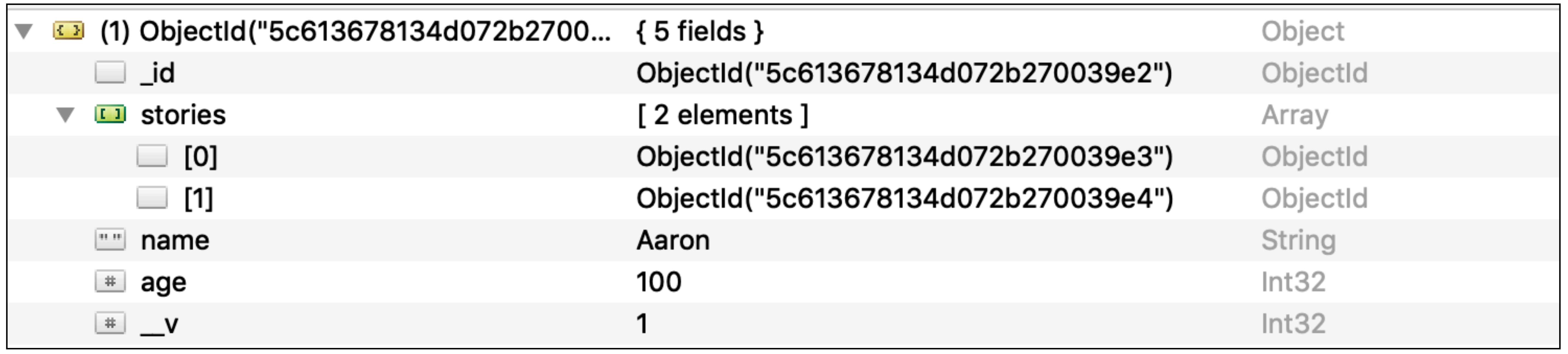


(1) ObjectId("5c6133d1aff1312ab2b85...	{ 4 fields }	Object
_id	ObjectId("5c6133d1aff1312ab2b85050")	ObjectId
name	Aaron	String
age	100	Int32
__v	0	Int32

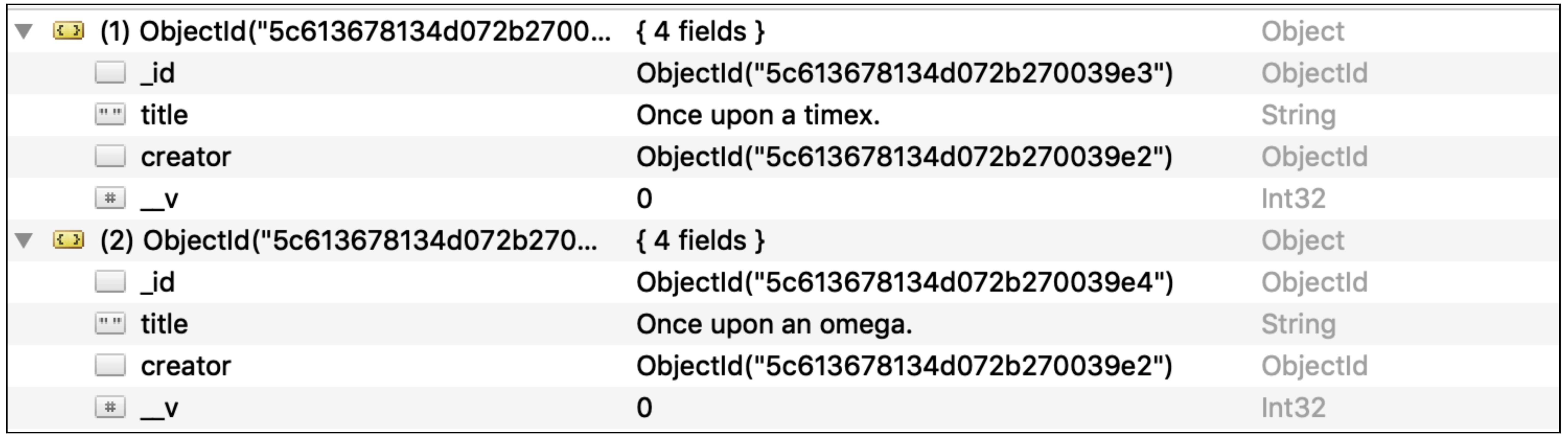
Model: One-to-Many Many-to-One



```
const personSchema = Schema({
  name: String,
  age: Number,
  stories: [
    {
      type: Schema.Types.ObjectId,
      ref: 'Story'
    }
  ]
});

const storySchema = Schema({
  creator: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Person'
  },
  title: String
});
```



```
▼ (1) ObjectId("5c613678134d072b2700...   { 5 fields }                          Object
      _id                                ObjectId("5c613678134d072b270039e2")  ObjectId
    ▼ stories                            [ 2 elements ]                        Array
        [0]                              ObjectId("5c613678134d072b270039e3")  ObjectId
        [1]                              ObjectId("5c613678134d072b270039e4")  ObjectId
      name                               Aaron                                 String
      age                                100                                   Int32
      __v                                1                                     Int32
```

