MangoDB Class

#MangoDB Notes

Role-Based Access Control (RBAC)

- A user can be assigned one or more roles, and the scope of user access to the database system is determined by those assigned roles.
- Users have no access to the system outside the designated roles.

- Importantly, MongoDB access control is not enabled by default; you have to enable it through the security.authorization setting.

- A role grants permission to perform particular actions on a specific resource. A single user account can consist of multiple roles. Roles can be assigned:
  - At the time of user creation
  - When updating the roles of existing users

- There are two types of Roles in MongoDB:

- **Built-In Roles:-** MongoDB provides built-in roles to offer a set of privileges that are commonly needed in a database system.
- **User-Defined Roles:-** If built-in roles do not provide all the expected privileges, database administrators can define custom roles using the createRole method. Those roles are called User-Defined roles.

Built-in Roles

- The most common built-in roles of MongoDB.
- **Database user roles**
- Database user roles are normal user roles that are useful in regular database interactions.

Role	Description
read	Read all non-system collections and the system.js collection
readWrite	Both Read and Write functionality on non-system collections and the system.js collection

Database administration roles

- These are roles that are used to carry out administrative operations on databases.

Role	Description
dbAdmin	Perform administrative tasks such as indexing and gathering statistics, but cannot manage users or roles
userAdmin	Provides the ability to create and modify roles and users of a specific database
dbOwner	This is the owner of the database who can perform any action. It is equal to combining all the roles mentioned above: readWrite, dbAdmin, and userAdmin roles

Cluster admin roles

- These roles enable users to interact and administrate MongoDB clusters.

Role	**Description**
clusterManager	Enables management and monitoring functionality on the cluster. Provides access to config and local databases used in sharding and replication
clusterMonitor	Provide read-only access to MongoDB monitoring tools such as Cloud Manager or Ops Manager monitoring agent
hostManager	Provides the ability to monitor and manage individual servers
clusterAdmin	This role includes the highest number of cluster administrative privileges allowing a user to do virtually anything. This functionality is equal to the combination of clusterManager, clusterMonitor, hostManager roles, and dropDatabase action.

Backup & restoration roles

- These are the roles that are required for backup and restoring data in a MongoDB instance. They can only be assigned with the admin database.

Role	**Description**
backup	Provides the necessary privileges to backup data. This role is required for MongoDB Cloud Manager and Ops Manager, backup agents, and the monogdump utility.
restore	Provides the privileges to carry out restoration functions

All database roles

- These are database roles that provide privileges to interact with all databases, excluding local and config databases.

Role	**Description**
readAnyDatabase	Read any database
readWriteAnyDatabase	Provides read and write privileges to all databases
userAdminAnyDatabase	Create and Modify users and roles across all databases
dbAdminAnyDatabase	Perform database administrative functions on all databases

Superuser roles

- MongoDB can provide either direct or indirect system-wide superuser access. The following roles grant superuser privileges scoped to a specified database or databases.

- dbOwner
- userAdmin
- userAdminAnyDatabase

- The true superuser role is the root role, which provides systemwide privileges for all functions and resources.

User-Defined Roles

- MongoDB Role Management provides the necessary methods to create and manage user-defined roles.
- The most commonly used methods for user-defined role creation are shown in the following table.

Method	Description
db.createRole()	Create a role and its privileges
db.updateRole()	Update the user-defined role
db.dropRole()	Delete a user-defined role
db.grantPrivilegesToRole()	Assigns new privileges to a role
db.revokePrivilegesFromRole()	Removes privileges from a role

MongoDB role management

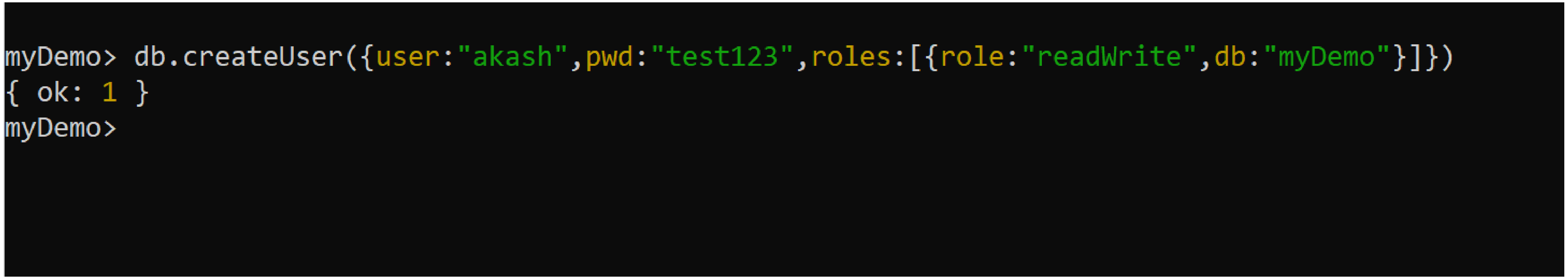
- Roles are defined using the following syntax:

```
roles: [
{
role: "<Role>", db: "<Database>"
}
]
```

- Assigning user roles at user creation
- First, create a user with read and write access to a specific database (myDemo) using the createUser method with roles parameter.

```
db.createUser(
{
user: "akash",
pwd: "test123",
roles: [
{
role: "readWrite",
db: "myDemo"
}
]
}
)
```

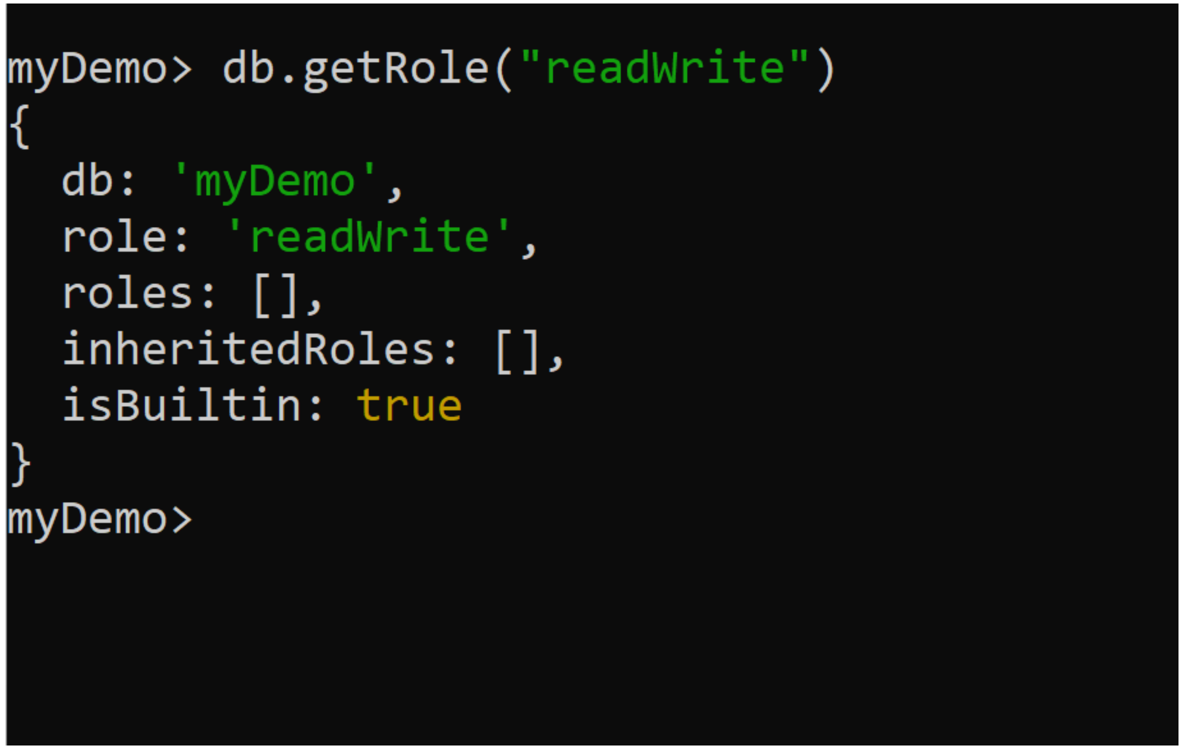
Example



```
myDemo> db.createUser({user:"akash",pwd:"test123",roles:[{role:"readWrite",db:"myDemo"}]})
{ ok: 1 }
myDemo>
```

