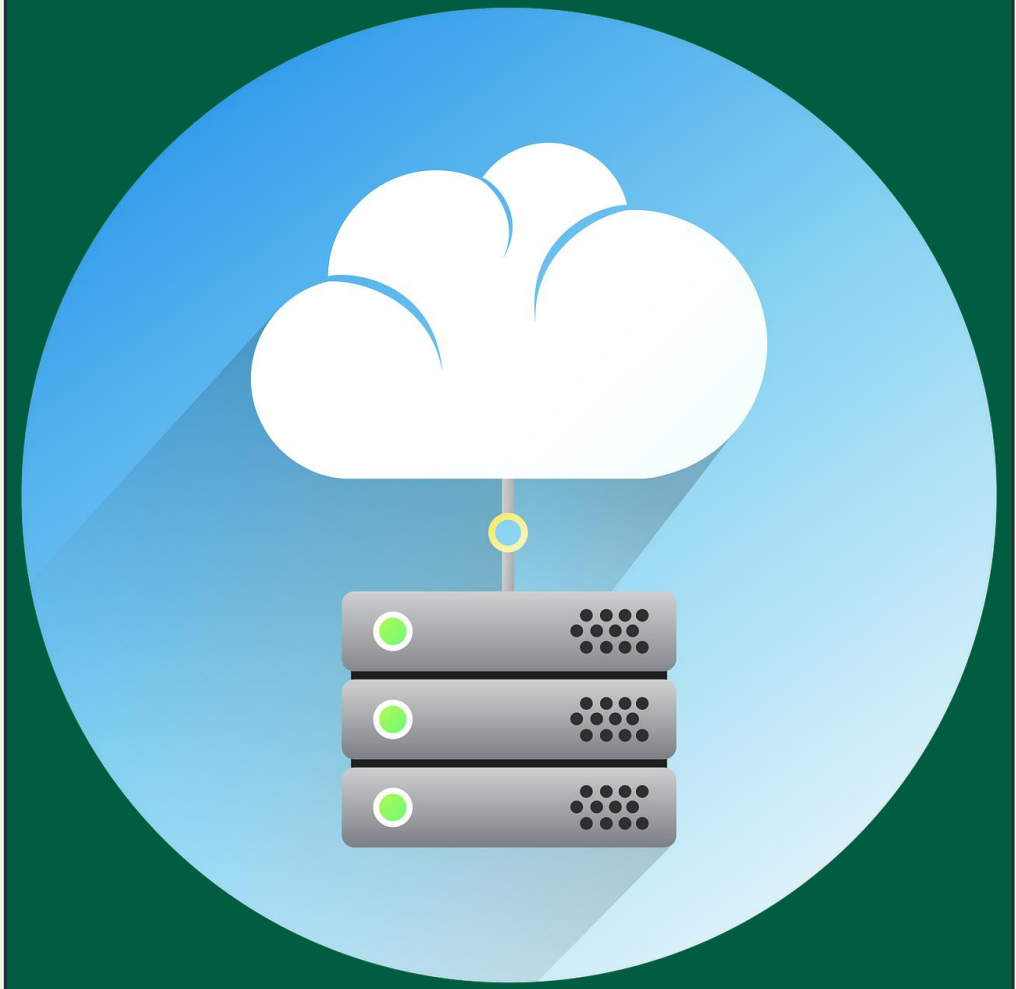
Session: 1

Introduction to MongoDB

Managing Large Datasets Using MongoDB



Objectives

- Explain the various types of databases
- Describe how data is stored in the MongoDB Server
- Explain the process of installing MongoDB Server and MongoDB Shell

Databases

A database is an organized collection of data that facilitates easy storage, access, and management of data electronically.

Hierarchical

Data is stored in the form of parent-child nodes.

Relational

Data is stored in the form of rows and columns that together form a table.

Object Oriented

Data is stored in the form of objects.

Network

Data is stored in the form of tree of records. Each parent node has multiple children nodes.

NoSQL

Data is stored in documents, graphs, and key-value pairs.

Relational and NoSQL Databases

ATTRIBUTES	RELATIONAL DATABASE	NoSQL DATABASE
Data Storage	Data is stored in tables as rows and columns	Data is stored as documents, key-value pairs, and graphs.
Schema	Has a defined schema	Supports dynamic schema
Scaling	Supports vertical scaling	Supports horizontal scaling
Performance	Multiple tables must be accessed to get data on a single entity causing decrease in performance	Data on a single entity is stored in the same location resulting in improved performance
Consistency	RDBMS support Atomicity, Consistency, Integrity Durability (ACID) properties	NoSQL databases mostly do not support ACID properties
Application	Best choice for data that must be related and stored in rule-based manner	Best choice for large volumes of data

Overview of MongoDB

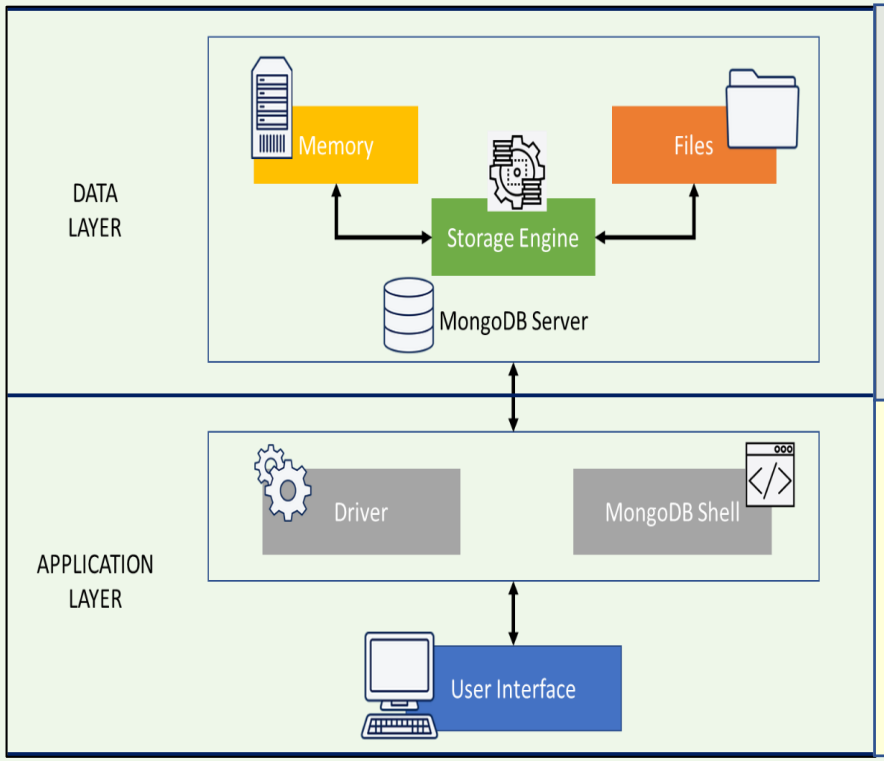
MongoDB is an open source, schema-less database that stores information in Binary JavaScript Object Notation (BSON).

MongoDB is designed to store large volume of data in a highly scalable environment.

It offers high flexibility and performance to users while creating and querying the data.

MongoDB Architecture

MongoDB Layers and its Vital Components



Data Layer

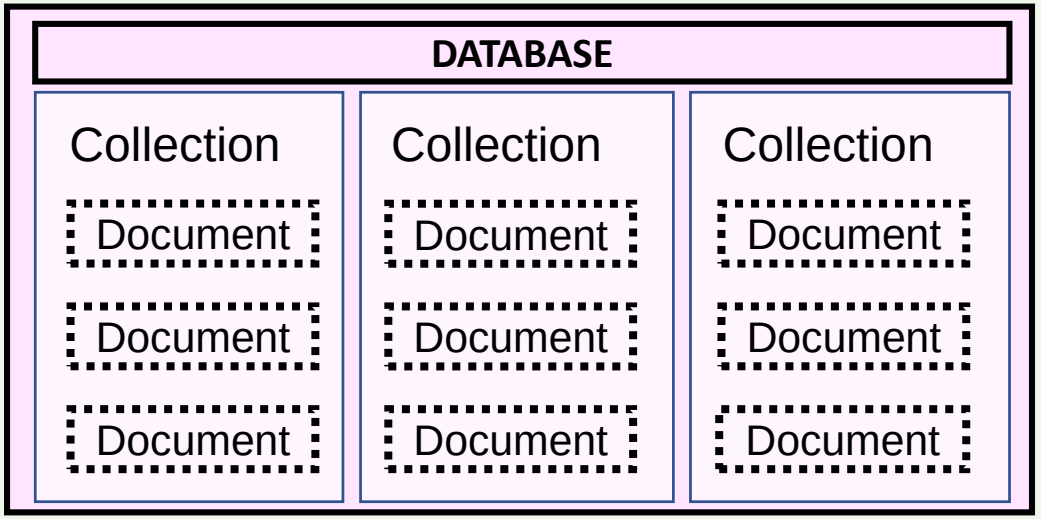
MongoDB server receives queries from the application layer and forwards them to the storage engine. The storage engine reads or writes data to the memory or the file. The storage engine decides the amount of data stored on the file and memory and how it is stored.

Application Layer

The user interface is used to interact with the database. The MongoDB driver helps to make the connection between the user interface and the MongoDB server. The MongoDB Shell is the command-line interface that is used to interact with the MongoDB server.

Database, Collection, and Document

- Data is stored as a set of field-value pairs in documents.
- A group of documents together make up a collection.
- Multiple collections make up a database.



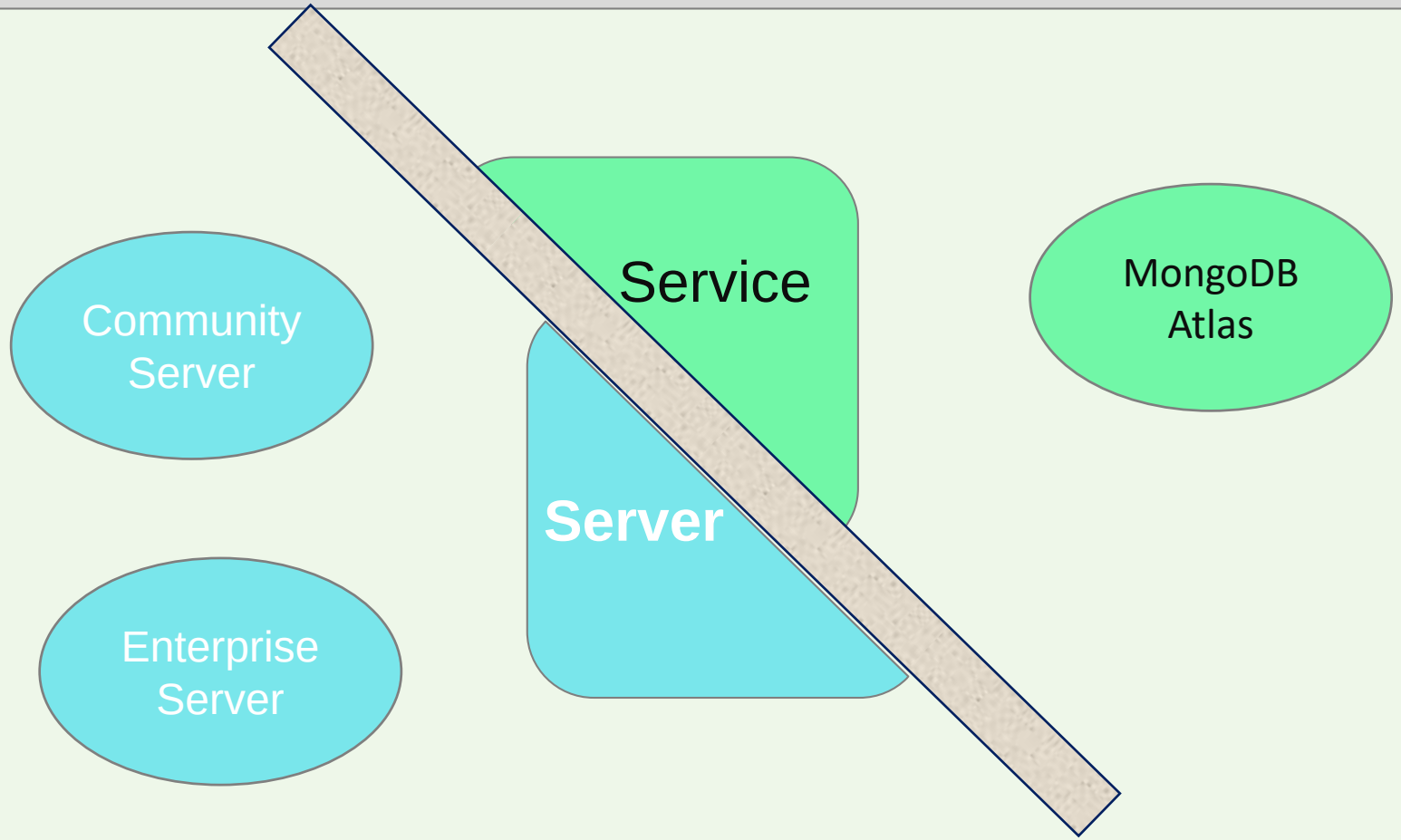
MongoDB stores data in the document in BSON format.

