Referencing Schema in another Schema

Deciding how to write the value of ref while referencing a schema

models → post.js

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

const postSchema = new mongoose.Schema({
    content: {
        type: String,
        required: true
    },
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    }
}, {
    timestamps: true
});

const Post = mongoose.model('Post', postSchema);
module.exports = Post;
```

Here the schema for posts has the attribute 'user'.

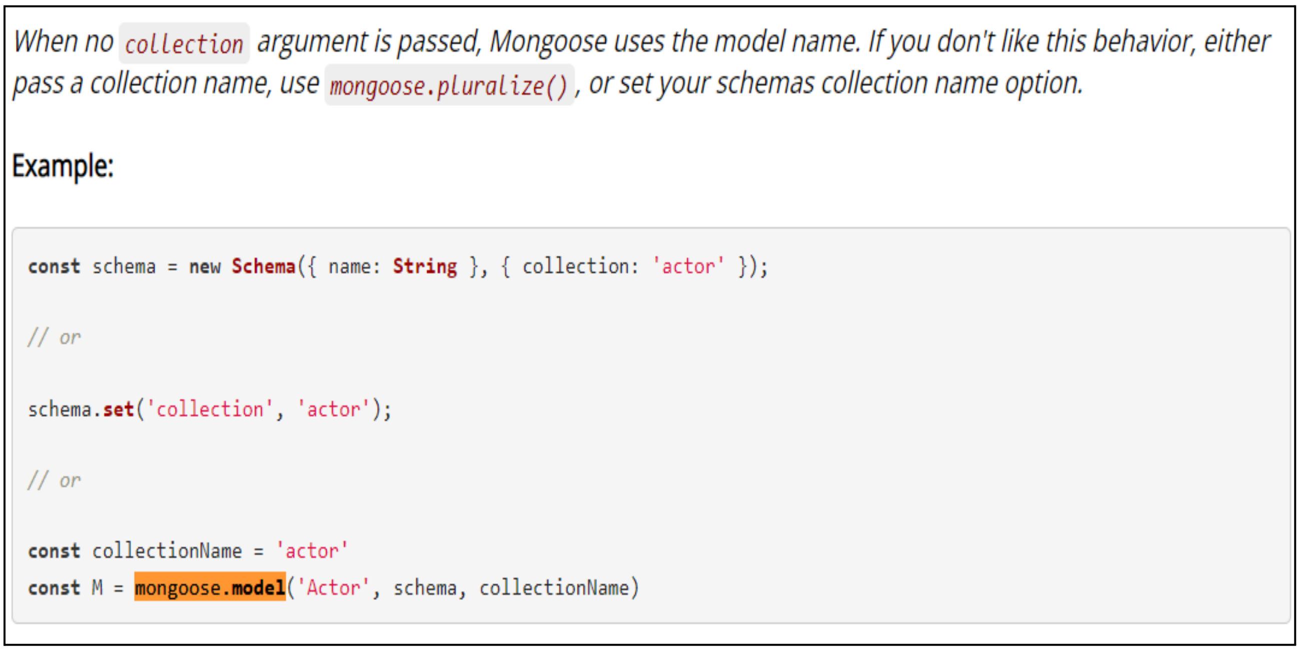
- The type of this attribute is the ObjectId of mongoose schema. It means the user attribute will have a value which will be an ObjectId.
- The ref of this attribute is 'User'. It means the value of the user attribute will be the ObjectId which will correspond to the id of an entry in the users database. But we don't write ref to be 'users' or 'user' or 'Users' as it has to be defined with respect to the schema name whose corresponding database we want to refer.

  This is how the User schema was exported. Here it was named 'User'.

  ```
  const User = mongoose.model('User', userSchema);
  module.exports = User;
  ```

Additional information for extra knowledge

Whenever we create a schema and corresponding entries for it, by default MongoDB creates a database and names it by suffixing ‘s’ at the end of the schema name (the first capital letter is changed to lower case). So the User schema names DB as ‘users ‘and Post schema as ‘posts’.
Refer to the below screenshot to know how to explicitly name the database.



When no `collection` argument is passed, Mongoose uses the model name. If you don't like this behavior, either pass a collection name, use `mongoose.pluralize()`, or set your schemas collection name option.

Example:

```
const schema = new Schema({ name: String }, { collection: 'actor' });

// or

schema.set('collection', 'actor');

// or

const collectionName = 'actor'
const M = mongoose.model('Actor', schema, collectionName)
```