Retrieving role information

- Using the getRole method, users can obtain information about a specific role.
- db.getRole("readWrite")

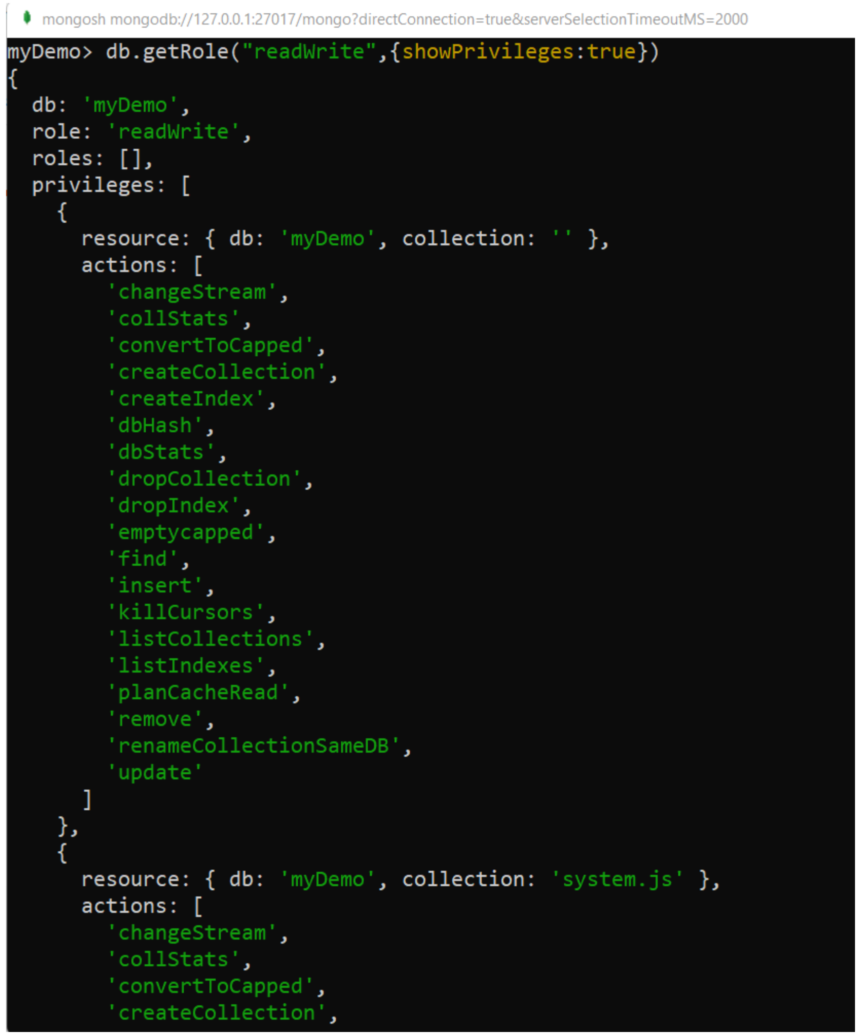


```
myDemo> db.getRole("readWrite")
{
  db: 'myDemo',
  role: 'readWrite',
  roles: [],
  inheritedRoles: [],
  isBuiltin: true
}
myDemo>
```

- To obtain the privileges associated with that role, use the showPrivileges option by setting its value as ‘true’. This can also be used in the getUser method.

- db.getRole("readWrite", { showPrivileges: true})

