Document	Purpose
Users	User information
Roles	Types of roles(eg: Student, Administrator, Teacher, Guest, etc)
Assessments	Name of assessments and which questions(question ids) they contains
Questions	Information about questions and which options(option ids) they have
Options	Options for the questions

			Relation	
Roles	-->	Users	: One to Many	[Many users can have the same role]
Assessments	-->	Questions	: Many to Many	[A single question CAN be repeated in many assessments]
Questions	-->	Options	: Many to Many	[A single option CAN be repeated in multiple questions]

Since a single user has only one role I could have nested user documents inside the Roles collection like so:

```
db.Roles.insert([{
        _id:'r1',
        name:'Student',
        url:'./welcome_student.html',
        users:  [
                        {_id:'user_1', name:'User One', role:'r1', password:'user_1123'},
                        {_id:'user_2', name:'User Two', role:'r1', password:'user_2123'},
                        {_id:'user_3', name:'User Three', role:'r1', password:'user_3123'}
                ]
},
{
        _id:'r0',
        name:'Administrator',
        url:'./welcome_admin.html',
        users:  [
                        {_id:'admin', name:'Default Administrator', role:'r0', password:'admin123'}
                ]
}])
```

but I wanted to keep the Users collection as a top-level collection because of it's importance/significance in the entire system
which is why I have a separate collection of Users.

I have used the "Manual References" method(from http://docs.mongodb.org/manual/reference/database-references/#document-references) for implementing the many to many ideology eg: in the Questions collection where the property 'options' holds all the _ids of the options which I want my question to have.
When constructing a complete question from the language-side this 'options' property will be used to further query the Options collection and extract the appropriate options.