SYNTAX

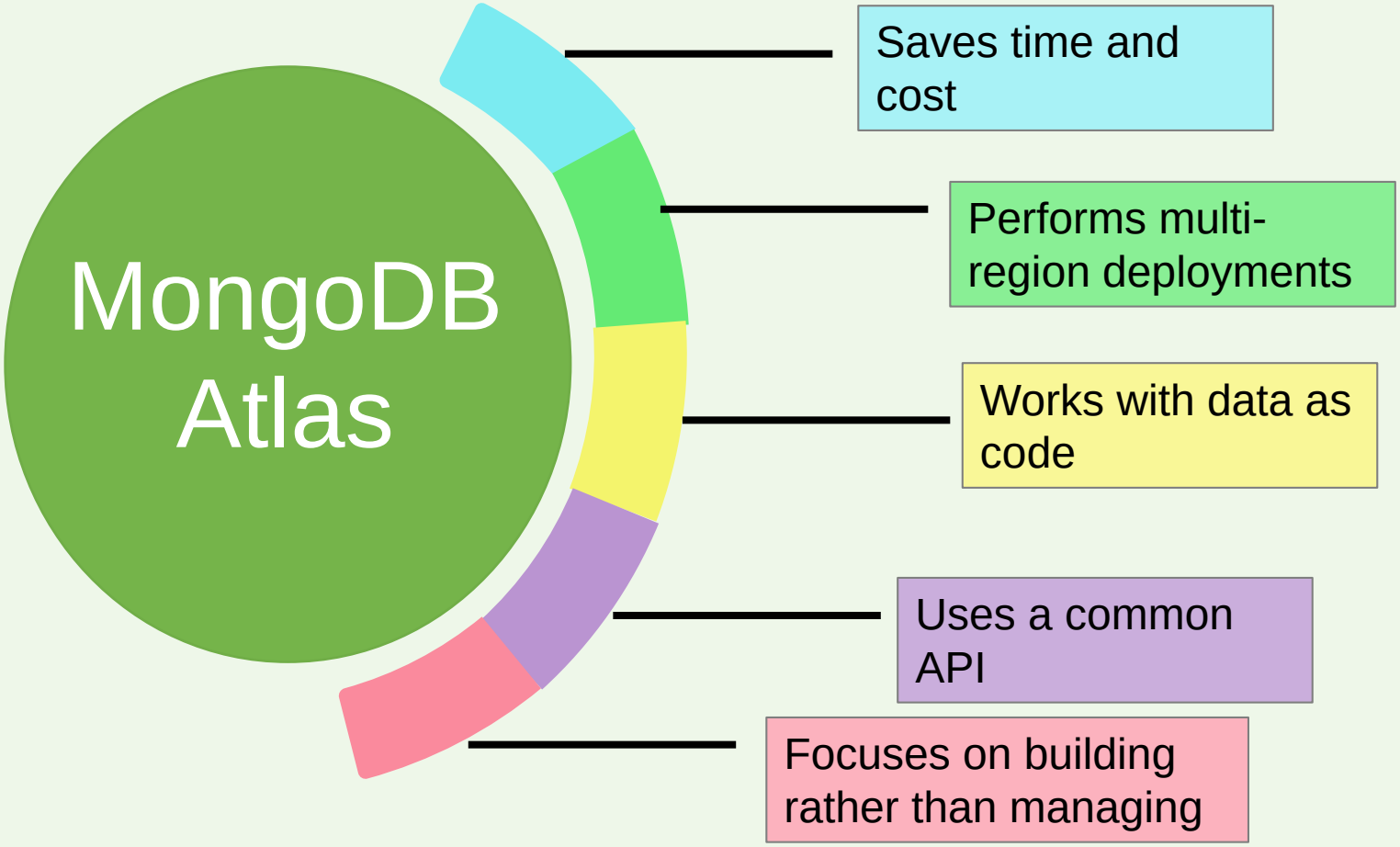
```
{
    field1: value1,
    field2: value2,
    field3: value3,
    ...
    fieldN: valueN
}
```

MongoDB Editions

MongoDB is available both as a server and as a service on the cloud.



MongoDB Atlas



Installing MongoDB Community Server

To install the MongoDB 6.0 Community edition on Windows systems:

1. Open a browser window and navigate to the URL: https://www.mongodb.com/try/download/community

2. The MongoDB Community Server Download page opens.

3. On this page, scroll down and ensure that:
   - In the **Version** drop-down, **6.0.5 (current)** is selected.
   - In the **Platform** drop-down, **Windows** is selected.
   - In the **Package** drop-down, **msi** is selected.

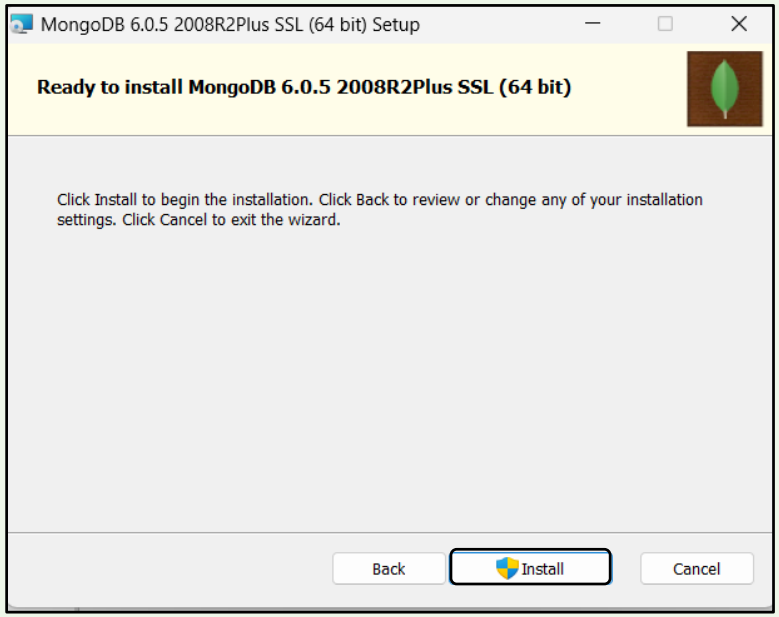
4. Click **Download**. After the download is complete, run the installer.

5. Click **Next**. Select the **I accept the terms in the License Agreement** checkbox to proceed.

6. Click **Next**, **Complete and Next**.

7. Ensure that the **Install MongoDB Compass** checkbox is selected.

8. Click **Next**, **Install, and Finish**.



Installing MongoDB Shell

To install the MongoDB Shell:

1. Open a browser window and navigate to the URL: https://www.mongodb.com/try/download/shell

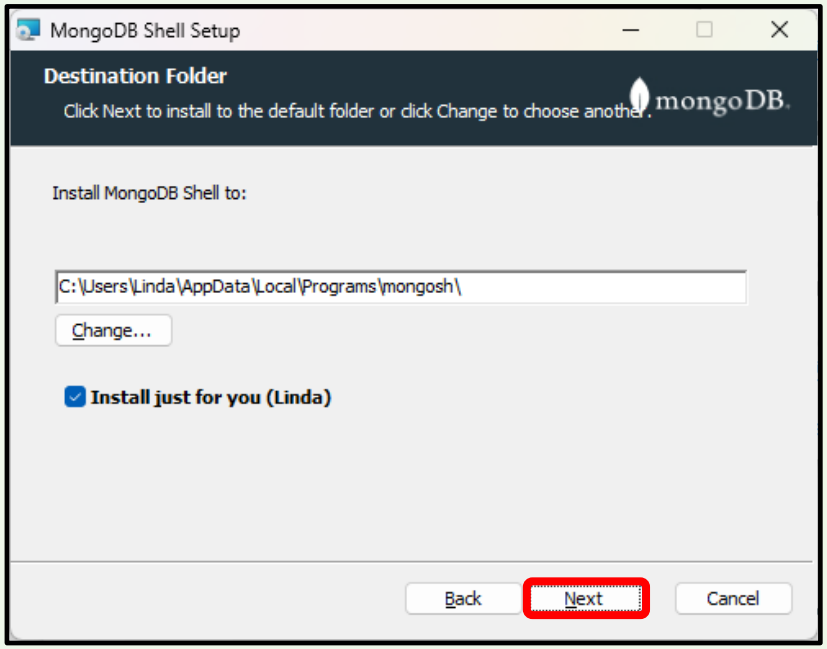
2. The MongoDB Shell Download page opens.

3. On this page, scroll down and ensure that:
   - In the **Version** drop-down, **1.8.2** is selected.
   - In the **Platform** drop-down, **Windows 64-bit (8.1+) (MSI)** is selected.
   - In the **Package** drop-down, **msi** is selected.