Example

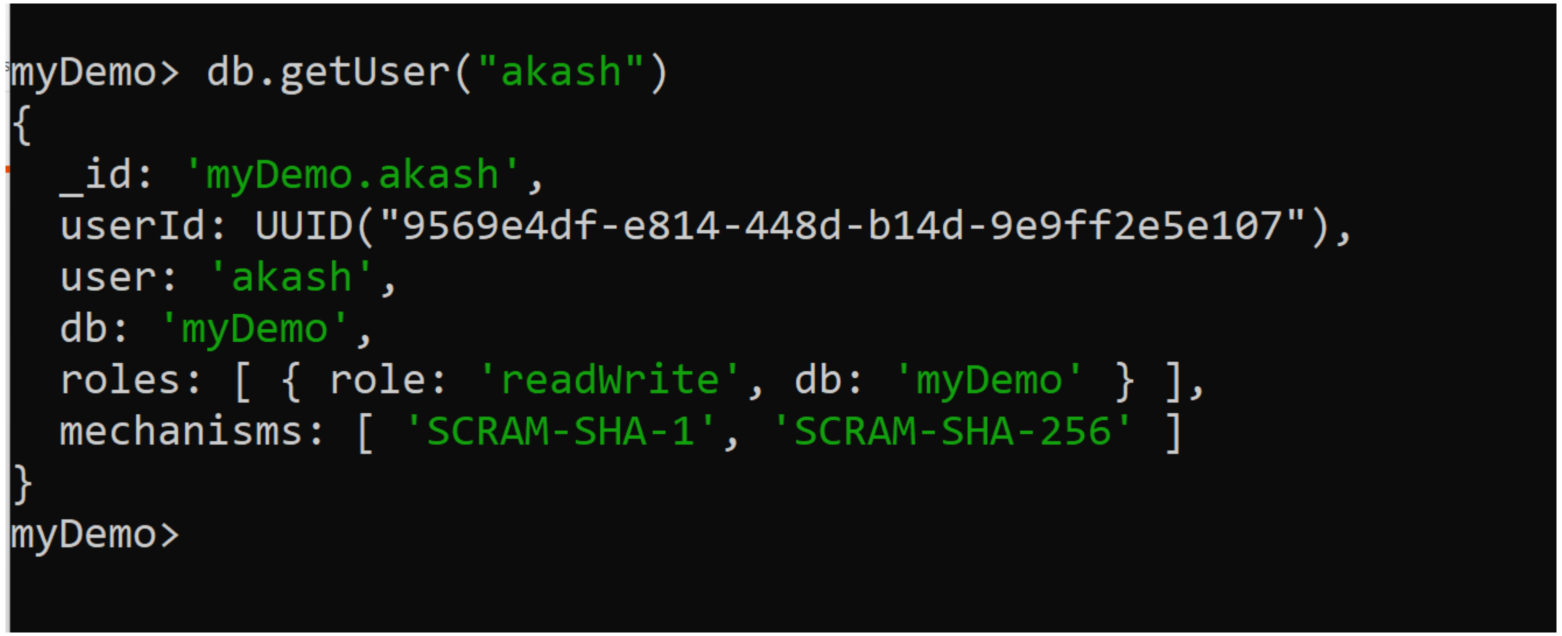


mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

```
myDemo> db.getRole("readWrite",{showPrivileges:true})
{
  db: 'myDemo',
  role: 'readWrite',
  roles: [],
  privileges: [
    {
      resource: { db: 'myDemo', collection: '' },
      actions: [
        'changeStream',
        'collStats',
        'convertToCapped',
        'createCollection',
        'createIndex',
        'dbHash',
        'dbStats',
        'dropCollection',
        'dropIndex',
        'emptycapped',
        'find',
        'insert',
        'killCursors',
        'listCollections',
        'listIndexes',
        'planCacheRead',
        'remove',
        'renameCollectionSameDB',
        'update'
      ]
    },
    {
      resource: { db: 'myDemo', collection: 'system.js' },
      actions: [
        'changeStream',
        'collStats',
        'convertToCapped',
        'createCollection',
```

Identifying assigned user roles

- The getUser method enables you to identify the roles assigned to a specific user by using the following syntax:
- db.getUser("<Username>")

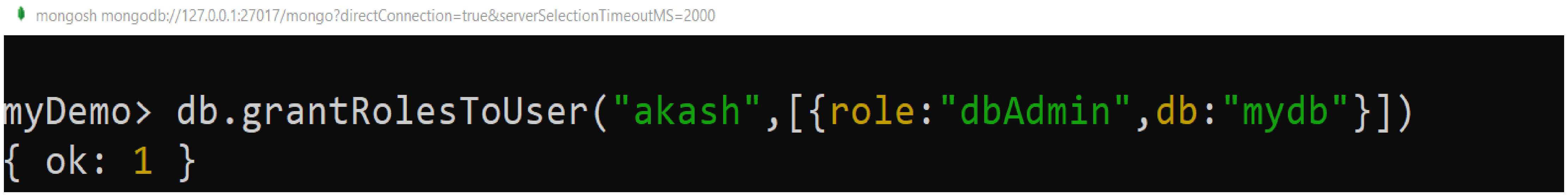


```
myDemo> db.getUser("akash")
{
  _id: 'myDemo.akash',
  userId: UUID("9569e4df-e814-448d-b14d-9e9ff2e5e107"),
  user: 'akash',
  db: 'myDemo',
  roles: [ { role: 'readWrite', db: 'myDemo' } ],
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
}
myDemo>
```

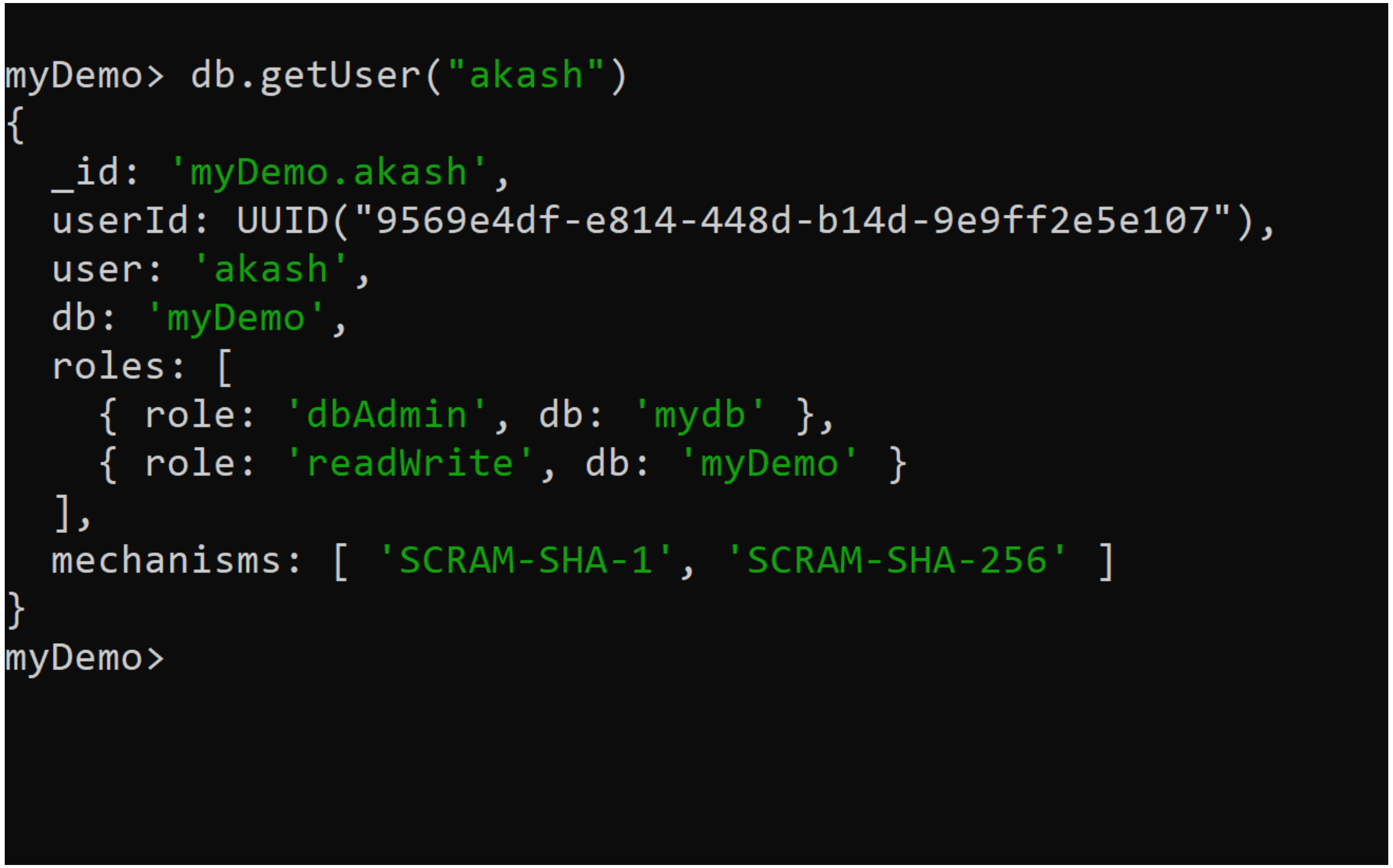
Granting & revoking user roles

- Using the grantRolesToUser and revokeRolesFromUser methods, you can modify the roles assigned to existing users. These methods use the following syntax:

```
db.<grantRolesToUser | revokeRolesFromUser> (
"<Username>",
[
{ role: "<Role>", db: "<Database>" }
]
)
```



