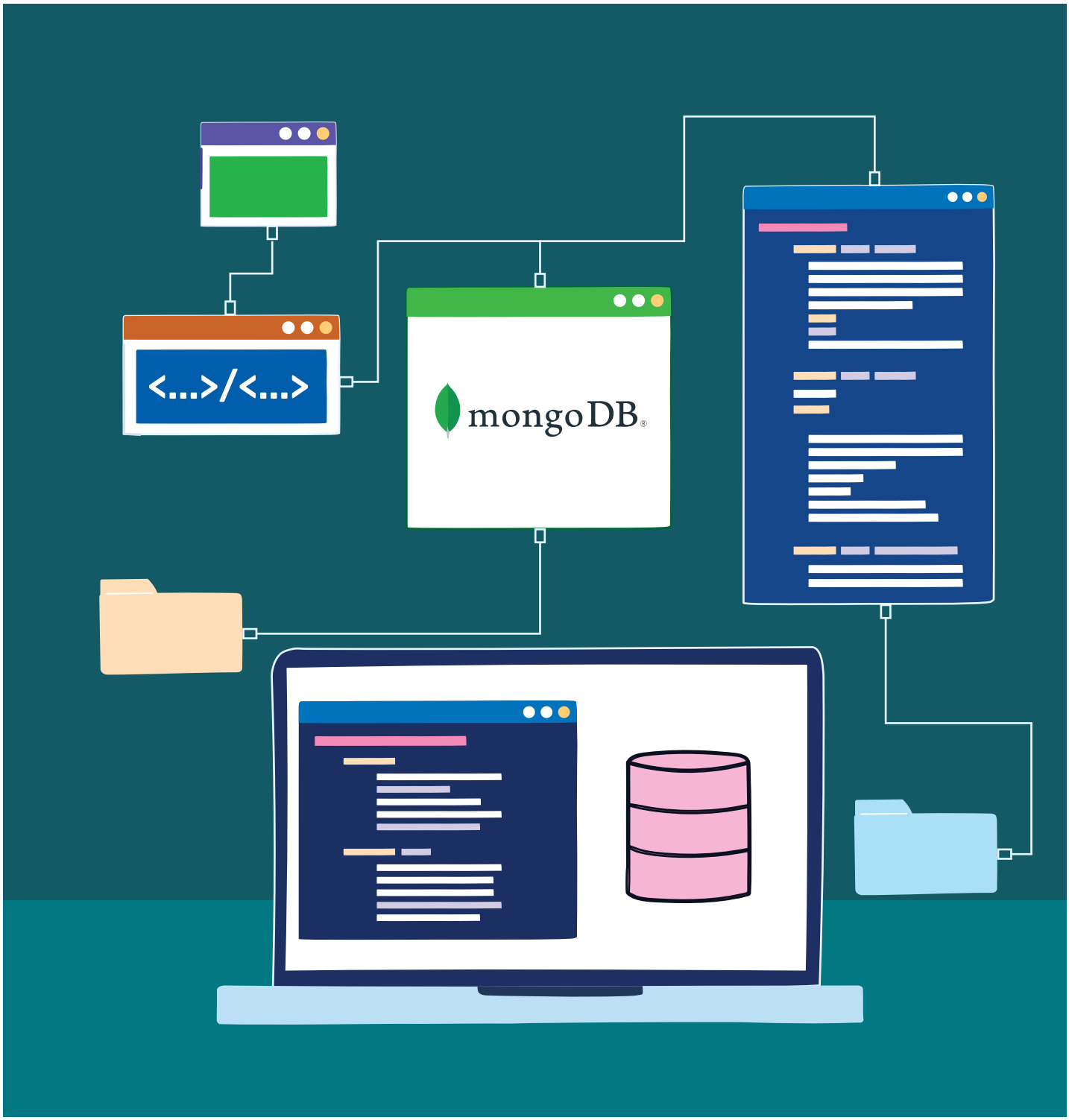
Cheatsheet

SCALER Topics



MONGODB CHEATSHEET

India's best tech learning company

Learn industry-relevant skills with top tech veterans

126% Avg. CTC Hike | **Top 1%** Industry Instructors | **900+** Placement Partners

Programs We Offer

- Software Development Course
- Data Science & Machine Learning

The Scaler Recipe to Transform Your Career

- A structured & flexible program, that cares for you
- Be Mentored 1:1 by Experienced Professionals
- Become part of a thriving community for life

Discover & connect with Alumni

Sudhanshu Gera
Software Engineer III

Pre Scaler: Wipro Limited → Post Scaler: Walmart
200% Hike

Ankit Pangasa
Senior Software Engineer

Pre Scaler: Adobe → Post Scaler: Google
200% Hike

Connect with Alum

SCALER TOPICS

Unlock your potential in software development with
FREE COURSES from **SCALER TOPICS!**

Register now and take the first step towards your future Success!