4. Click **Download**. After the download is complete, run the installer.

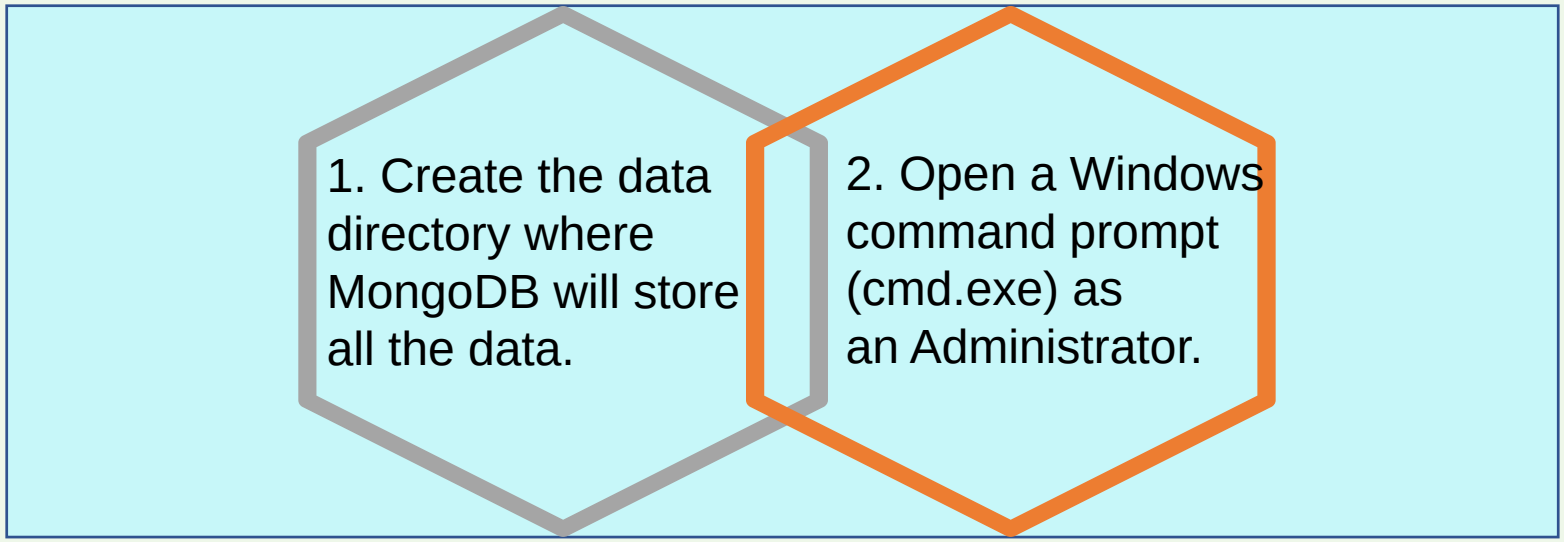
5. The MongoDB Shell Setup wizard opens and click **Next**

6. Check the folder path and click **Next**, then click **Install** and click **Finish**



Setting up the Environment

To set up the environment:



To start the MongoDB database, run the command:

```
"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe" --
dbpath="c:\data\db"
```

Connecting MongoDB Server and MongoDB Shell

To connect the MongoDB Shell with MongoDB Server:

01 Open another Windows command prompt.

02 To connect the MongoDB Shell to a MongoDB deployment running on **localhost** with the default port 27017, run the command: `mongosh`

→ MongoDB Shell gets connected to the MongoDB server.

03 To check the connection, run the command: `show dbs`.

→ The list of existing databases is displayed.

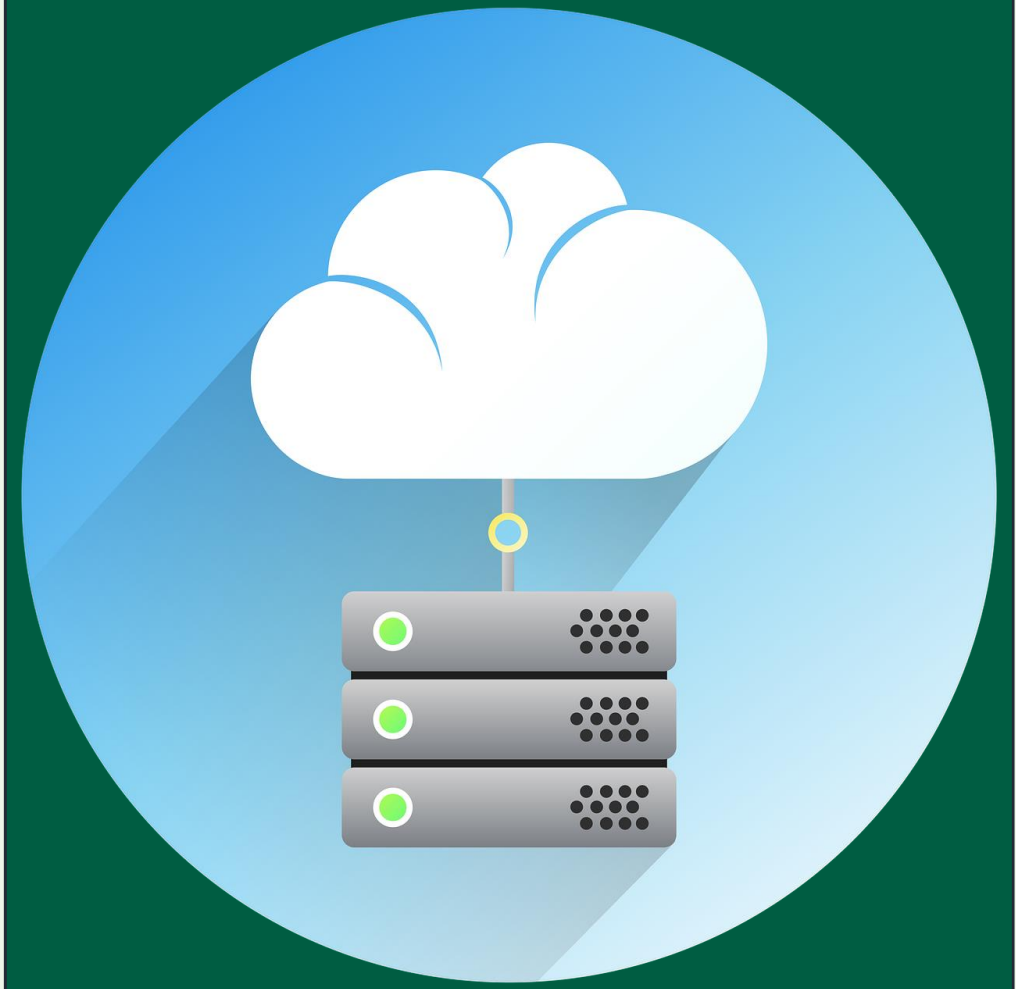
Summary

- A NoSQL database is a non-relational data that can support huge volumes of unstructured data.
- MongoDB is an open-source NoSQL database that stores data in the BSON format.
- The drivers, shell, and storage engine are vital elements for MongoDB to function.
- MongoDB stores data in the form of databases, collections, and documents.
- MongoDB works in two layers: application layer and data layer.
- Mongo DB Compass is the GUI for interacting with the MongoDB Server.
- MongoDB Atlas is the product MongoDB offers to compute on the cloud.
- To set up the MongoDB environment, first install the MongoDB Server, then install the MongoDB Shell, and then use `mongosh` to connect the server to the shell.

Session: 2

Working with MongoDB Database

Managing Large Datasets Using MongoDB



Objectives

- Explain the installation of MongoDB Database Tools and set the environment variable for MongoDB Database Tools
- Explain the method to load a sample dataset into the MongoDB server
- Describe different datatypes in MongoDB
- Explain the methods to create a database and collection
- Describe the ways to insert, query, update, and delete documents, collections, and databases

MongoDB Database Tools

- mongodump
- mongofiles
- mongorestore
- bsondump
- mongoexport
- mongostat
- mongoimport
- mongotop

Installing MongoDB Database Tools

1. Open the browser window and navigate to the Website: https://www.mongodb.com/try/download/database-tools

2. In this page, scroll down:
   - In the **Version** drop-down, ensure that **100.7.0** is selected.
   - In the **Platform** drop-down, ensure that **Windows x86_64** is selected.
   - In the **Package** drop-down, select the **msi** option.

3. Click **Download**. Once the download is complete, run the installer**.**

4. Once the **MongoDB Tools 100 Setup** wizard opens, click **Next**.

5. Select **I accept the terms in the License Agreement** check box.

6. Click **Next.**
   The **Custom Setup** page opens.

7. Click **Next**. Click **Install** when **MongoDB Tools 100 page** opens.

8. Click **Finish** when **Completed the MongoDB Tools 100 Setup Wizard** opens.

Setting up the Environment Variables

After MongoDB Database Tools are installed, set the environment variables for the executable file.

1. Open **Control Panel** in Windows.
2. In the **Control Panel window**, select **System and Security**.
3. In the **System and Security** window that opens, click **System**.
4. In the **System** > **About** window that opens, select **Advanced system settings**. The **System Properties** window opens.
5. In the **System Properties** window, select **Environment Variables**.
6. In the **Environment Variables** window that opens**,** under the **System variables** section, select **Path** and click **Edit**.
7. In the **Edit environment variable** window that opens, click **New**, and then click **Browse** to navigate to the folder where the database tools are installed.
8. Click **OK** in all the windows opened in Steps 5 through 7.

The environment variable is now set for MongoDB tools.

Load Sample Dataset into MongoDB Server

To load a sample dataset into MongoDB Server:

1	Copy the sample dataset named `Sample_dataset` to the C drive.
2	Open the command prompt.
3	To run the mongod instance, execute the command: `"C:\Program Files\MongoDB\Server\6.0\bin\mongod.exe" --dbpath="c:\data\db"`
4	Open a new command prompt and type the command as: `mongoimport --db sample_weatherdata --collection data --jsonArray --file C:\Sample_dataset\sample_weatherdata\data.json`

The sample dataset is now imported into MongoDB server.

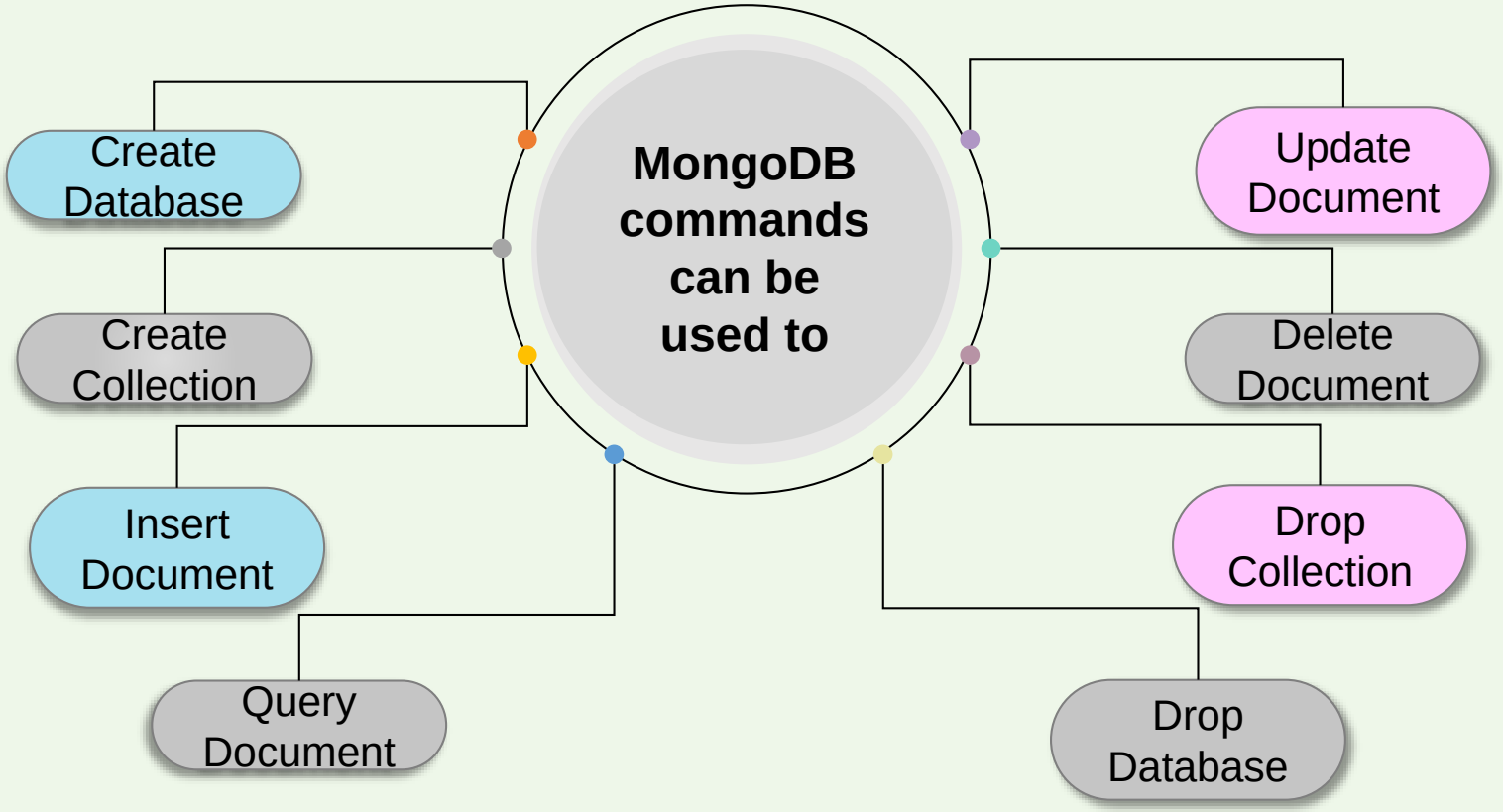
Datatypes in MongoDB

String	Integer	Double	Decimal	Boolean
A String is used to store text.	An Integer is used to store numeric values.	A Double is used to store numeric values that contain 8-byte floating point values.	Decimal is used to store 128-bit floating point decimal values.	Boolean is used to store true or false as values for the fields.