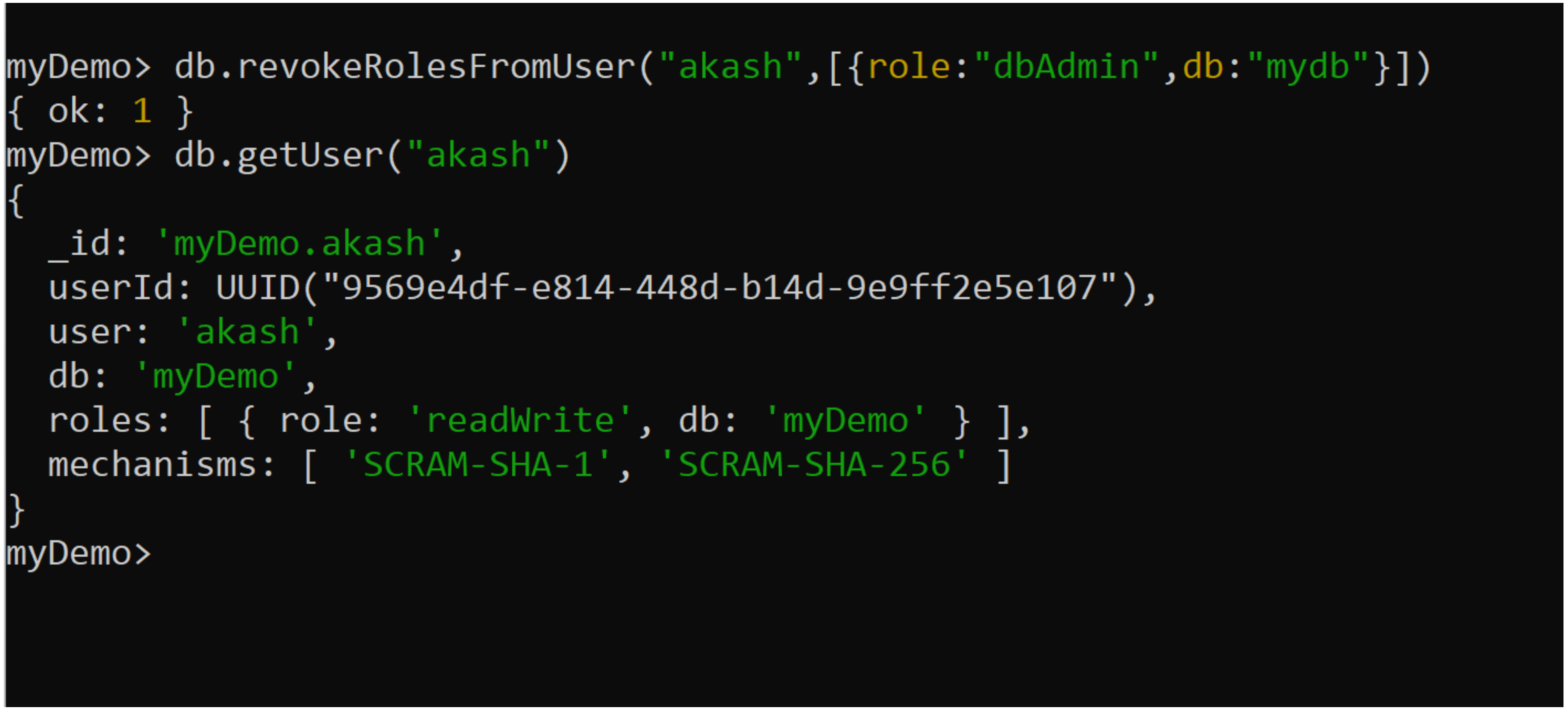
```
mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

myDemo> db.grantRolesToUser("akash",[{role:"dbAdmin",db:"mydb"}])
{ ok: 1 }
```



```
myDemo> db.getUser("akash")
{
  _id: 'myDemo.akash',
  userId: UUID("9569e4df-e814-448d-b14d-9e9ff2e5e107"),
  user: 'akash',
  db: 'myDemo',
  roles: [
    { role: 'dbAdmin', db: 'mydb' },
    { role: 'readWrite', db: 'myDemo' }
  ],
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
}
myDemo>
```

- Now you have added a new role to the user, “akash”. So, let’s remove that newly granted role using the revokeRolesFromUser method.

```
db.revokeRolesFromUser(
“akash",
[
{ role: "dbAdmin", db: “mydb" }
]
)
```



```
myDemo> db.revokeRolesFromUser("akash",[{role:"dbAdmin",db:"mydb"}])
{ ok: 1 }
myDemo> db.getUser("akash")
{
  _id: 'myDemo.akash',
  userId: UUID("9569e4df-e814-448d-b14d-9e9ff2e5e107"),
  user: 'akash',
  db: 'myDemo',
  roles: [ { role: 'readWrite', db: 'myDemo' } ],
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
}
myDemo>
```

Creating user-defined roles

- Using the createRole method, you can create a new role according to your needs.

Syntax

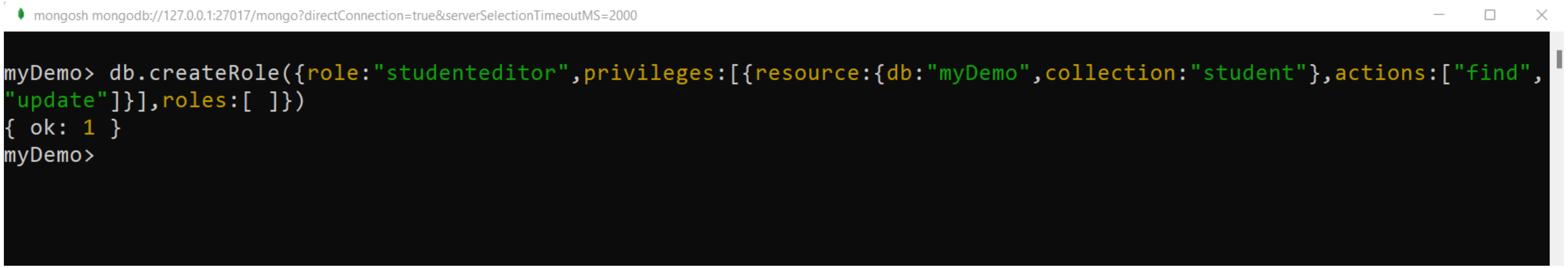
```
db.createRole(
{
role: "<RoleName>",
privileges: [
{
resource: { db: "<Database>", collection: "<Collection>"},
actions: [ "<Actions>" ]
}
],
roles: [
{ role: "<Role>", db: "<Database>" }
]
}
)
```

- To associate a user-defined role to all databases or collections, you can specify the resources with empty double quotes, as shown below.

- resource: { db: "", collection: ""}

- Here,you will create a role that limits a user’s access to a specific database collection (student collection of myDemo database).
- It limits the user actions to find and update commands without inheriting any other privileges.

