SESSION 6
MONGODB SHELL METHODS

Learning Objectives

In this session, students will learn to:

- List different MongoDB Shell methods
- Describe collection methods in MongoDB Shell
- Explain various database methods in MongoDB Shell
- Explain role management methods in MongoDB Shell
- Describe various user management methods in MongoDB Shell

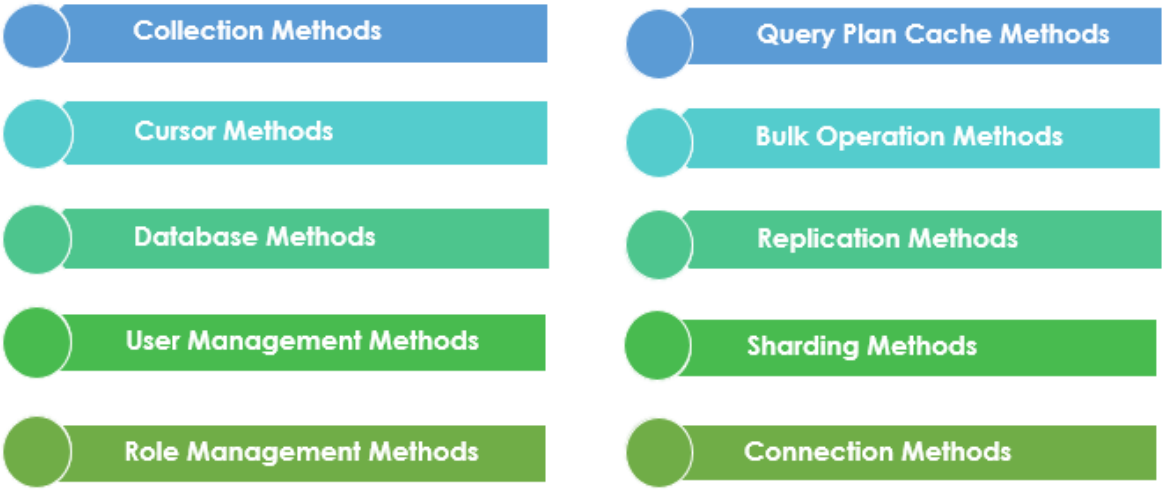
As discussed in an earlier session, MongoDB Shell is the command-line interface that provides quick and easy access to the MongoDB database. MongoDB Shell provides various collection methods to create, delete, rename, and manage collections in a database. It also provides database methods to create, run, drop, and fetch data from collections. There are also role management methods and user management methods to create, update, and drop roles and users in a database. This session will provide an overview of different MongoDB Shell methods to manage collections, users, and roles in a MongoDB database.

6.1 Introduction to MongoDB Shell Methods

MongoDB provides a collection of methods that facilitate the users to:

- Create, insert, update, and delete documents, collections, and databases
- Change the way a query is executed
- Manage users and authentication
- Provide built-in roles and authorization based on roles
- Perform bulk operations
- Replicate data in multiple MongoDB servers
- Distribute data of very large datasets across multiple machines

Some of the important methods provided by MongoDB are:



6.2 Collection Methods

MongoDB offers a range of collection methods to perform the Create, Read, Update, and Delete (CRUD) operations which has been covered in earlier sessions.

- **Count**

  MongoDB provides the `countDocuments` method to count the number of documents in a collection. The syntax for the `countDocuments` method is:

```
db.collection.countDocuments(query, options)
```

Table 6.1 describes the parameters of the `countDocuments` method.

Parameter	Fields	Type	Description
query	-	document	Specifies the criteria based on which the documents must be counted; if empty, all the documents are counted
options		document	Provides optional fields
	limit	integer	Specifies the maximum number of documents that must be counted
	skip	integer	Specifies the number of documents to skip before beginning the count
	hint	string or document	Specifies an index name to be used in the query
	maxTimeMS	integer	Specifies the maximum time the query can run

Table 6.1: Parameters of the `countDocuments` Method

In the `sample_analytics` database, consider that the user wants to know the total number of documents in the `accounts` collection. To count the number of documents, the user can run the query as:

```
db.accounts.countDocuments({})
```

The output of this query is shown in Figure 6.1.

```
sample_analytics> db.accounts.countDocuments()
1746
```

Figure 6.1: Output of the `countDocuments` Method

Consider that the user wants the total number of documents in the `accounts` collection that has the value of the `limit` field set greater than `10000`. To get the required count, the user can execute the query as:

```
db.accounts.countDocuments( { limit: { $gte: 10000}})
```

The output of the query is shown in Figure 6.2.

```
sample_analytics> db.accounts.countDocuments( { limit: { $gte: 10000}})
1701
```

Figure 6.2: Output of the `countDocuments` Method with Condition

- **Distinct**

  MongoDB provides the `distinct` method to fetch distinct values for a field in a collection. The syntax for the `distinct` method is:

```
db.collection.distinct(field, query, options)
```

Table 6.2 describes the parameters of the `distinct` method.