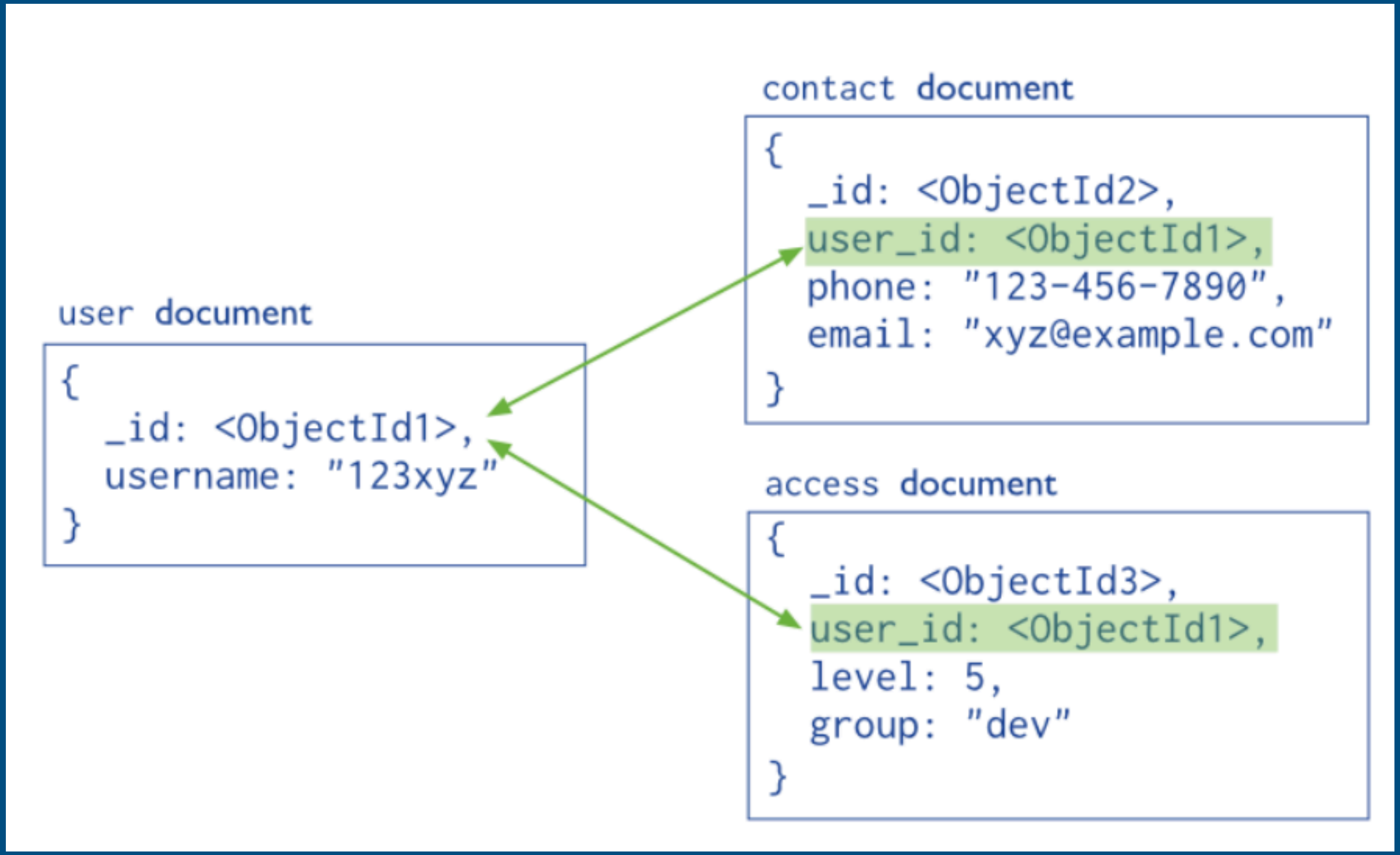


```
▼ (1) ObjectId("5c613678134d072b2700...   { 4 fields }                          Object
      _id                                ObjectId("5c613678134d072b270039e3")  ObjectId
      title                              Once upon a timex.                    String
      creator                            ObjectId("5c613678134d072b270039e2")  ObjectId
      __v                                0                                     Int32
▼ (2) ObjectId("5c613678134d072b270...    { 4 fields }                          Object
      _id                                ObjectId("5c613678134d072b270039e4")  ObjectId
      title                              Once upon an omega.                   String
      creator                            ObjectId("5c613678134d072b270039e2")  ObjectId
      __v                                0                                     Int32
```

Example Documents

```
 async function testStories() {

  var aaron = new Person({
      name: 'Aaron',
      age: 100
  });

  const newPerson = await aaron.save();

  const story1 = new Story({
    title: 'Once upon a timex.',
    creator: newPerson._id
  });

  const story2 = new Story({
    title: 'Once upon an omega.',
    creator: newPerson._id
  });

  await story1.save();
  await story2.save();

  newPerson.stories.push(story1._id);
  newPerson.stories.push(story2._id);

  await newPerson.save();
}
```

Mongo References



contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Full Stack Web Development