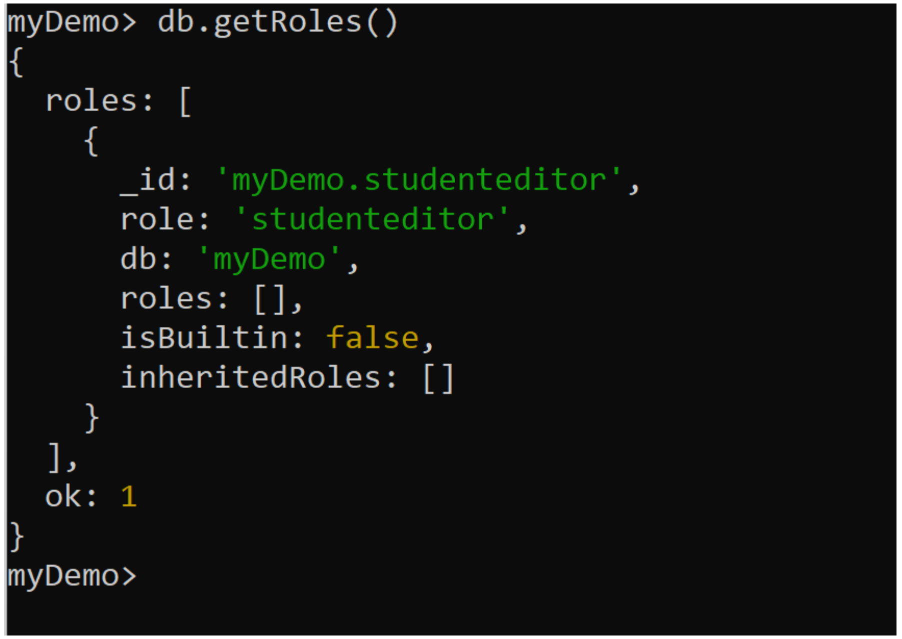
```
db.createRole(
{
role: "studenteditor",
privileges: [
{
resource: { db: "myDemo", collection: "student"},
actions: [ "find", "update" ]
}
],
roles: [ ]
}
)
```



mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

```
myDemo> db.createRole({role:"studenteditor",privileges:[{resource:{db:"myDemo",collection:"student"},actions:["find",
"update"]}],roles:[ ]})
{ ok: 1 }
myDemo>
```

- We can identify the user-defined roles using the isBuiltin parameter.
- The studenteditor role has false for the that parameter, indicating it as a non-built-in role.

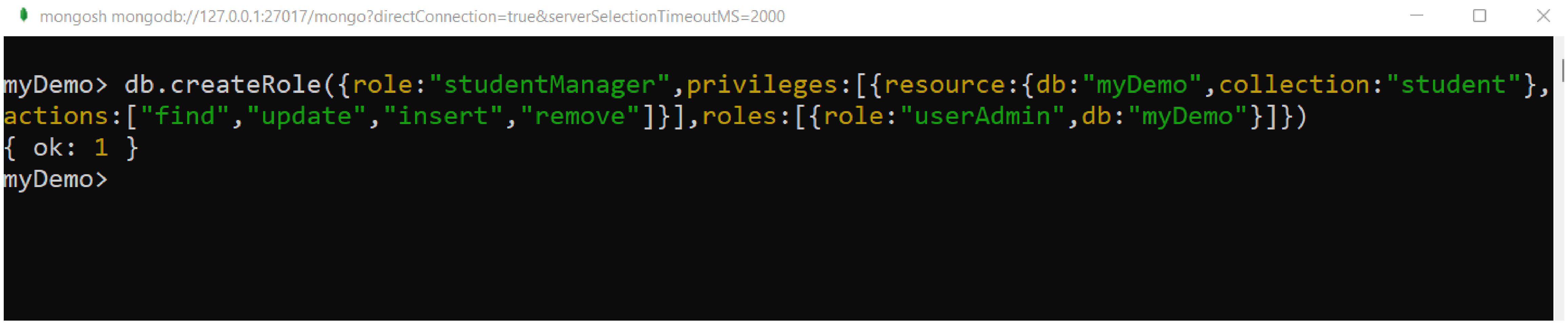


```
myDemo> db.getRoles()
{
  roles: [
    {
      _id: 'myDemo.studenteditor',
      role: 'studenteditor',
      db: 'myDemo',
      roles: [],
      isBuiltin: false,
      inheritedRoles: []
    }
  ],
  ok: 1
}
myDemo>
```

Example 2

- In this example we will create an studentmanager role with all the CURD privileges and inherit the privileges from the userAdmin role.

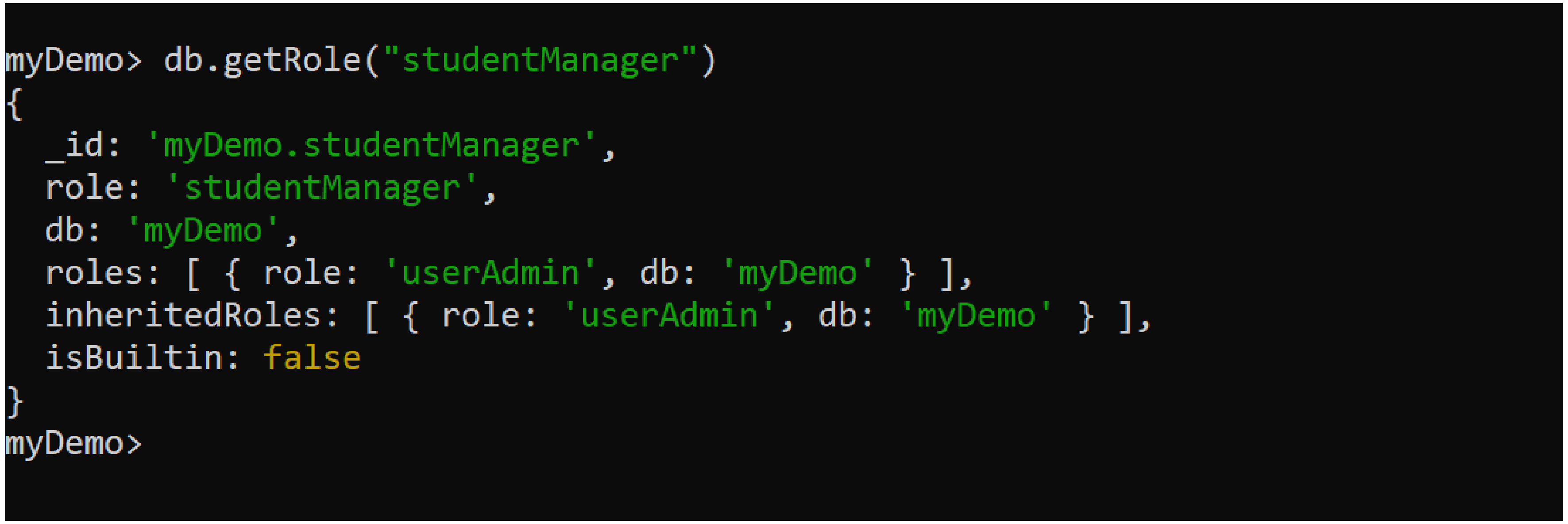
```
db.createRole(
{
role: "studentmanager",
privileges: [
{
resource: { db: "myDemo", collection: "student"},
actions: [ "find", "update", "insert", "remove" ]
}
],
roles: [
{ role: "userAdmin", db: "myDemo" }
]
}
)
```



mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

```
myDemo> db.createRole({role:"studentManager",privileges:[{resource:{db:"myDemo",collection:"student"},
actions:["find","update","insert","remove"]}],roles:[{role:"userAdmin",db:"myDemo"}]})
{ ok: 1 }
myDemo>
```

- If you check the studentManager role using the getRole method, it will display the details of the role, including the inherited permissions.