Parameter	Fields	Type	Description
`field`	-	`string`	Specifies the field for which distinct values must be fetched
`query`	-	`document`	Specifies the criteria based on which distinct values must be fetched
`options`	`collation`	`document`	Specifies language-specific rules such as lettercase

Table 6.2: Parameters for the `distinct` method

Consider that the user wants to fetch the distinct values for the `storeLocation` field in the `products` document of the `accounts` collection. To get the desired output, the user can execute the query as:

```
db.accounts.distinct( "products" )
```

The output of the query is shown in Figure 6.3.

```
sample_analytics> db.accounts.distinct( "products" )
[
  'Brokerage',
  'Commodity',
  'CurrencyService',
  'Derivatives',
  'InvestmentFund',
  'InvestmentStock'
]
```

Figure 6.3: Output of the `distinct` Method

- **Find and Update**

  MongoDB provides the `findOneAndUpdate` method to locate a document and update its fields based on the criteria specified in the method. The syntax for the `findOneAndUpdate` method is:

```
db.collection.findOneAndUpdate(filter, update, options)
```

Table 6.3 describes the parameters of the `findOneAndUpdate` method.

Parameter	Fields	Type	Description
filter	-	document	Specifies the criteria based on which the document must be updated; if empty, then the first document in the collection is updated
update	-	document	Specifies the updates that must be done
options		document	Specifies optional fields
	projection	document	Specifies the fields that must be returned; if omitted, then all the fields are returned
	sort	document	Specifies the order in which the documents matched by the filter parameter must be sorted
	maxTimeMS	number	Specifies the maximum

Parameter	Fields	Type	Description
			time in milliseconds within which the query must run; if exceeded, throws an error
	`upsert`	`boolean`	Converts an update to insert a new document if set to `true` and no matching document exists for the criteria specified in `filter`
	`returnDocument`	`string`	Specifies whether the original or the updated document will be returned • If set to `before`, the original document is returned • If set to `after`, the updated document is returned • Takes precedence over `returnNewDocument` parameter
	`returnNewDocument`	`boolean`	Fetches the updated document if set to `true`
	`collation`	`document`	Specifies language-specific rules such as lettercase
	`arrayFilters`	`array`	Specifies the array criteria based on which elements in an array must be modified

Table 6.3: Parameters for the `findOneAndUpdate` Method

Consider that the user wants to find the document in the `accounts` collection, where `account_id` is `674364` and increase the value of its `limit` by `500`. To view the existing `limit` in this document before performing the update operation, the user can execute the query as:

```
db.accounts.find({ "account_id" : 674364 })
```

The output of the query is shown in Figure 6.4.

```
sample_analytics> db.accounts.find({"account_id":674364})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238f"),
    account_id: 674364,
    limit: 10000,
    products: [ 'InvestmentStock' ]
  }
]
```

Figure 6.4: Output of the `find` Method with Specific `account_id`

The value in the `limit` field is `10000`. To increase the value in the `limit` field by `500` for this document, execute the query as:

```
db.accounts.findOneAndUpdate({ "account_id" : 674364 },
{ $inc : { "limit" : 500 } }, {returnNewDocument:true})
```

The output of the query is shown in Figure 6.5.

```
sample_analytics> db.accounts.findOneAndUpdate({ "account_id" : 674364 },
 { $inc : { "limit" : 500 } }, {returnNewDocument:true})
{
  _id: ObjectId("5ca4bbc7a2dd94ee5816238f"),
  account_id: 674364,
  limit: 10500,
  products: [ 'InvestmentStock' ]
}
```

Figure 6.5: Output of the `findOneAndUpdate` Method

The value in the `limit` field is updated as `10500` in the returned document. Here, `$inc` is the update operator and the `returnNewDocument` option when set to `true` ensures that the updated document is returned.

- **Find and Replace**

  MongoDB provides the `findOneAndReplace` method to locate a document and replace it based on the conditions specified in the method. The syntax for the `findOneAndReplace` method is:

```
db.collection.findOneAndReplace(filter, replacement,
options)
```

Table 6.4 describes the parameters of the `findOneAndReplace` method.