Null	Array	Object	Object ID	Date
Null is used to store NULL value.	Array is a collection of multiple values that are of same type or different types.	Object is used to store embedded documents.	Object ID is a unique identifier for each document in a collection.	Date is used to store date as a 64-bit signed integer (UTC DateTime format).

Working with Databases and Collections

MongoDB offers a set of commands that facilitates the user to work with database and collections.



Create Database and Collections

Create Database

```
Use DATABASE_NAME
```

To create a database named `Student_detail`:

```
Use Student_detail
```

Create Collection

```
db.createCollection
(name, options)
```

To create a collection named `Studentinfo` in the `Student_detail` database:

```
db.createCollection
("Studentinfo")
```

Insert Document

Insert a Single Document → `db.collection.insertOne()`

To insert a document into the `Studentname` collection →

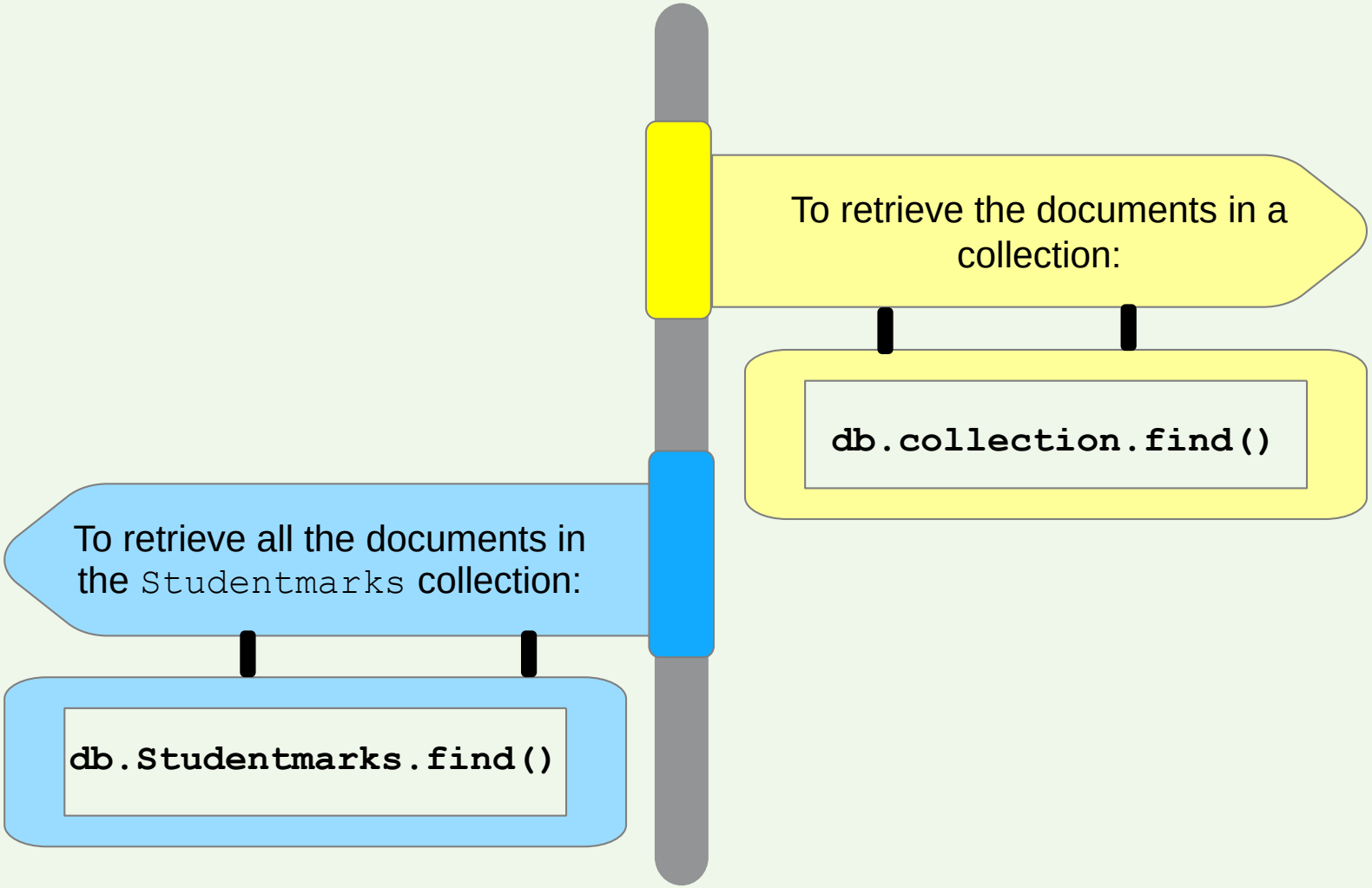
```
db.createCollection("Studentname")
db.Studentname.insertOne({"Name":
"Richard"})
```

Insert Multiple Documents → `db.collection.insertMany()`

To insert multiple documents into the `Studentname` collection →

```
db.Studentmarks.insertMany([
{ Name: "Robert", Age: 16, Subject1: 89, Subject2: 78},
{ Name: "Oliver", Age: 17, Subject1: 78, Subject2: 85},
{ Name: "Henry", Age: 15, Subject1: 90, Subject2: 93} ])
```

Query a Document



Update Document

MongoDB provides two different methods to update documents in a collection.

Update a Single Document

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

To update the value of `name` field as '`David`' where the value in the `name` field is '`Richard`'.

```
db.Studentmarks.updateOne
({"Name":"Robert"},
{$set:{"Name":"David"}})
```

Update Multiple Documents

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

To update and set the value of '`Subject2`' as `99` in all documents where the value of '`Subject1`' is greater than `80`.

```
db.Studentmarks.updateMany
({"Subject1":{$gt: 80}},
{$set:{"Subject2":99}})
```

Delete Document

MongoDB provides two different methods to delete documents in a collection.

Delete a Single Document

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

To delete the document where the value of the `name` field is '`Oliver`':

```
db.Studentmarks.deleteOne ({"Name":"Oliver"})
```

Delete Multiple Documents

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

To delete all the documents where the value in the '`Age`' field is greater than `14`:

```
db.Studentmarks.deleteMany ({"Age": {$gt:14}})
```

Drop Collection and Drop Database

Drop Collection

To remove a collection from the database:	`db.collection.drop()`
To remove the `Studentmarks` collection from the database:	`db.Studentmarks.drop()`

Drop Database

To drop a database:	`db.dropDatabase()`

Summary

- MongoDB provides a set of tools that helps users to manage the databases in the MongoDB environment.
- To start using the commands in MongoDB, install MongoDB Database Tools and set the environment variable for it.
- A sample dataset can be loaded into the MongoDB server using the `mongoimport` tool.
- MongoDB stores data in BSON format. It supports various datatypes such as `string`, `integer`, `double`, `Boolean`, `null`, `array`, `object`, `object ID`, and `date`.
- MongoDB provides different commands and methods to create and delete databases and collections. It also provides methods to insert, query, update, and delete documents and collections.

Session: 3

MONGODB OPERATORS

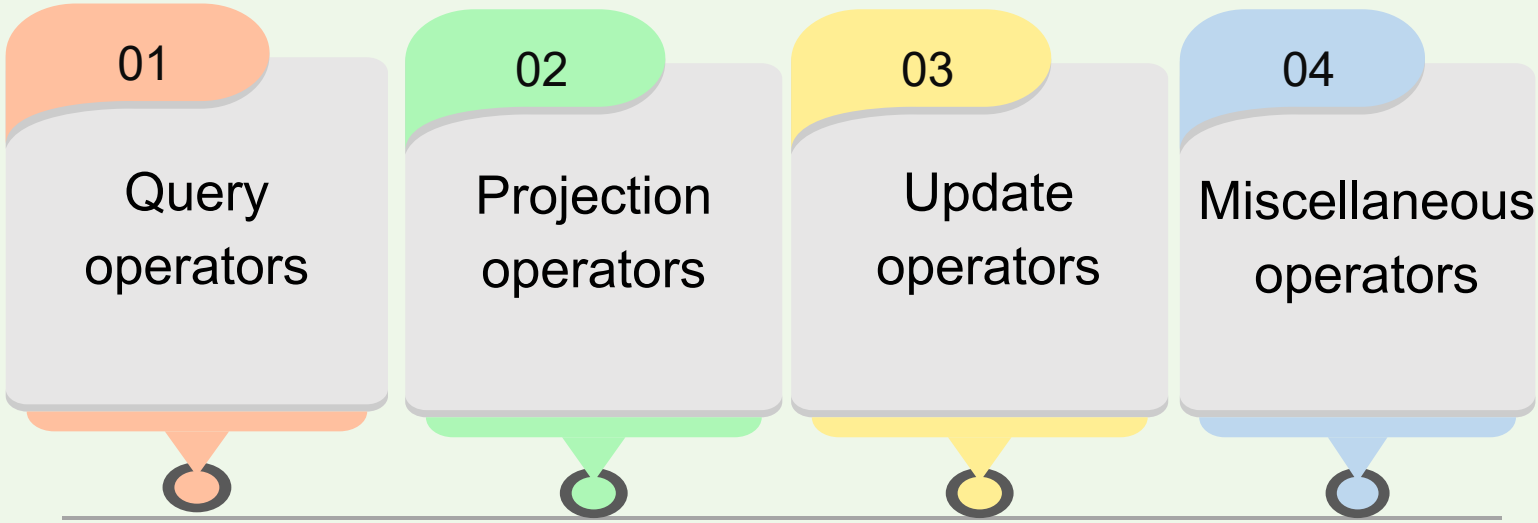
Managing Large Datasets Using MongoDB

Objectives

- Describe the types of MongoDB operators
- Explain how to use the query and projection operators
- Explain how to modify field and array data using update operators