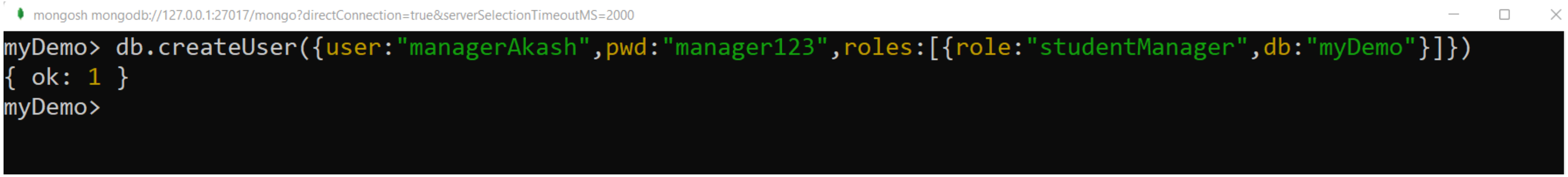
```
myDemo> db.getRole("studentManager")
{
  _id: 'myDemo.studentManager',
  role: 'studentManager',
  db: 'myDemo',
  roles: [ { role: 'userAdmin', db: 'myDemo' } ],
  inheritedRoles: [ { role: 'userAdmin', db: 'myDemo' } ],
  isBuiltin: false
}
myDemo>
```

Assigning user-defined roles to users

- You can assign user-defined roles to a new user or update the roles of an existing user in the same way you do it with a built-in role.

Creating a new user:

```
db.createUser(
{
user: "managerAkash",
pwd: "manager123",
roles: [
{
role: “studentManager",
db: "myDemo"
}
]
}
)
```

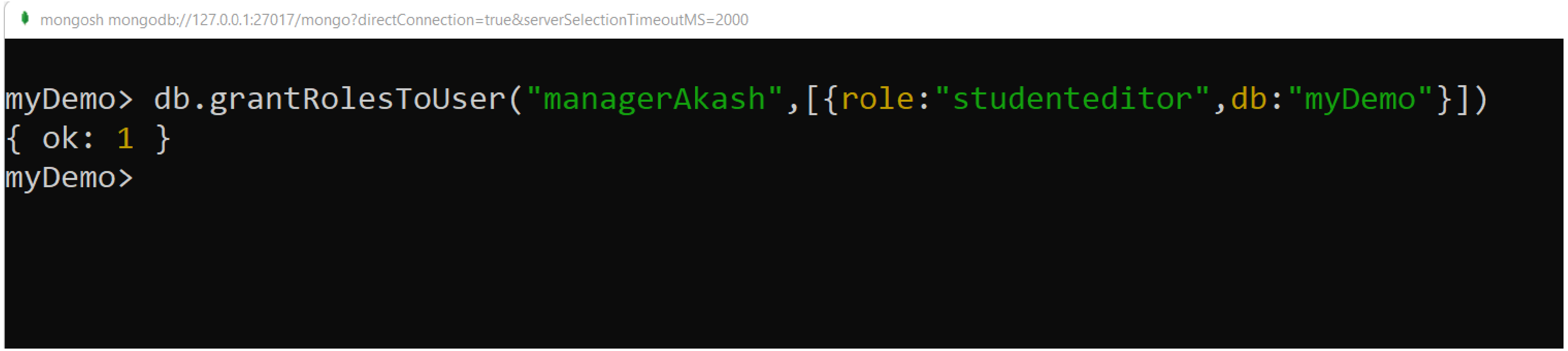


mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

```
myDemo> db.createUser({user:"managerAkash",pwd:"manager123",roles:[{role:"studentManager",db:"myDemo"}]})
{ ok: 1 }
myDemo>
```

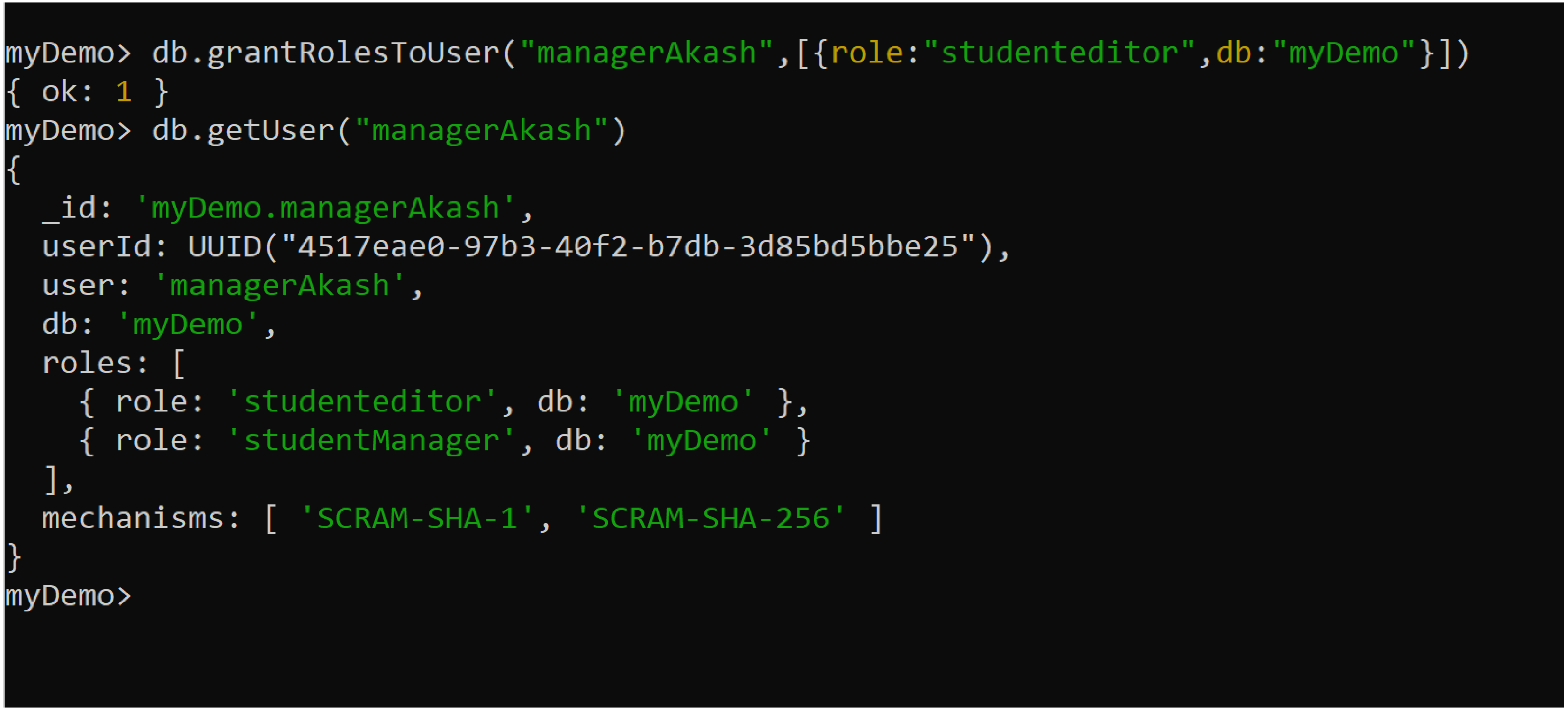
Granting new roles:

```
db.grantRolesToUser(
"mangerAkash",
[
{ role: "studenteditor", db: "myDemo" }
]
)
```


mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

```
myDemo> db.grantRolesToUser("managerAkash",[{role:"studenteditor",db:"myDemo"}])
{ ok: 1 }
myDemo>
```

Show Roles



```
myDemo> db.grantRolesToUser("managerAkash",[{role:"studenteditor",db:"myDemo"}])
{ ok: 1 }
myDemo> db.getUser("managerAkash")
{
  _id: 'myDemo.managerAkash',
  userId: UUID("4517eae0-97b3-40f2-b7db-3d85bd5bbe25"),
  user: 'managerAkash',
  db: 'myDemo',
  roles: [
    { role: 'studenteditor', db: 'myDemo' },
    { role: 'studentManager', db: 'myDemo' }
  ],
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
}
myDemo>
```

Updating & deleting user-defined roles

- You can update the user-defined roles using the updateRole method and delete the roles using the dropRole method.