Parameter	Fields	Type	Description
`filter`	-	`document`	Specifies the criteria based on which the document must be replaced; if empty, then the first document in the collection is replaced
`replacement`		`document`	Specifies the replacement document; _id value in the replacement document must be same as that of the replaced document
`options`		`document`	Specifies optional fields
	`projection`	`document`	Specifies the fields that must be returned; if omitted, then all the fields are returned
	`sort`	`document`	Specifies the order in which the documents matched by the filter parameter must be sorted
	`maxTimeMS`	`number`	Specifies the maximum time in milliseconds within which the query must run; if exceeded, throws an error
	`upsert`	`boolean`	Inserts a document specified by the replacement parameter if set to `true` and there is no document matching the criteria in the filter parameter
	`returnDocument`	`string`	Specifies whether the original or the updated document will be returned • If set to `before`, the original document is returned

Parameter	Fields	Type	Description
			• If set to `after`, the updated document is returned • Takes precedence over `returnNewDocument` parameter
	`returnNewDocument`	`boolean`	Fetches the replaced document if set to `true`
	`collation`	`document`	Specifies language-specific rules such as lettercase

Table 6.4: Parameters of the `findOneAndReplace` Method

Consider that the user wants to find all documents in the `accounts` collection, where the value of the `limit` field is greater than `5000`. In the first document returned, the user wants to replace the value of the products field as `Commodity` and the `limit` field as `2500`. To view all the documents where the value of the `limit` field is greater than `5000` before performing the replacement operation, the user can execute the query as:

```
db.accounts.find({"limit":{$gte:5000}})
```

The output of the query is shown in Figure 6.6.

```
sample_analytics> db.accounts.find({"limit":{$gte:5000}})
[
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238f"),
    account_id: 674364,
    limit: 10500,
    products: [ 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238e"),
    account_id: 198100,
    limit: 11500,
    products: [ 'Derivatives', 'CurrencyService', 'InvestmentStock' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee5816238d"),
    account_id: 557378,
    limit: 10000,
    products: [ 'InvestmentStock', 'Commodity', 'Brokerage', 'CurrencyService' ]
  },
  {
    _id: ObjectId("5ca4bbc7a2dd94ee58162390"),
    account_id: 278603,
    limit: 10000,
    products: [ 'Commodity', 'InvestmentStock' ]
  },
```

Figure 6.6: Output of the `find` Method with Specific `Limit` Value

In the first document that is returned, the value of the products field is `InvestmentStock`, and the `limit` field is `10000`. To replace the value of the `products` field as `Commodity` and the `limit` field as `2500`, the user can execute the query as:

```
db.accounts.findOneAndReplace({ "limit" : { $gt : 5000 } },{ "product" : "Commodity", "limit" : 2500 }, {returnNewDocument:true})
```

The output of the query is shown in Figure 6.7.

```
sample_analytics> db.accounts.findOneAndReplace({ "limit" : { $gt : 5000 } },{ "product" : "Commodity", "limit" : 2500 }, {returnNewDocument:true})
{
  _id: ObjectId("5ca4bbc7a2dd94ee5816238f"),
  product: 'Commodity',
  limit: 2500
}
```

Figure 6.7: Output of the `findOneAndReplace` Method

The replaced document is returned.

- **Find and Delete**

  MongoDB provides the `findOneAndDelete` method to locate a document and delete it based on the criteria specified in the method. The syntax for the `findOneAndDelete` method is:

```
db.collection.findOneAndDelete(filter, options)
```

Table 6.5 describes the parameters of the `findOneAndDelete` method.

Parameter	Fields	Type	Description
`filter`	-	`document`	Specifies the criteria based on which the document must be deleted; if empty, then the first document in the collection is deleted
`options`		`document`	Specifies optional fields
	`writeConcern`	`document`	Specifies the write concern of the document
	`projection`	`document`	Specifies the fields to be returned
	`sort`	`document`	Specifies the order in which the documents matched by the filter parameter must be sorted
	`maxTimeMS`	`number`	Specifies the maximum time in milliseconds within which the query must run; if exceeded, throws an error
	`collation`	`document`	Specifies language-specific rules such as lettercase

Table 6.5: Parameters for the `findOneAndDelete` Method

Consider that the user wants to find all the `products` that are '`Commodity`' and sort the returned documents in ascending order on the basis of the `limit` field. To delete the first such document returned, the user can execute the query as:

```
db.accounts.findOneAndDelete({ "products" : "Commodity"},
{sort: {"limit" : 1}})
```

The output of the query is shown in Figure 6.8.

```
sample_analytics> db.accounts.findOneAndDelete({ "products" : "Commodity"}, {sort: {"limit" : 1}})
{
  _id: ObjectId("5ca4bbc7a2dd94ee58162458"),
  account_id: 852986,
  limit: 7000,
  products: [
    'Derivatives',
    'Commodity',
    'CurrencyService',
    'InvestmentFund',
    'InvestmentStock'
  ]
}
```

Figure 6.8: Output of the `findOneAndDelete` Method

The deleted document is returned as the output of this method.