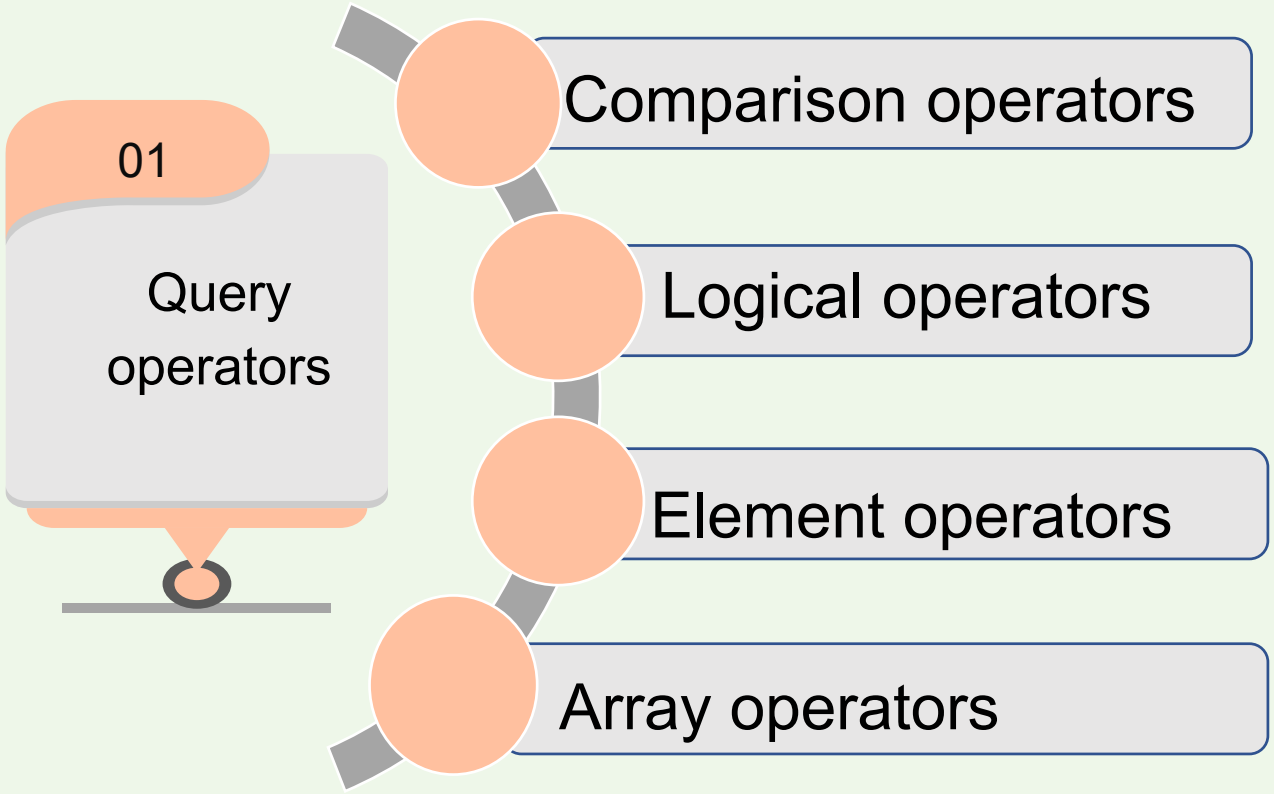
What is an Operator?

- Reserved words or symbols that instruct the compiler or interpreter to perform specific mathematical or logical operations on the dataset as required
- Operators in MongoDB are widely classified as:



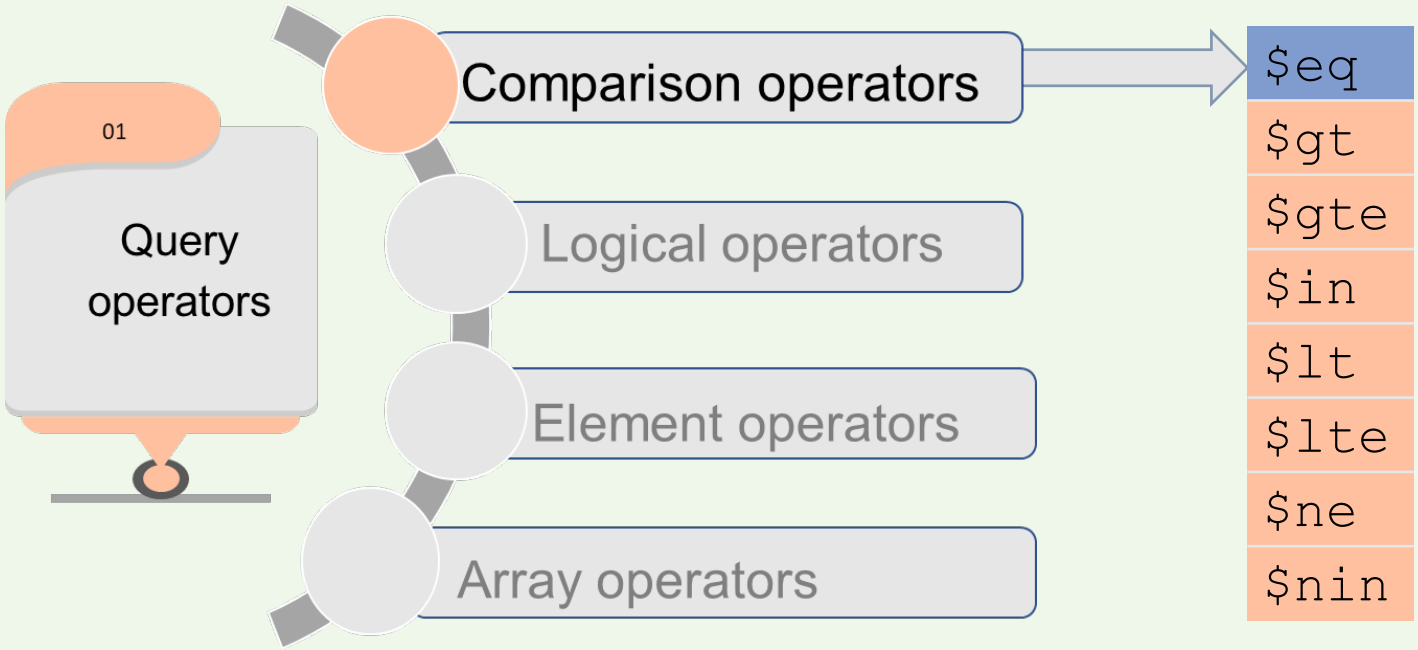
Query Operators

Query operators help in retrieving data from a database. There are four types of query operators:



Comparison Operators

Comparison operators compare the field's value with the specified value and return the documents that match the value. There are eight comparison operators:

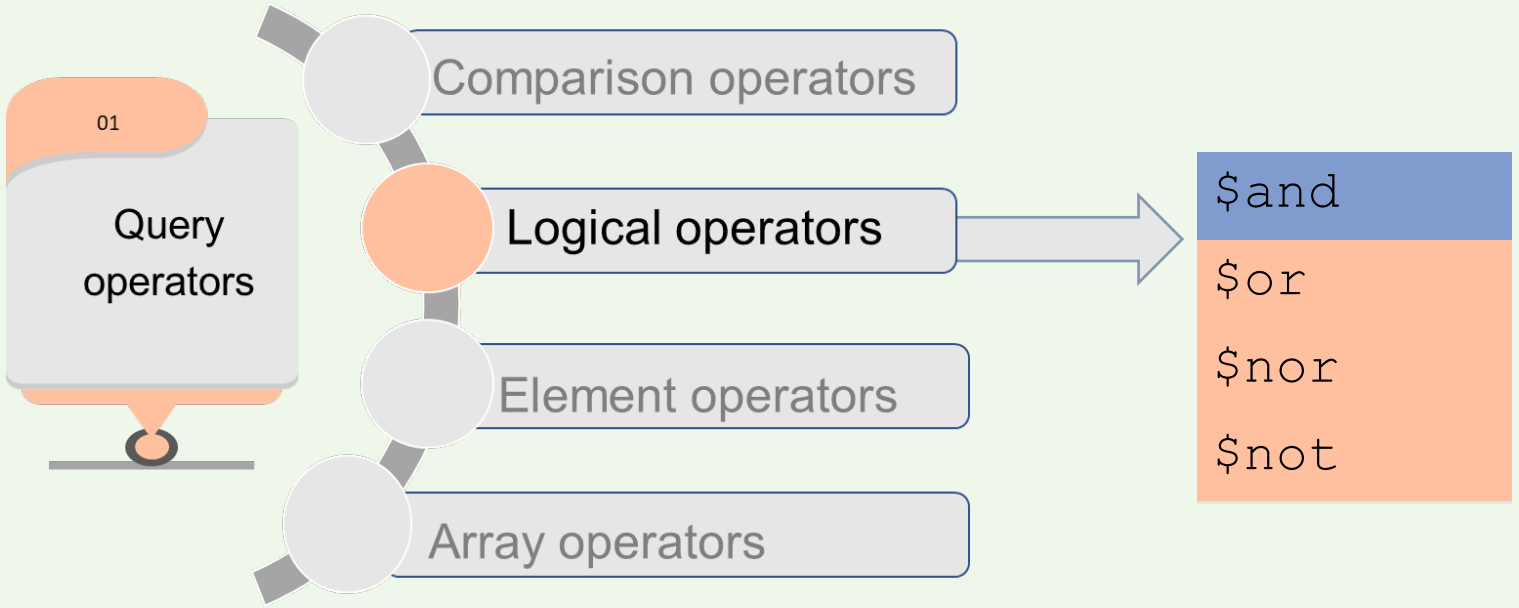


The `$eq` operator matches documents where the field value is equal to a specified value.

```
{ <field>: { $eq: <value> }}
```

Logical Operators

Logical operators join multiple expressions and return documents based on the operator used.

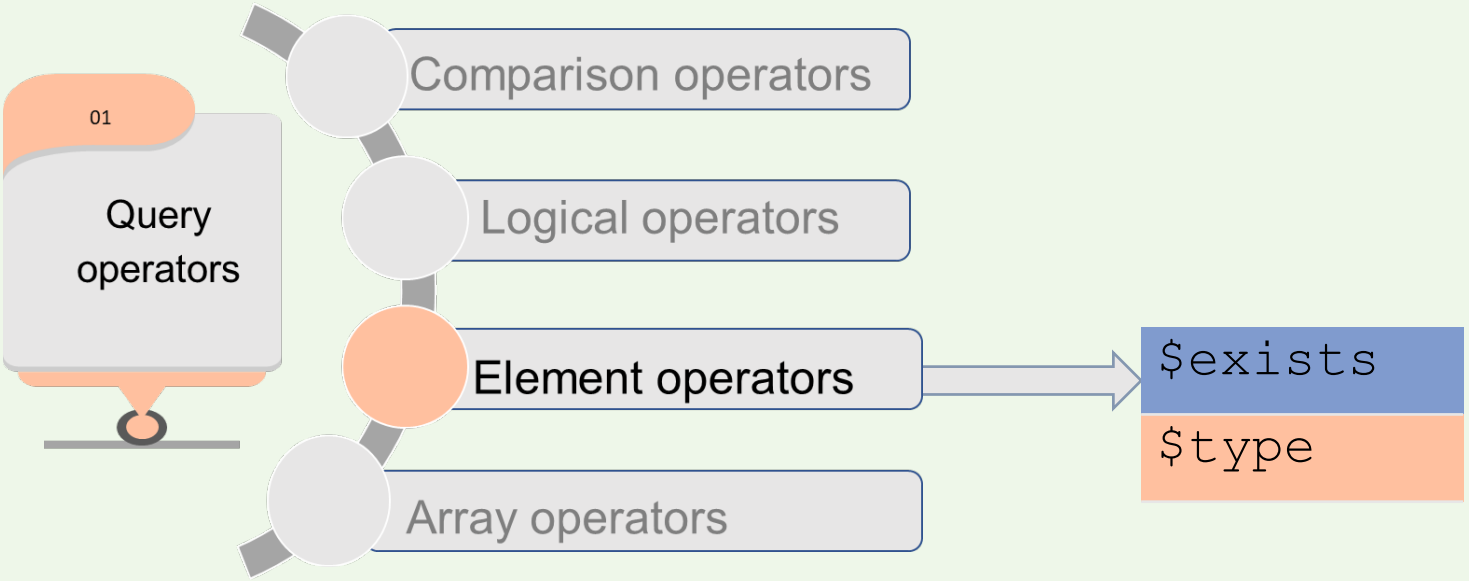


The `$and` operator lists documents that match all the expressions in the specified array.

```
{ $and: [ { <expression1> }, { <expression2> } , ... , {
                    <expressionN> } ] }
```