- Update Role

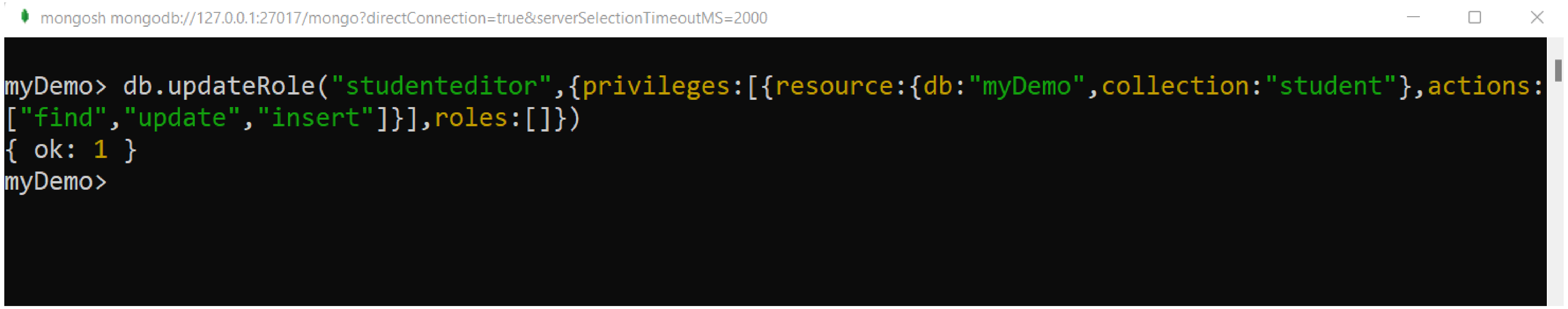
```
db.updateRole(
"studenteditor",
{
privileges: [
{
resource: { db: "myDemo", collection: "student"},
actions: [ "find", "update", "insert" ]
}
],
roles: [ ]
}
)
```

- Show Role

```
db.getRole("studenteditor",{showPrivileges:true})
```

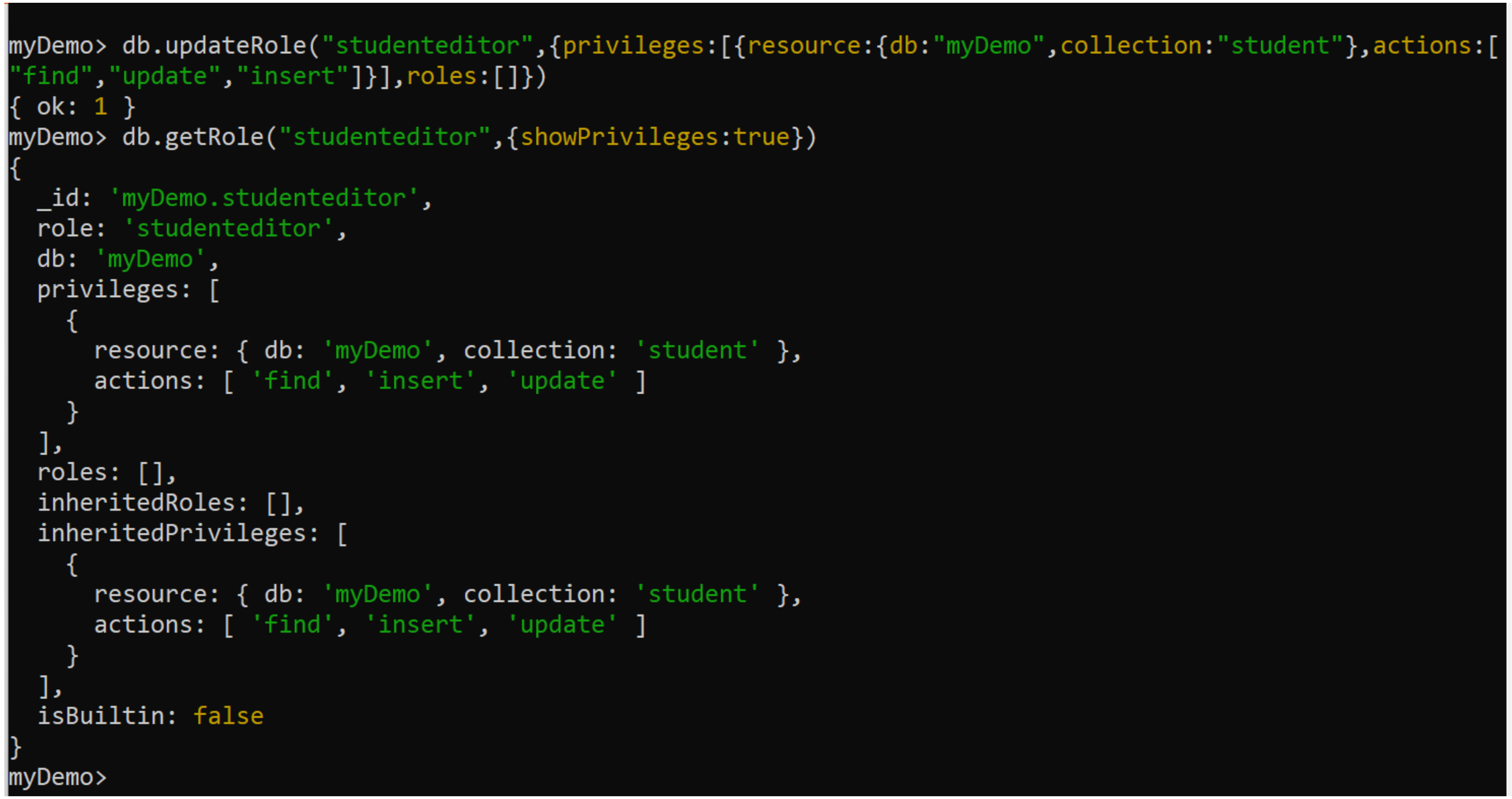
Example

```
db.updateRole(
"studenteditor",
{
privileges: [
{
resource: { db: "myDemo", collection: "student"},
actions: [ "find", "update", "insert" ]
}
],
roles: [ ]
}
)
```



```
mongosh mongodb://127.0.0.1:27017/mongo?directConnection=true&serverSelectionTimeoutMS=2000

myDemo> db.updateRole("studenteditor",{privileges:[{resource:{db:"myDemo",collection:"student"},actions:
["find","update","insert"]}],roles:[]})
{ ok: 1 }
myDemo>
```