6.3 Database Methods

The database methods provided by MongoDB help to create, run, drop, and fetch data from collections and views in a database. Some of the database methods—`db.dropDatabase`, `db.runCommand`, `db.createCollection`, `db.getSiblingsDB`—were covered in previous sessions.

- **Create View**

  Views are read-only entities that are created from collections or other views in the same database. Views are created by running an aggregation pipeline on the source collection or view. MongoDB provides a method to create a view based on a collection or another view.

The syntax to create a view is:

```
db.createView(view, source, pipeline, collation)
                        (or)
db.createCollection(view, source, pipeline, collation)
```

Table 6.6 describes the parameters of the `createView` method.

Parameter	Type	Description
`view`	`string`	Specifies the name of the view to be created
`source`	`string`	Specifies the name of the source collection or view from which the view must be created
`pipeline`	`array`	Specifies the array that has the aggregation pipeline stages as its elements; cannot include the `$out` and `$merge` stages
`collation`	`document`	Specifies language-specific rules such as lettercase

Table 6.6: Parameters for the `createView` Method

Consider that the user wants to create a view named `AccountsLimit5000` from the `accounts` collection in the `sample_analytics` database. This view must contain only those documents where the value of the `limit` field is `5000`. To create this view, the user can execute the query as:

```
db.accounts.createView("AccountsLimit5000","accounts",
[{ $match: { limit: 5000 }}])
```

The output of the query is shown in Figure 6.9.

```
sample_analytics> db.createView("AccountsLimit5000", "accounts", [{ $match:
{ limit: 5000}} ])
{ ok: 1 }
```

Figure 6.9: Output of the `createView` Method

A new view named `AccountsLimit5000` has been created.

To fetch documents from this view, the user can execute the query as:

```
db.AccountsLimit5000.find()
```

The output of the query is shown in Figure 6.10.

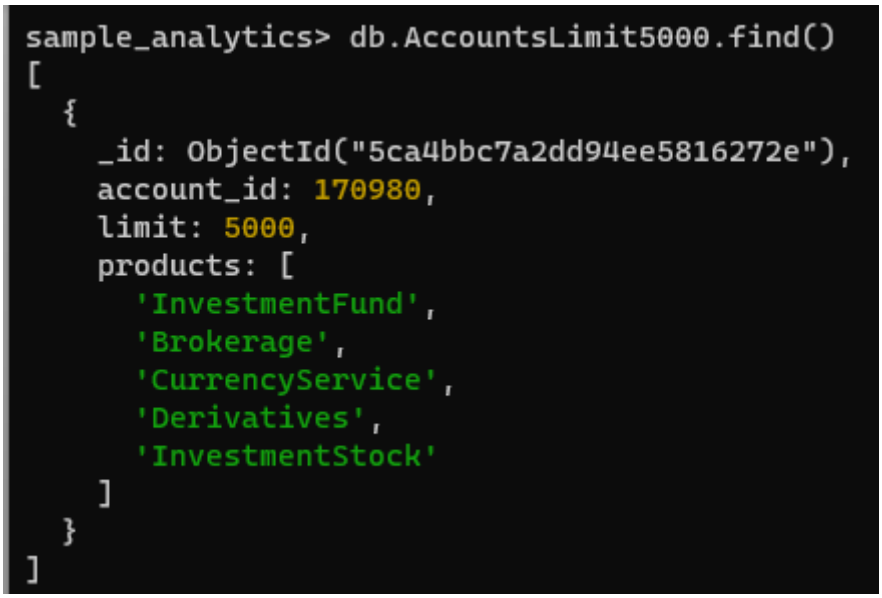


Figure 6.10: Fetch Documents from a View

- **Get Collection Names**

  MongoDB provides the `getCollectionNames` method to get the names of all the collections and views in a database.