C++ for Beginners

5.9k enrolled ₹ Free

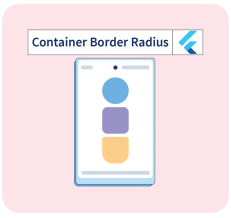
Java for Beginners

6.8k enrolled ₹ Free

That's not it. Explore 20+ Courses by clicking below

Explore Other Courses

Practice **CHALLENGES**
and become 1% better everyday

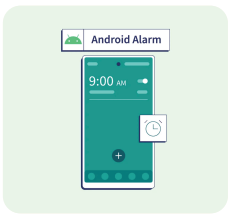


How to Set Border Radius for Container in Flutter?

Article

No. Of Questions : 3

Go to Challenge >



Android Alarm

Article

No. Of Questions : 3

Go to Challenge >

Explore Other Challenges

Basic Commands

Command	Explanation
mongosh	Open a connection to your local MongoDB instance. All other commands will be run within this mongosh connection.
db.help()	Show help for database methods.
db	Show current database name
cls	Clear the terminal screen
show collections	Shows all the collections
show dbs	Show all databases in the current MongoDB instance
show databases	Print a list of all existing databases available to the current user.
use DATABASE_NAME	Creates a new database with the specified name. If not exists, else switch to that database.
exit	Exit the mongosh session

CRUD Commands

CREATE SCHEMA

Mongoose command used to define the structure of a collection in a MongoDB database.

```
Const <schema_name> = new mongoose schema({
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
...
});
const <model_name> =
mongoose.model(<model_name>,
<Schema_name>);
```

EXAMPLES:

Create a collection called “customers” with columns for customer_id, customer_name, contact_no, and an integer field called age:

```
const customerSchema = new
mongoose.Schema({
customer_id: Number,
customer_name: String,
contact_no: String,
age: Number,
});
const Customer =
mongoose.model('Customer',
customerSchema);
```

INSERT INFO

Mongoose command used to insert data into a collection in a MongoDB database.

```
db.collection.insertOne({
"field1": value1,
"field2": value2,
// ...
});

db.collection.insertMany([
{
"field1": value1,
"field2": value2,
// ...
},
{
"field1": value4,
"field2": value5,
// ...
}
]);
```

EXAMPLES:

The following code1 inserts a document into a collection called users with the fields name and age. In case 2, it inserts two documents into the products collection.

```
db.users.insertOne({
"name": "John Doe",
"age": 30
});

db.products.insertMany([
{
"name": "iPhone 13",
"price": 999
},
{
"name": "Samsung Galaxy S22",
"price": 899
}
]);
```

READ COMMAND	EXAMPLES:
Mongoose command used to read data from a collection in a MongoDB database	The following code retrieves all documents from the users collection where the name field is equal to "John Doe":
`db.collection.find(queryDocument);`	`db.users.find({ "name": "John Doe" });`

Some more Read commands are:

Command	Explanation	Example
db.collection.find()	Finds all documents in the collection.	db.users.find()
db.collection.find({ <query> })	Finds documents that match the specified query criteria.	db.users.find({ "name": "John Doe" })
db.collection.findOne()	Finds a single document that matches the specified query criteria.	db.users.findOne({ "name": "John Doe" })
db.collection.find().limit(N)	Limits the number of documents returned to N.	db.users.find().limit(10)
db.collection.find().skip(N)	Skips the first N documents.	db.users.find().skip(10)
db.collection.find().sort({ <field>: <direction> })	Sorts the documents in the specified order.	db.users.find().sort({ "name": 1 })
db.collection.find().explain()	Shows the execution plan for the query.	db.users.find().explain()

UPDATE COMMANDS	EXAMPLES:
It is used to update data in a collection in a MongoDB database.	The code1 update the age of a person name “John Doe” to 31, code2 increase the price of all the products that are priced at 899 to 999.

`db.collection.updateOne(QUERY, UPDATE)`
 	
`db.collection.updateMany(QUERY, UPDATE)`
	`db.collection.updateOne({ "name": "John Doe" },`
	`{ "$set": { "age": 31 } });`
	
	`db.products.updateMany({ "price": 899 },`
	`{ "$set": { "price": 999 } });`

Some most common update operators in MongoDB:

Operator	Explanation	Example
$set	Sets the value of a field.	{ "$set": { "name": "John Doe" } }
$inc	Increments the value of a field by a specified amount.	{ "$inc": { "age": 1 } }
$dec	Decrements the value of a field by a specified amount.	{ "$dec": { "age": 1 } }
$mul	Multiplies the value of a field by a specified amount.	{ "$mul": { "price": 2 } }
$rename	Renames a field.	{ "$rename": { "oldName": "newName" } }
$unset	Removes a field from a document.	{ "$unset": { "address": "" } }
$push	Adds a value to an array field.	{ "$push": { "tags": "MongoDB" } }
$addToSet	Adds a value to an array field only if the value does not already exist in the array.	{ "$addToSet": { "tags": "MongoDB" } }
$pop	Removes the first or last element from an array field.	{ "$pop": { "tags": 1 } }
$pull	Removes all elements from an array field that match a specified value.	{ "$pull": { "tags": "MongoDB" } }
$pullAll	Removes all elements from an array field that match a specified array of values.	{ "$pullAll": { "tags": ["MongoDB", "Python"] } }

DELETE COMMANDS

MongoDB command used to delete data from a collection in a MongoDB database.

```
1: db.collection.deleteOne(queryDocument);

2:db.collection.deleteMany(queryDocument);
```

EXAMPLES:

The code1 deletes the first document from the products collection where the name field is equal to "iPhone 13". In code 2, it deletes all products having a “price” of 899.

```
db.products.deleteOne({ "name": "iPhone
13" });

db.products.deleteMany({ "price": 899
});
```

Comparison Query Operators

It is used to compare the values of two fields or a field and a value in a document.

Operator	Explanation	Example
$eq	Matches values that are equal to a specified value.	Finds all users with the operating system macOS. `db.users.find({ system: { $eq: "macOS" } })`
$gt	Matches values that are greater than a specified value.	Deletes all users with an age greater than 99. `db.users.deleteMany( { age: { $gt: 99 } })`
$gte	Matches values that are greater than or equal to a specified value.	Updates all access to "valid" for users with an age greater than or equal to 21. `db.users.updateMany( { age: { $gte: 21 } }, { access: "valid" })`
$in	Matches any of the values specified in an array.	Finds users with the place field as NYC or SF. `db.users.find({ place: { $in: ["NYC", "SF"] } })`
$lt	Matches values that are less than a specified value.	Deletes all users with an age less than 18. `db.users.deleteMany( { age: { $lt: 18 } })`
$lte	Matches values that are less than or equal to a specified value.	Updates all access to "invalid" for users with an age less than or equal to 17. `db.users.updateMany( { age: { $lte: 17 } }, { access: "invalid" })`

$ne	Matches all values that are not equal to a specified value.	Finds users with the place field set to anything other than NYC. `db.users.find({ place: { $ne: 'NYC' } })`
$nin	Matches none of the values specified in an array.	Finds users with the place field not equal to NYC or SF. `db.users.find({ place: { $nin: ["NYC", "SF"] } })`

Indexing Commands

Indexing is the process of creating a special data structure that allows MongoDB to quickly find documents that match a query.

Command	Explanation	Example
db.collection.createIndex()	Builds an index on a collection.	Creates the "account creation date" index in the "users" collection in ascending order. `db.users.createIndex({ "account creation date": 1 })`
db.collection.dropIndex()	Removes a specified index on a collection.	Removes the "account creation date" index from the "users" collection. `db.users.dropIndex({ "account creation date": 1 })`
db.collection.dropIndexes()	Removes all indexes but the **_id** (no parameters) or a specified set of indexes on a collection.	Drops all indexes except the _id index in the "users" collection. `db.users.dropIndexes()`
db.collection.getIndexes()	Returns an array of documents that describe the existing indexes on a collection.	Retrieves a list of indexes defined in the "users" collection. `db.users.getIndexes()`
db.collection.reIndex()	Rebuilds all existing indexes on a collection.	Drops and recreates all indexes on the "users" collection. `db.users.reIndex()`
db.collection.totalIndexSize()	Reports the total size used by the indexes on a collection. Provides a wrapper around the totalIndexSize field of the collStats output.	Retrieves the total size of all indexes for the "users" collection `db.users.totalIndexSize()`

Sharding Commands

Sharding is a method for distributing or partitioning data across multiple computers. This is done by partitioning the data by key ranges and distributing the data among two or more database instances.

Command	Explanation	Example
sh.abortReshardCollection()	Aborts a running reshard operation.	`sh.abortReshardCollection ("users")` Aborts a running reshard operation on the users collection.
sh.addShard()	Adds a shard to a sharded cluster.	`sh.addShard("cluster/ mongodb3.example.net:27327")` Adds the cluster replica set and specifies one member of the replica set.
sh.commitReshardCollection()	Forces a resharding operation to block writes and complete.	`sh.commitReshardCollection ("records.users")` Forces the resharding operation on the records.users to block writes and complete.
sh.disableBalancing()	Disables balancing on a single collection in a sharded database. Does not affect balancing of other collections in a sharded cluster.	`sh.disableBalancing ("records.users")` Disables the balancer for the specified sharded collection.
sh.enableAutoSplit()	Enables auto-splitting for the sharded cluster.	`sh.enableAutoSplit()`
sh.enableSharding()	Enables sharding for a collection.	`sh.enableSharding("records")` Enables sharding for the records database.