Show Roles



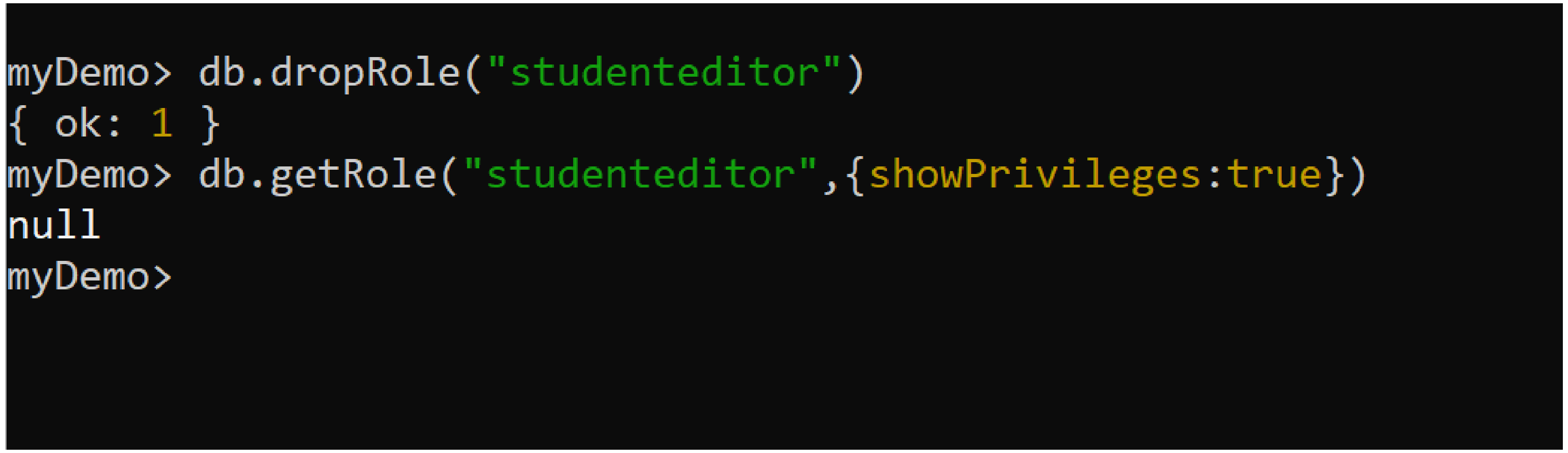
```
myDemo> db.updateRole("studenteditor",{privileges:[{resource:{db:"myDemo",collection:"student"},actions:[
"find","update","insert"]}],roles:[]})
{ ok: 1 }
myDemo> db.getRole("studenteditor",{showPrivileges:true})
{
  _id: 'myDemo.studenteditor',
  role: 'studenteditor',
  db: 'myDemo',
  privileges: [
    {
      resource: { db: 'myDemo', collection: 'student' },
      actions: [ 'find', 'insert', 'update' ]
    }
  ],
  roles: [],
  inheritedRoles: [],
  inheritedPrivileges: [
    {
      resource: { db: 'myDemo', collection: 'student' },
      actions: [ 'find', 'insert', 'update' ]
    }
  ],
  isBuiltin: false
}
myDemo>
```

- The most important thing to keep in mind when updating roles is that it will completely replace old values in the privileges and roles arrays.
- Therefore, you need to provide the complete arrays with the modifications when updating a user-defined role.

Drop Role

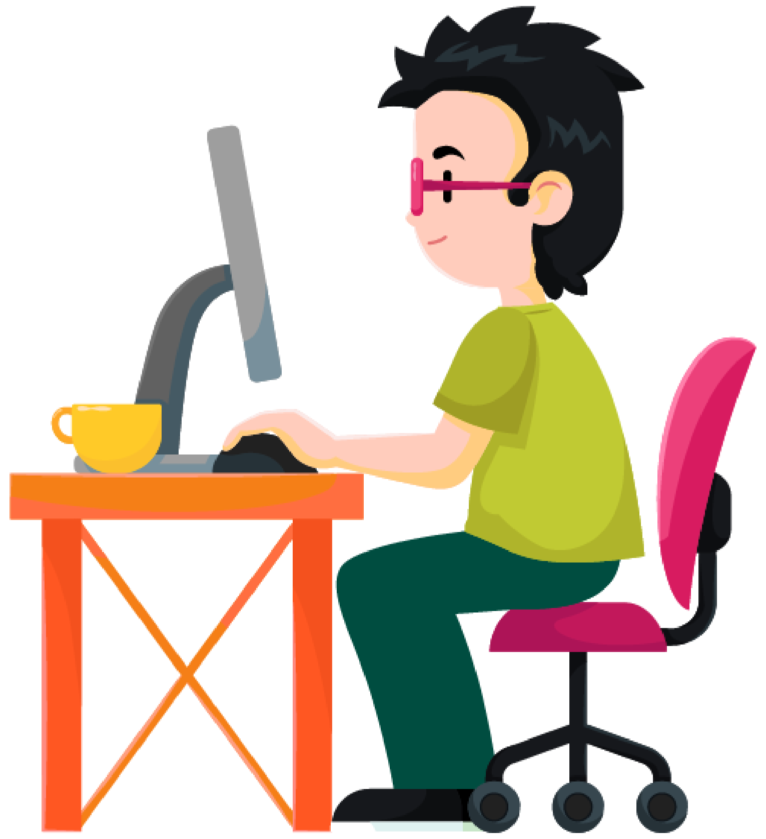
- The dropRole method has a single functionality to remove a user-defined role. You can remove the inventoryeditor role from the database, as shown below.

- db.dropRole("studenteditor")



```
myDemo> db.dropRole("studenteditor")
{ ok: 1 }
myDemo> db.getRole("studenteditor",{showPrivileges:true})
null
myDemo>
```

Get Exclusive Video Tutorials



www.aptutorials.com

https://www.youtube.com/user/Akashtips

Get More Details

www.akashsir.com

If You Liked It !

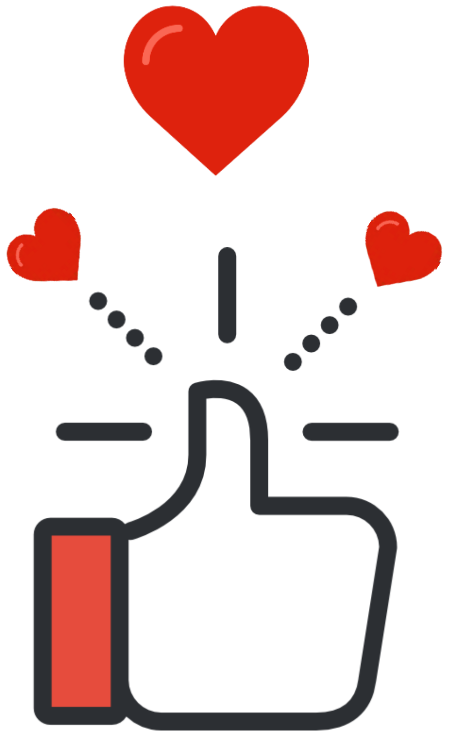
Rating Us Now

Just Dial

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET

Sulekha

https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad



Connect With Me

Akash Padhiyar
#AkashSir

www.akashsir.com
www.akashtechnolabs.com
www.akashpadhiyar.com
www.aptutorials.com

# Social Info

Facebook	Akash.padhiyar
Instagram	Akashpadhiyar
Twitter	Akash_padhiyar
WhatsApp	+91 99786-21654

#Akashpadhiyar
#aptutorials