  Consider that the user wants to get the names of all the collections and views in the `sample_analytics` database. To perform this task, the user can execute the query as:

```
db.getCollectionNames()
```

The output of the query is shown in Figure 6.11.

```
sample_analytics> db.getCollectionNames()
[ 'accounts', 'system.views', 'transactions', 'AccountsLimit5000' ]
```

Figure 6.11: Output of the `getCollectionNames` Method

- **Check Connection**

  MongoDB provides the `getMongo` method to test the connection between `mongosh` and the database instance. To test the connection, the user can execute the query as:

```
db.getMongo()
```

  The output of the query is shown in Figure 6.12.

```
sample_analytics> db.getMongo()
mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=20
00&appName=mongosh+1.8.2
```

Figure 6.12: Output of the `getMongo` Method

- **Statistics**

  MongoDB provides the `stats` method that returns the use state of a database. To get the usage statistics of a database, the user can execute the query as:

```
db.stats()
```

  The output of the query is shown in Figure 6.13.

```
sample_analytics> db.stats()
{
  db: 'sample_analytics',
  collections: 3,
  views: 1,
  objects: 3491,
  avgObjSize: 4686.686049842452,
  dataSize: 16361221,
  storageSize: 9797632,
  indexes: 3,
  indexSize: 114688,
  totalSize: 9912320,
  scaleFactor: 1,
  fsUsedSize: 159485407232,
  fsTotalSize: 511258390528,
  ok: 1
}
```

Figure 6.13: Output of the `stats` Method

6.4 Role Management Methods

The role management methods provided by MongoDB help to create, update, and drop roles. These methods also facilitate the users to grant and revoke privileges to roles. Let us discuss a few role management methods in MongoDB.

- **Create Role**

  MongoDB provides a method to create a role in the database in which the method is executed. The syntax to create a role is:

```
db.createRole(role, writeConcern)
```

  Table 6.7 describes the parameters of the `createRole` method.

Parameter	Field	Type	Description
`role`	`role`	`string`	Specifies the name of the role to be created
	`privileges`	`array`	Specifies the privileges that must be granted to the role; if empty, no privileges are granted
	`roles`	`array`	Specifies an array of roles from which privileges must be inherited for this role; if empty, no roles are inherited
	`authentication Restrictions`	`array`	Specifies the authentication restrictions enforced on the role
`writeConcern`		`document`	Specifies the level of write concern

Table 6.7: Parameters of the `CreateRole` Method

  Consider that the user wants to create a new role named `mynewrole` on the `sample_analytics` database from the `admin` database.

To create a new role, the user can execute the query as:

```
use admin

db.createRole({ role: "mynewrole", privileges: [{
resource: { db: "sample_analytics", collection: "accounts"
}, actions: ["update", "insert", "remove"] }, { resource:
{ db: "", collection: "" }, actions: ["find"] }], roles:
[{ role: "readWrite", db: "sample_analytics" }] }, { w:
"majority", wtimeout: 5000 })
```

The output of the query is as shown in Figure 6.14.

```
sample_analytics> use admin
switched to db admin
admin> db.createRole({ role: "mynewrole", privileges: [{ resource: { db: "saadmin>
 db.createRole({ role: "mynewrole", privileges: [{ resource: { db: "sample_analyti
cs", collection: "accounts" }, actions: ["update", "insert", "remove"] }, { resour
ce: { db: "", collection: "" }, actions: ["find"] }], roles: [{ role: "readWrite",
 db: "sample_analytics" }] }, { w: "majority", wtimewtimeout: 5000 })
{ ok: 1 }
```

Figure 6.14: Output of the `createRole` Method

The `mynewrole` role is created.

- **Get Role**

MongoDB provides the `getRole` method to get the roles from which privileges are inherited for this role. The syntax for this method is:

```
db.getRole(rolename, arguments)
```

Table 6.8 describes the parameters of the `getRole` method.

Parameter	Field	Type	Description
`rolename`	-	`string`	Specifies the name of the role
`arguments`	`showAuthentication Restrictions`	`boolean`	Includes the IP address from which the users of this role can connect to the database as part of the output if set to `true`

Parameter	Field	Type	Description
	showBuiltinRoles	boolean	Includes built-in roles as part of the output if set to true
	showPrivileges	boolean	Includes direct privileges and privileges inherited from other roles as part of the output if set to true

Table 6.8: Parameters of the `getRole` Method

Consider that the user wants to get the details about the `mynewrole` role. To perform this task, the user can execute the query as:

```
db.getRole( "mynewrole" )
```

The output of the query is shown in Figure 6.15.

```
admin> db.getRole("mynewrole")
{
  _id: 'admin.mynewrole',
  role: 'mynewrole',
  db: 'admin',
  roles: [ { role: 'readWrite', db: 'sample_analytics' } ],
  inheritedRoles: [ { role: 'readWrite', db: 'sample_analytics' } ],
  isBuiltin: false
}
```

Figure 6.15: Output of the `getRole` Method

Consider that the user wants to get the roles and privileges of the `mynewrole` role. To perform this task, the user can execute the query as:

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

The output of the query is shown in Figure 6.16.

```
admin> db.getRole( "mynewrole", { showPrivileges: true } )
{
  _id: 'admin.mynewrole',
  role: 'mynewrole',
  db: 'admin',
  privileges: [
    {
      resource: { db: 'sample_analytics', collection: 'accounts' },
      actions: [ 'insert', 'remove', 'update' ]
    },
    { resource: { db: '', collection: '' }, actions: [ 'find' ] }
  ],
  roles: [ { role: 'readWrite', db: 'sample_analytics' } ],
  inheritedRoles: [ { role: 'readWrite', db: 'sample_analytics' } ],
  inheritedPrivileges: [
    {
      resource: { db: 'sample_analytics', collection: 'accounts' },
      actions: [ 'insert', 'remove', 'update' ]
    },
    { resource: { db: '', collection: '' }, actions: [ 'find' ] },
    {
      resource: { db: 'sample_analytics', collection: '' },
      actions: [
        'changeStream',
```

Figure 6.16: Output of the `getRole` Method with Privileges

The roles and privileges associated with the `mynewrole` role are displayed as output.

- **Update Role**

MongoDB provides the `updateRole` method to update privileges, roles, and restrictions of a user-defined role. The syntax for this method is:

```
db.updateRole(rolename, update, writeConcern)
```

Table 6.9 describes the parameters of the `updateRole` method.

Parameter	Field	Type	Description
`rolename`	-	`string`	Specifies the name of the role
`update`	`privileges`	`array`	Specifies the privileges that must be granted to the role; mandatory if `roles` field is not mentioned
	`roles`	`array`	Specifies the roles from which

Parameter	Field	Type	Description
			this role inherits; mandatory if `privileges` field is not mentioned
	`authentication Restrictions`	`array`	Specifies the authentication restrictions enforced by the server on the role such as IP addresses from which the users of this role can connect to the database

Table 6.9: Parameters of the `updateRole` Method

Consider that the user wants to update the roles and privileges of the `mynewrole` role. To perform this task, the user can execute the query as:

```
use admin

db.updateRole("mynewrole", { privileges: [{ resource:
{ db: "sample_analytics", collection: "accounts" },
actions: ["update", "createCollection",
"createIndex"] }], roles: [{ role: "read", db:
"sample_analytics" }] }, { w: "majority" })
```

The output of the query is shown in Figure 6.17.

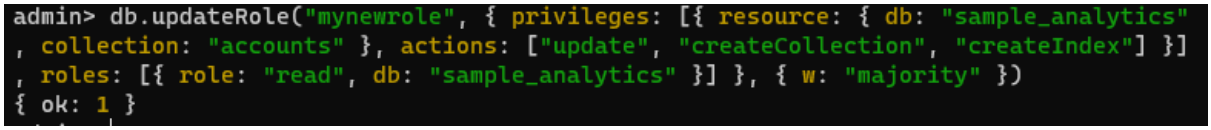


Figure 6.17: Output of the `updateRole` Method

To view the updated role, the user can execute the query as:

```
db.getRole( "mynewrole")
```

The output of the query is shown in Figure 6.18.

```
admin> db.getRole( "mynewrole")
{
  _id: 'admin.mynewrole',
  role: 'mynewrole',
  db: 'admin',
  roles: [ { role: 'read', db: 'sample_analytics' } ],
  inheritedRoles: [ { role: 'read', db: 'sample_analytics' } ],
  isBuiltin: false
}
```

Figure 6.18: Output After the Role Update

It can be seen that the roles of the `mynewrole` role is updated.

- **Drop Role**

  MongoDB provides the `dropRole` method to delete a user-defined role from the database in which the role was created. The syntax for this method is:

```
db.dropRole(rolename, writeConcern)
```

Table 6.10 describes the parameters of the `dropRole` method.

Parameter	Field	Type	Description
`role`	`role`	`string`	Specifies the name of the role to be dropped
`writeConcern`		`document`	Specifies the level of write concern

Table 6.10: Parameters of the `dropRole` Method

Consider that the user wants to drop the `mynewrole` role from the `admin` database. To drop the role, the user can execute the query as:

```
db.dropRole( "mynewrole", { w: "majority" } )
```

The output of the query is shown in Figure 6.19.

```
admin> db.dropRole( "mynewrole", { w: "majority" } )
{ ok: 1 }
```

Figure 6.19: Output of the `dropRole` Method

The `mynewrole` method is dropped.

6.5 User Management Methods

User management methods provided by MongoDB help to create a user, get roles to the user, change user password, and drop the user. Let us now learn about a few user management methods in MongoDB.

- **Create User**

  MongoDB provides the `createUser` method to create a user in the database in which the method is executed. The syntax to create a user is:

```
db.createUser(user, writeConcern)
```

Table 6.11 describes the parameters of the `createUser` method.