Command	Explanation	Example
sh.help()	Returns help text for the sh methods.	`sh.help()`
sh.moveChunk()	Migrates a chunk in a sharded cluster.	`sh.moveChunk("records.users", { zipcode: "10003" }, "shardexample")` Finds the chunk that contains the documents with the zipcode field set to 10003 and then moves that chunk to the shard named shardexample.
sh.reshardCollection()	Initiates a resharding operation to change the shard key for a collection, changing the distribution of your data.	`sh.reshardCollection("records.users", { order_id: 1 })` Reshards the users collection with the new shard key { order_id: 1 }.
sh.shardCollection()	Enables sharding for a collection.	`sh.shardCollection ("records.users", { zipcode: 1 })` Shards the users collection by the zipcode field.
sh.splitAt()	Divides an existing chunk into two chunks using a specific value of the shard key as the dividing point.	`sh.splitAt( "records.users", { x: 70 })` Splits a chunk of the records.users collection at the shard key value x: 70.
sh.splitFind()	Divides an existing chunk that contains a document matching a query into two approximately equal chunks.	`sh.splitFind ( "records.users", { x:70 })` Splits, at the median point, a chunk that contains the shard key value x: 70.
sh.status()	Reports on the status of a sharded cluster, as db.printShardingStatus().	`sh.status()`

Command	Explanation	Example
sh.waitForPingChange()	Internal. Waits for a change in ping state from one of the mongos in the sharded cluster.	`sh.waitForPingChange()` Internal. Waits for a change in ping state from one of the mongos in the sharded cluster.
refineCollectionShardKey()	Modifies the collection's shard key by adding new field(s) as a suffix to the existing key.	`refineCollectionShardKey ("test.orders", { customer_id: 1, order_id: 1 })` Modifies the collection's shard key by adding new field(s) as a suffix to the existing key.
convertShardKeyToHashed()	Returns the hashed value for the input.	`convertShardKeyToHashed(ObjectId("5b2be413c06d924ab26ff9ca"))` Returns the hashed value for the input.

User Management Commands

Make updates to users in the MongoDB Shell.

Command	Explanation
db.auth()	Authenticates a user to a database.
db.changeUserPassword()	Updates a user's password.
db.createUser()	Creates a new user for the database on which the method is run.
db.dropAllUsers()	Removes all users from the current database.
db.getUsers()	Returns information for all users in the database.
db.grantRolesToUser()	Grants a role and its privileges to a user.
db.removeUser()	Removes the specified username from the database.
db.revokeRolesFromUser()	Removes one or more roles from a user on the current database.
db.updateUser()	Updates the user's profile on the database on which you run the method.
passwordPrompt()	Prompts for the password in mongosh shell.

Role Management Commands

Make updates to roles in the MongoDB Shell.

Command	Explanation
db.createRole()	Creates a new user-defined role.
db.dropRole()	Deletes a user-defined role.
db.dropAllRoles()	Deletes all user-defined roles associated with a database.
db.getRole()	Returns information for the specified role.
db.getRoles()	Returns information for all the user-defined roles in a database.
db.grantPrivilegesToRole()	Assigns privileges to a user-defined role.
db.revokePrivilegesFromRole()	Removes the specified privileges from a user-defined role.
db.grantRolesToRole()	Specifies roles from which a user-defined role inherits privileges.
db.revokeRolesFromRole()	Removes inherited roles from a role.
db.updateRole()	Updates a user-defined role.

Some more mongo DB commands:

Command	Explanation	Example
aggregate()	Performs an aggregation operation on a collection.	`db.collection.aggregate ([{$group: { _id: "$name", count: {$sum: 1} }}])`
$group()	Groups documents by the specified field and performs aggregation operations on the groups.	`{$group: { _id: "$name", count: {$sum: 1} }}`
$match()	Filters documents that match the specified query criteria.	`{$match: { "age": { $gt: 30 } }}`
$project()	Selects the fields that should be included in the output documents.	`{$project: { "name": 1, "age": 1 }}`
$sort()	Sorts the documents in the specified order.	`{$sort: { "age": 1 }}`
$limit()	Limits the number of documents that are returned.	`{$limit: 10}`
$skip()	Skips the first N documents.	`{$skip: 10}`
$unwind()	Unwinds an array field into multiple documents.	`{$unwind: "$orders"}`
$lookup()	Performs a join operation on two collections.	`{$lookup: { from: "orders", localField: "_id", foreignField: "customerId" }}`
$facet()	Performs a multi-level aggregation.	`{$facet: { "age": [{$group: { _id: "$age", count: {$sum: 1} }}], "name": [{$group: { _id: "$name", count: {$sum: 1} }]}}`