Element Operators

Element operators check if a particular field is available in a MongoDB document or if a particular field with a specified type is available in the document. There are two element operators:



The `$exists` operator checks if the specified field is present in a document.

```
{ field: { $exists: <boolean> } }
```

Array Operators

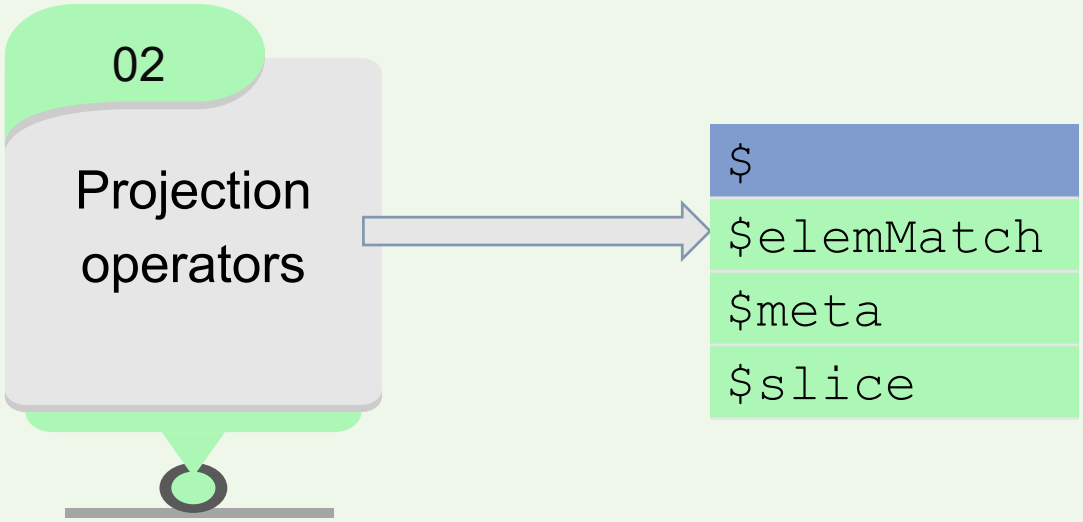
Array operators returns documents based on the conditions specified for array fields. There are three array operators:

The `$all` operator returns documents that have an array field will all the elements specified in the query.

```
{ <field>: { $all: [ <value1> , <value2> ... ] } }
```

Projection Operators

Projection operators returns the fields or elements in the query result based on the condition specified in the query. There are four projection operators:

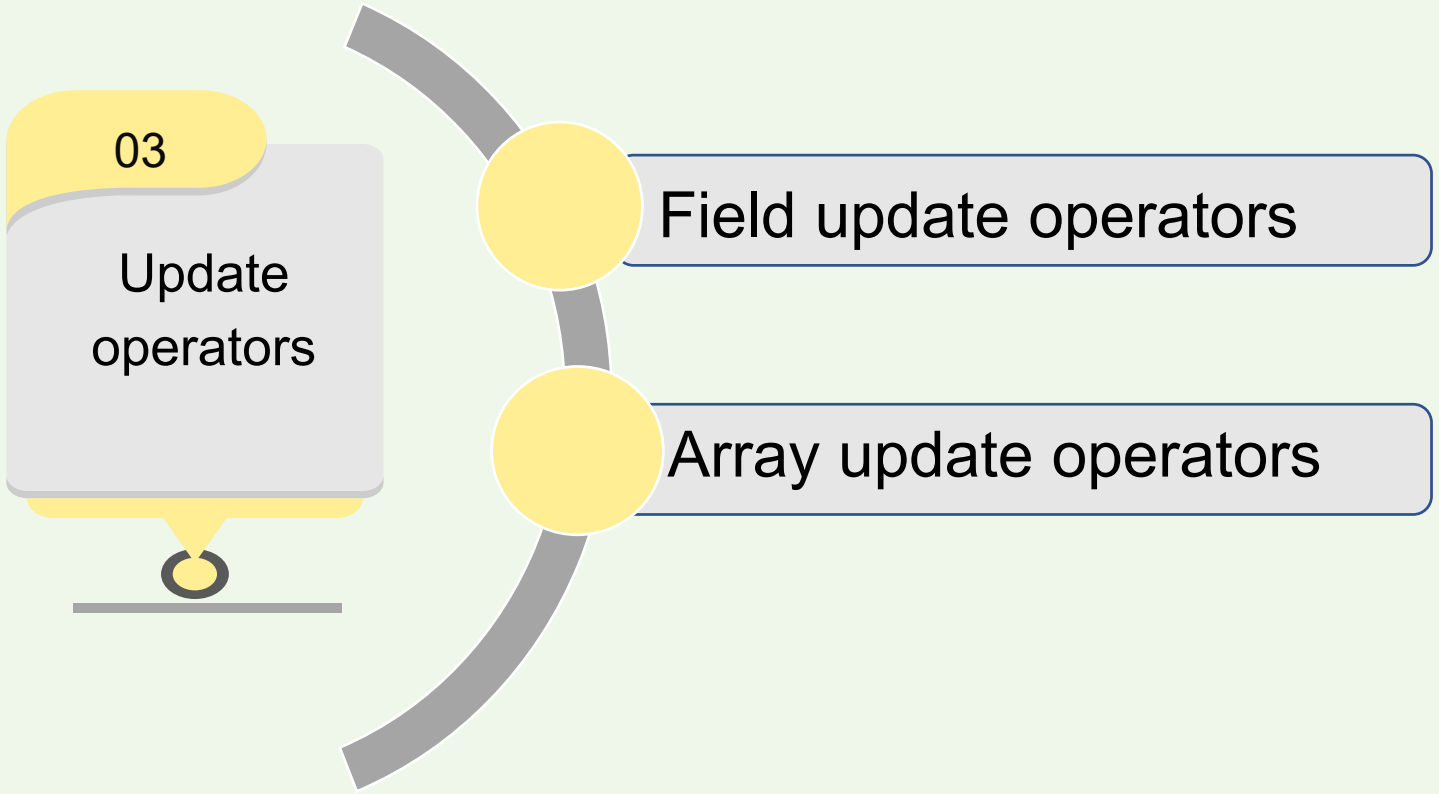


The `$` operator returns the first element in an array that matches the condition specified in the query or the first element in the array, if no query condition is specified.

```
db.collection.find( { <array>: <condition> ... },
                    { "<array>.$": 1 } )
```

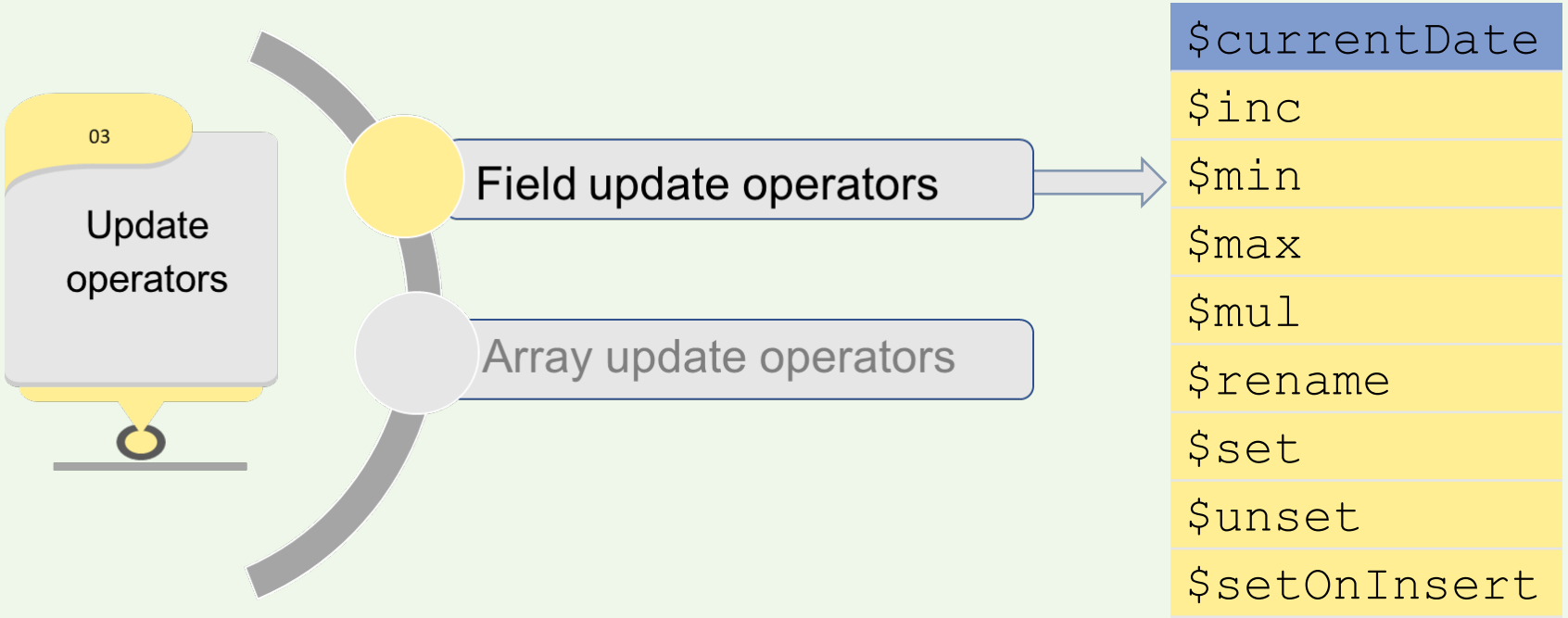
Update Operators

Update operators help users modify documents in a collection. There are two types of update operators:



Field Update Operators

MongoDB offers nine different operators to modify data in the fields. They are:

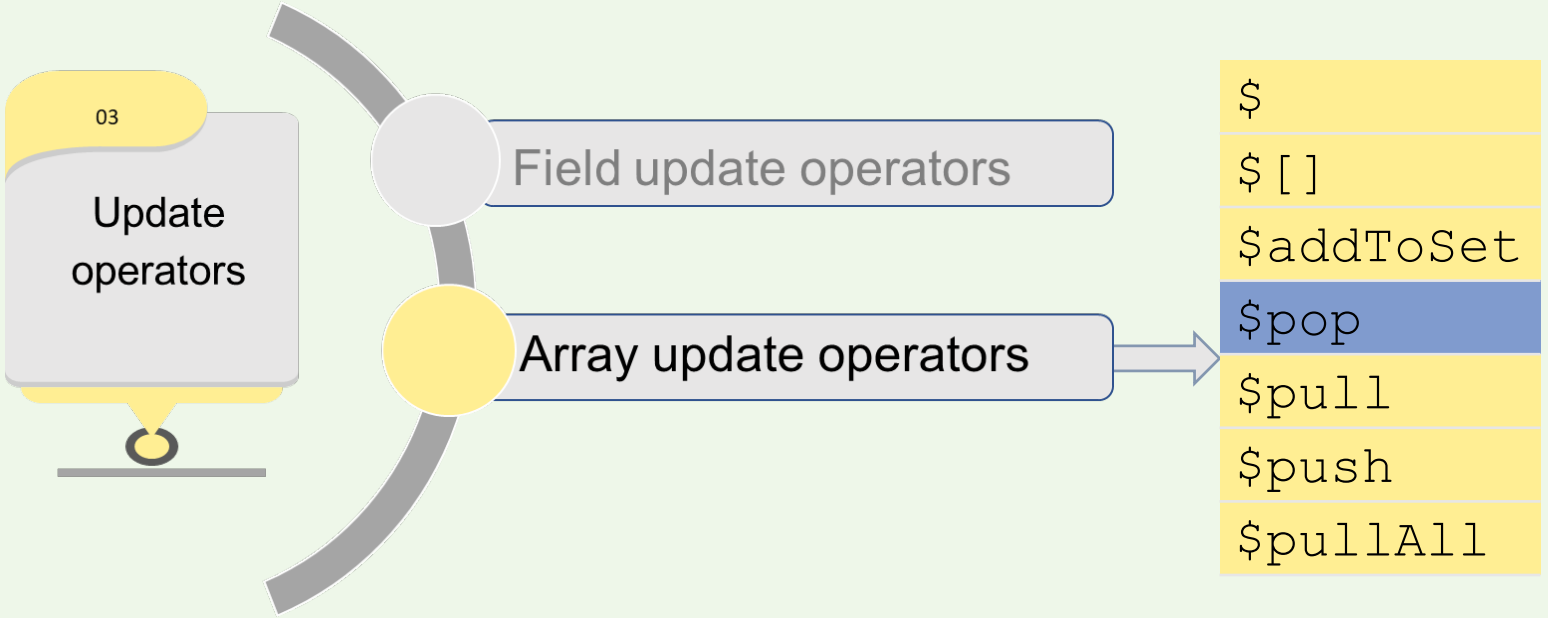


The `$currentDate` operator is used when the value in a field must be set to the current date.

```
{ $currentDate: { <field1>: <typeSpecification1>, … } }
```