Parameter	Field	Type	Description
`user`	`user`	`string`	Specifies the name of the user to be created
	`pwd`	`string`	Specifies the password of the user which can be a cleartext string or a `passwordPrompt` that prompts the user for password
	`customData`	`document`	Contains random information about the user that the admin wishes to store such as employee ID or full name of the user
	`roles`	`array`	Specifies the roles assigned to the user
	`authentication Restriction`	`array`	Specifies the authentication restrictions enforced on the user

Parameter	Field	Type	Description
writeConcern		document	Specifies the level of write concern

Table 6.11: Parameters of the `CreateUser` Method

Consider that the user wants to create a user named `user_account1` with `readWrite` and `dbAdmin` roles for the `accounts` collection in the `sample_analytics` database. To create a new user, the user can execute the query as:

```
db.createUser({ user: "user_account1", pwd:
passwordPrompt(), roles: ["readWrite", "dbAdmin"] })
```

When executed, the query prompts for a password. The user can enter `account1` as the password. The output of the query is shown in Figure 6.20.

```
switched to db sample_analytics
sample_analytics> db.createUser({user: "user_account1", pwd: passwordPrompt(),
...       roles: [ "readWrite", "dbAdmin" ] })
Enter password
********{ ok: 1 }
```

Figure 6.20: Output of the `createUser` Method

The user named `user_account1` is created. Users can also be created without specifying any roles.

- **Change User Password**

MongoDB provides the `changeUserPassword` method to change the password for the user created in the current database. The syntax for this method is:

```
db.changeUserPassword(username, password)
```

Table 6.12 describes the parameters of the `changeUserPassword` method.

Parameter	Type	Description
username	string	Specifies the name of the user for whom the password must be changed
password	string	Specifies the new password of the user which can be a cleartext string or a passwordPrompt that prompts the user for password

Table 6.12: Parameters of the changeUserPassword Method

Consider that the user wants to change the password for the user_account1 user. To change the password, the user can execute the query as:

```
db.changeUserPassword("user_account1", passwordPrompt())
```

When executed, the query prompts for a password. Enter user1 as the new password. The output of the query is shown in Figure 6.21.

```
sample_analytics> db.changeUserPassword("user_account1", passwordPrompt())
Enter password
*****{ ok: 1 }
```

Figure 6.21: Output of the changeUserPassword Method

The password for the user_account1 user is now changed to user1.

- **Get User**

MongoDB provides the getUser method to get user details such as password, privileges, and custom data of the user. The syntax for this method is:

```
db.getUser(username, args)
```

Table 6.13 describes the parameters of the `getUser` method.

Parameter	Field	Type	Description
username	-	string	Specifies the name of the user for whom the details must be obtained
args	showCredentials	boolean	Displays the password of the user if set to `true`
	showCustomerData	boolean	Displays the custom data if set to `true`
	showPrivileges	boolean	Displays the entire privilege set of the user if set to `true`
	showAuthenticationRestrictions	boolean	Displays the authentication restrictions of the user such as the allowed IP addresses to connect to the database if set to `true`

Table 6.13: Parameters of the `getUser` Method

Consider that the user wants to get details about the `user_account1` user in the `sample_analytics` database. To perform this task, the user can execute the query as:

```
db.getUser("user_account1")
```

The output of the query is shown in Figure 6.22.

```
sample_analytics> db.getUser("user_account1")
{
  _id: 'sample_analytics.user_account1',
  userId: new UUID("19cb9e6d-f088-402c-b9e9-7a041dee3321"),
  user: 'user_account1',
  db: 'sample_analytics',
  roles: [
    { role: 'readWrite', db: 'sample_analytics' },
    { role: 'dbAdmin', db: 'sample_analytics' }
  ],
  mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
}
```

Figure 6.22: Output of the `getUser` Method

- **Drop User**

  MongoDB provides the `dropUser` method to drop the user from the database.

  The syntax for this method is:

```
db.dropUser(username, writeConcern)
```

  Table 6.14 describes the parameters of the `dropUser` method.