Array Update Operators

MongoDB offers seven different operators to modify data in an array. They are:

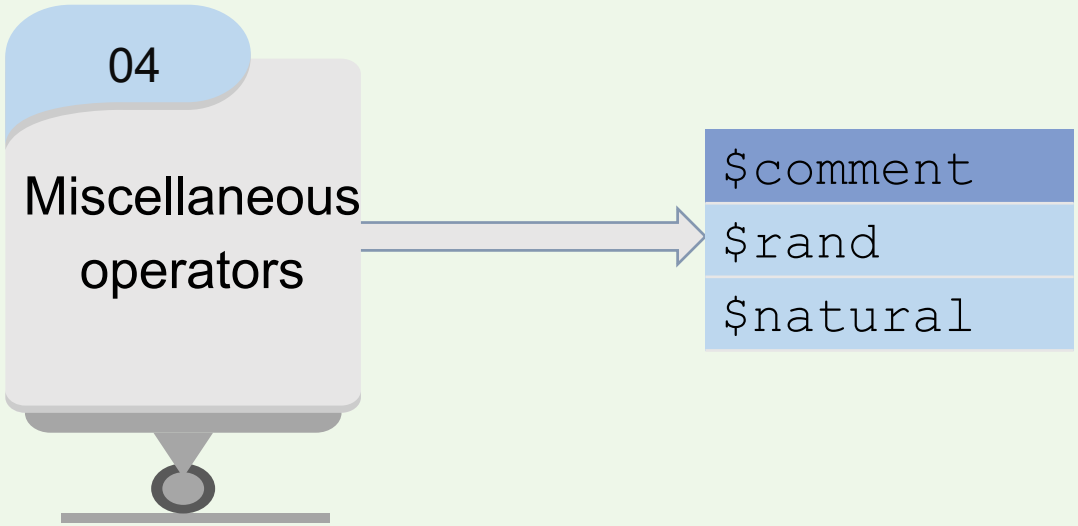


The `$` operator is used when the first or last item has to be removed from an array. Syntax:

```
{ $pop: { <field>: <-1 | 1>, ... } }
```

Miscellaneous Operators

Apart from the operators discussed, MongoDB offers other operators for interacting with the databases:



The `$comment` operator adds comments to a query predicate to explain what the query is intended to do.

```
db.collection.find( { <query>, $comment: <comment> } )
```

Summary

- MongoDB offers multiple operators to allow users to create complex queries.
- Main types of operators in MongoDB are query, projection, and update operators.
- Query operators allow operations that are based on comparison (both field and array), logic, and existence of a field.
- Projection operators help to specify what fields of a document a query must return.
- Update operators allow modification of both the field and array data.

Session: 4

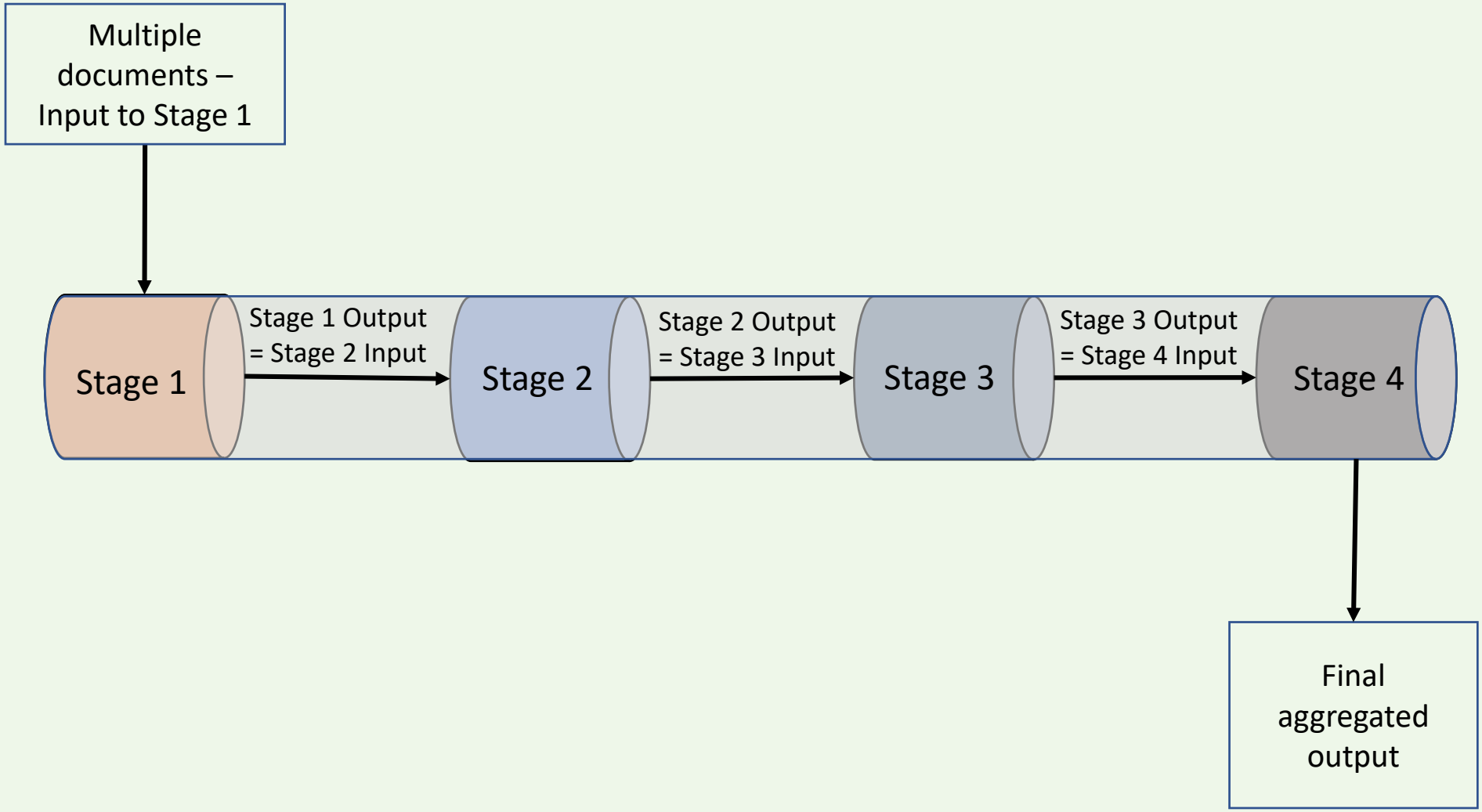
Aggregation Pipeline

Managing Large Datasets Using MongoDB

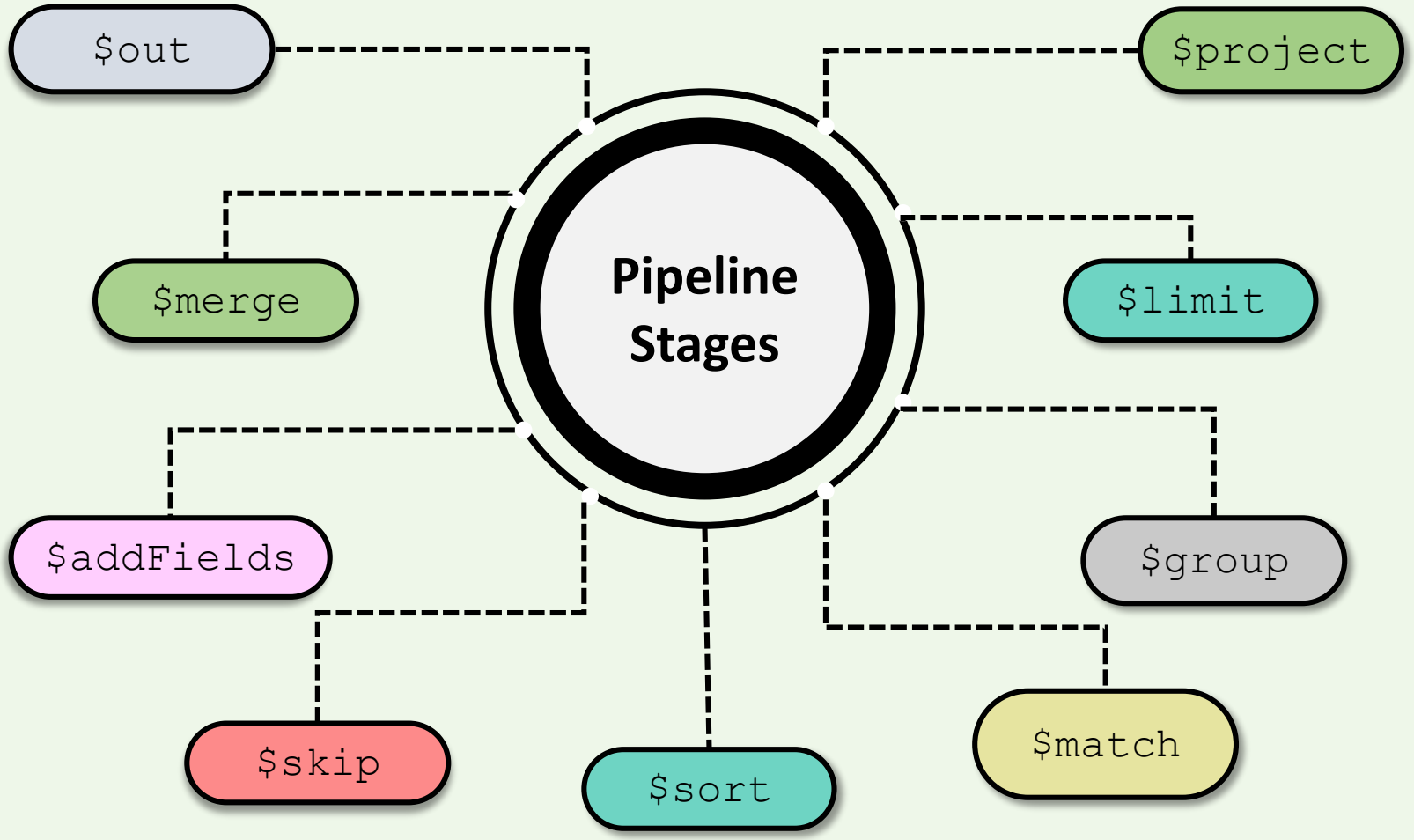
Objectives

- Explain aggregation pipeline in MongoDB
- Describe the stages in the aggregation pipeline
- Explain the expressions that can be used in the aggregation pipeline