Parameter	Field	Type	Description
username	-	string	Specifies the name of the user for whom the details must be obtained
args	showCredentials	boolean	Displays the password of the user if set to true
	showCustomerData	boolean	Displays the user data if set to true
	showPrivileges	boolean	Displays the entire privilege set of the user if set to true
	showAuthenticationRestrictions	boolean	Displays the authentication restrictions of the user if set to true

Table 6.14: Parameters of the `dropUser` Method

Consider that the user wants to drop the `user_account1` user in the `sample_analytics` database. To perform this task, the user can execute the query as:

```
db.dropUser("user_account1", {w: "majority",
wtimeout: 5000})
```

The output of the query is shown in Figure 6.23.

```
sample_analytics> db.dropUser("user_account1", {w: "majority", wtimeout: 5000})
{ ok: 1 }
```

Figure 6.23: Output of the `dropUser` Method

6.6 Summary

- MongoDB Shell provides various methods such as collection methods, database methods to create, insert, update, and delete documents, collections, and databases.
- MongoDB Shell includes methods to locate documents based on specified conditions and update, replace, or delete the first document in the output.
- MongoDB allows views to be created by running aggregation pipelines on the source collections or other views.
- MongoDB Shell provides methods that fetch statistical information about the collections or users.
- MongoDB Shell also provides role management methods and user management methods to create, update, and drop roles and users in a database.

Test Your Knowledge

1. Which of the following options are used to count all the documents in the `inventory` collection?

   a. `db.inventory.countDocuments({})`
   b. `db.inventory.countDocuments()`
   c. `db.inventory.countDocuments().all()`
   d. `db.inventory.countAllDocuments()`

2. Which of the following option is the correct syntax of the `findOneAndUpdate` method?

   a. `db.collection.findOneAndUpdate(filter, query, update)`
   b. `db.collection.findOneAndUpdate(field, update, options)`
   c. `db.collection.findOneAndUpdate(filter, update, options)`
   d. `db.collection.findOneAndUpdate(query,field,update)`

3. Consider that you have created a collection named `product_price` with the fields: `product_id`, `product_name`, and `product_price`. Which of the following query will you use to create a view named `Product_price_lessthan_2000` from the `product_price` collection in the product database?

   a. `db.createView("Product_price_lessthan2000","product",[{ $match: { product_price: {$lt:2000} }}])`
   b. `db.product.createView("Product_price_lessthan2000","product", [{ $match: { product_price: {$lt:2000} }}])`
   c. `db.createView("Product_price_lessthan2000","product",[{ $group: { product_price: {$lt:2000} }}])`
   d. `db.createView("Product_price_lessthan2000","product",[{ $project: { product_price: {$lt:2000} }}])`

4. Which of the following query will you use to display the roles and privileges of the role named `user1_role` from the `product` database.

   a. `db.viewRole( "user1_role", { privileges: true } )`
   b. `db.getRole( "user1_role", { showPrivileges: true } )`
   c. `db.product.getRole( "user1_role", { showPrivileges: true } )`
   d. `db.viewRolePrivileges( "mynewrole", { showPrivileges: true } )`

5. Which of the following user management method will allow you to change the password of a user for the current database?

   a. `db.userPasswordchange(username, passoword)`
   b. `db.PasswordChange(username, passoword)`
   c. `db.userChangePassword(username, passoword)`
   d. `db.changeUserPassword(username, passoword)`

Answers to Test Your Knowledge

1	a, b
2	c
3	a
4	b
5	d

Try it Yourself

1. Create a database named `Student` and a collection named `sub_marks`.
2. Insert the following four documents into the `sub_marks` collection.

```
[
{
Name: "Wilson Churchil",
rollno:1234,
age:16,
program_language:["C","C++", "Java"],
Total_marks:280
},
{
Name: "Adam Smith",
rollno:2457,
age:17,
program_language:["Java","C++", "Python"],
Total_marks:270
},
{
Name: "Rita Richard",rollno:1221,
age:15,
program_language:["Perl","Ruby", "Java"],Total_marks:268
},
{
Name: "Michael Franklin",
rollno:2134,
age:15,
program_language:["Python", "Java","C#"],Total_marks:290
}
]
```

Using the `sub_marks` collection, perform the following tasks:

3. Count the number of students who have learned '`java`' as one of the subjects.
4. Fetch the distinct subjects learned by students.
5. Use the collection method to find the student named '`Rita Richard`' and update the `Total_marks` field as `288`.
6. Create a view named `subject_marks` which contains only those documents where the `Total_marks` field is greater than `275`.

7. Display the use state of the database `Student`.
8. Create a new role named `role1` in the `Student` database and grant permissions to `insert`, `update`, and `remove` data from the `stu_marks` collection.
9. Display detailed information about the `role1` role.
10. Update `role1` and grant permission to `update` the new collection `stu_detail` in the `Student` database. Also, add a new role named `user2` and grant `read` permission to the `stu_detail` collection.
11. Drop `role1` from the `Student` database.
12. Create a user `stu_user1` for the `stu_mark` collection with `readwrite` permission.
13. Display the details of the user `stu_user1` with password.