Aggregation Pipeline Stages



Pipeline Stages



$project Stage

The $project stage takes in documents and does manipulations such as including new fields, excluding existing fields, updating the values before sending to it the next stage.

```
{$project: {<specification(s)>}}
```

```
db.inspections.aggregate( [ { $project : {certificate_number : 1,
                business_name : 1, address:1 } } ] )
```

$limit Stage

The $limit stage helps in limiting the number of documents that are passed on to the next stage.

```
{$limit: {<64-bit positive integer>}}
```

```
db.inspections.aggregate( [{ $limit : 2 },
{ $project : {certificate_number : 1, business_name : 1,
address:1 }}] )
```

$group Stage

The $group stage helps in grouping documents according to a field, key, or group of fields.

```
{$group: { _id: <expression>, //Group Key
<field1>: {<accumulator1>: <expression1>}, … }}
```

```
db.inspections.aggregate( [{$limit:5000 }, { $group :
{ _id : "$address.city", count: { $count: { }} } }] )
```

<accumulator1>

$avg	$bottom	$bottomN	$top	$topN	$min	$minN
$max	$mergeObjects	$first	$last	$count	$sum	

$match Stage

The $match stage helps in filtering documents based on conditions and sends the results to the next step.

```
{$match:{<expression>}}… }}
```

```
db.inspections.aggregate([{$project:{_id:0,"address.city":
1, business_name: 1}}, {$match:{"address.city": "JERSEY
CITY"}}, {$limit:10}]))
```

$sort Stage

The $sort stage uses the supplied field(s) to sort the documents in either ascending or descending order and sends it to the next stage.

```
{$sort:{field1:<sort order>,
field2:<sort order>,…}}
```

```
db.inspections.aggregate([{$project:
      {_id:0,"address.city":1,
business_name: 1}}, {$limit: 5000},
    {$sort:{"address.city": -1,
"business_name": 1}}, {$limit:10}])
```

`$skip` Stage

The `$skip` stage removes some of the documents that are passed to the next stage.

```
{$skip:{<64-bit positive integer}}
```

```
db.inspections.aggregate([{
$project:{_id:0, "address.city":1,business_name:1}},
                    {$limit: 5000},
  {$sort:{"address.city": -1, "business_name": 1}},
                    {$skip:10}])
```

$addFields Stage

The $addFields stage adds new fields to the documents passed on to the next stage.

```
{$addFields:{<new field1>:<expression1>….}}
```

```
db.inspections.aggregate ([{$addFields: {"address.country":
  "USA"}},{$limit:5}, {$project:{_id:0,business_name: 1,
      "address.city": 1, "address.country": 1}}])
```

$out Stage

The $out stage copies the query results to a collection.

```
{ $out: { db: "<output-db>", coll:
    "<output-collection>" } }
```

```
db.inspections.aggregate( [{$limit:8},{
 $project : { certificate_number : 1 ,
business_name : 1, addresaddress:1 } },
   { $out : "out_inspection1" } ] )
```

$merge Stage

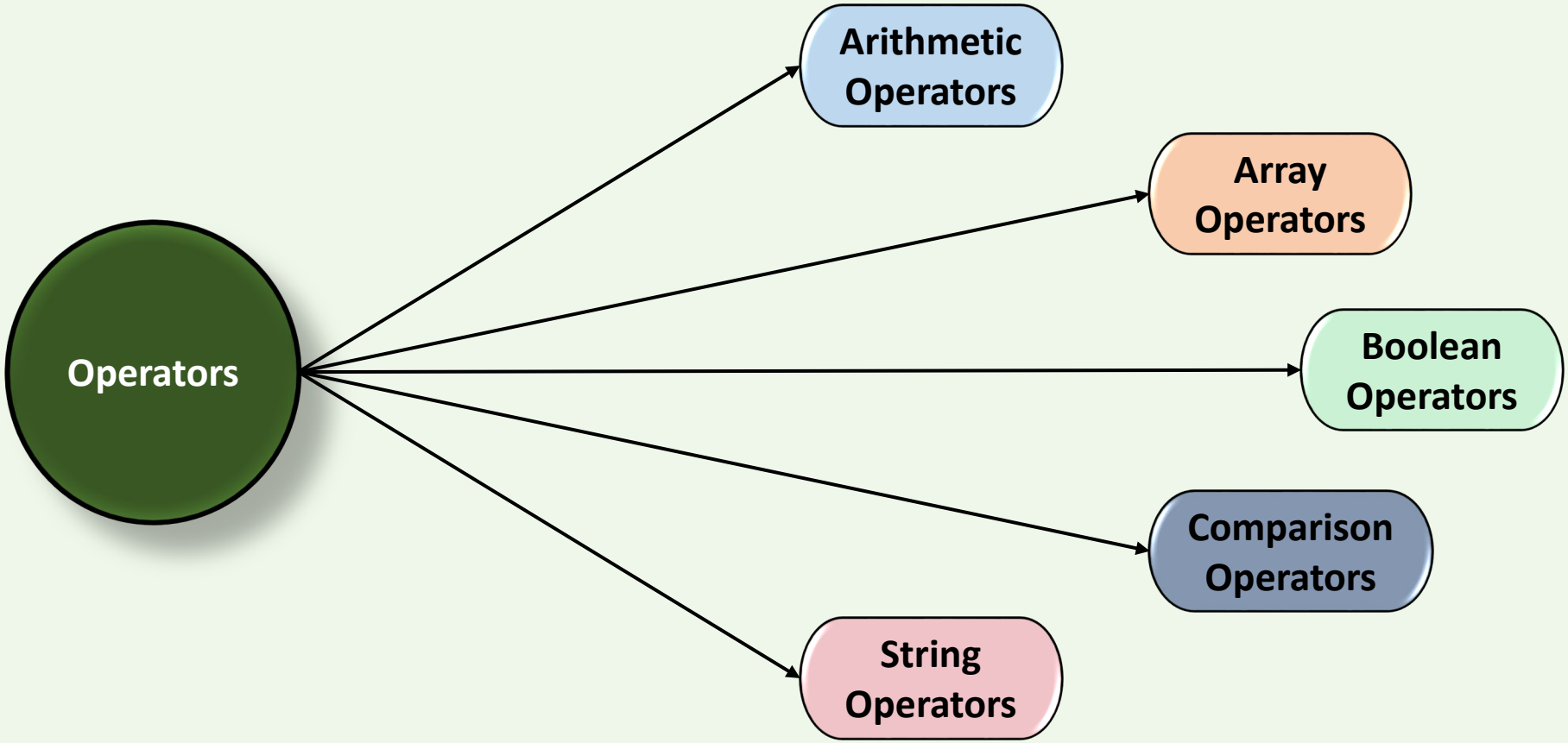
The $merge stage saves the output of the pipeline to a new collection in the same database or a separate database.

```
{ $merge: {into: <collection> -or- { db: <db>, coll: <collection> },
        on: <identifier field> -or- [ <identifier field1>, ...],
let: <variables>, whenMatched: replace|keepExisting|merge|fail|pipeline>,
                whenNotMatched: <insert|discard|fail> } }
```

```
db.inspections.aggregate( [{$skip:10}, { $match: {
  "address.city": "NEW YORK" }} ,{ $project : {
certificate_number : 1 , business_name : 1, address:1 }
    },{$limit:3} { $out : "out_inspection" } ] )
```

Aggregation Pipeline Operators

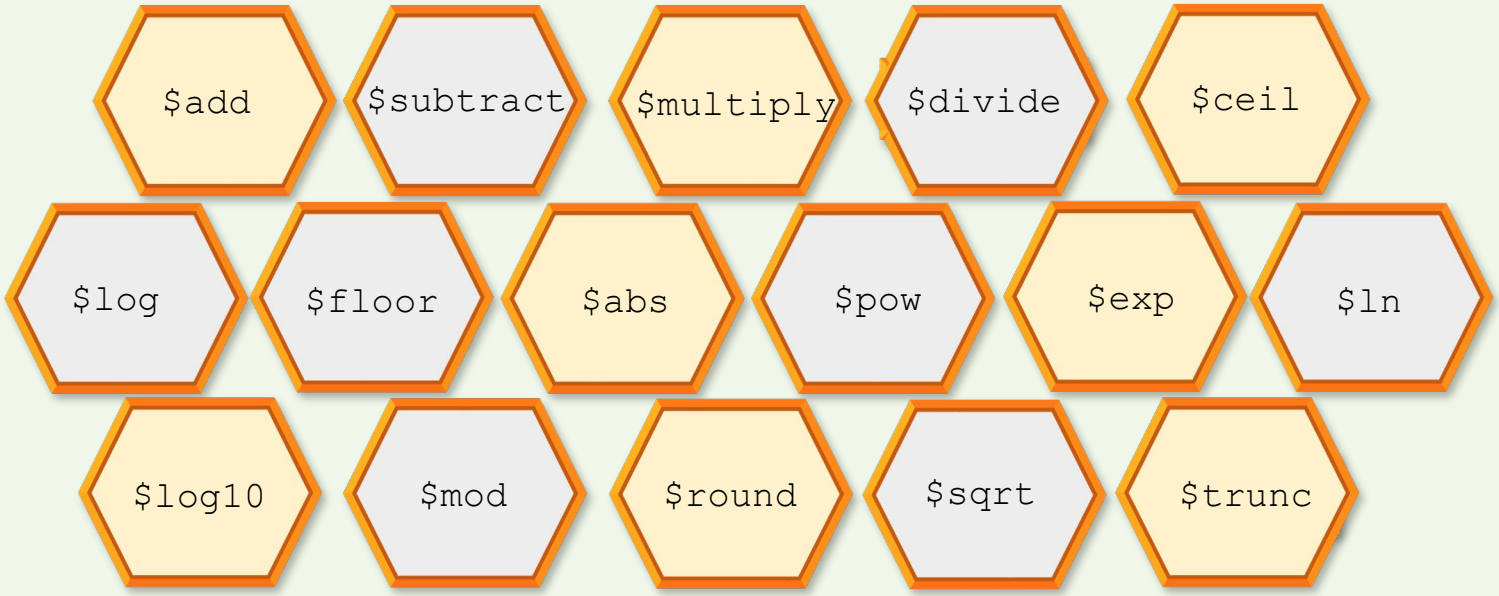
Aggregation pipeline operators are used in various stages of the aggregation pipeline to perform arithmetic or logical calculations.



Arithmetic Operators

Arithmetic operators perform arithmetic operations on the supplied operands and return a single value as result.

Arithmetic operators:




```
$add:{<expression1>,
  <expression2>,…}
```

```
db.accounts.aggregate([ { $project:
{ account_id: 1, limit: 1, newLimit:
{$add:["$limit", 2000]} } }])
```

Array Operators

Array operators are used to perform operations on arrays.

$first	$last	$sortArray	$size	$map
$firstN	$lastN	$maxN	$slice	$range
	$isArray	$minN	$arrayToObject	$reduce
	$concatArray	$arrayElemAt	$objecttoArray	$reverseArray
	$filter	$indexofArray	$in	$zip

```
$first/$last:<expression>
```

```
db.grades.aggregate([{ $addFields:
{exam_score: { $first: "$scores" } } },
{$project:{student_id:1,class_id:1,
exam_score:1}}])
```

Boolean Operators

Boolean operators take parameters in the form of an expression and resolve them to boolean values.

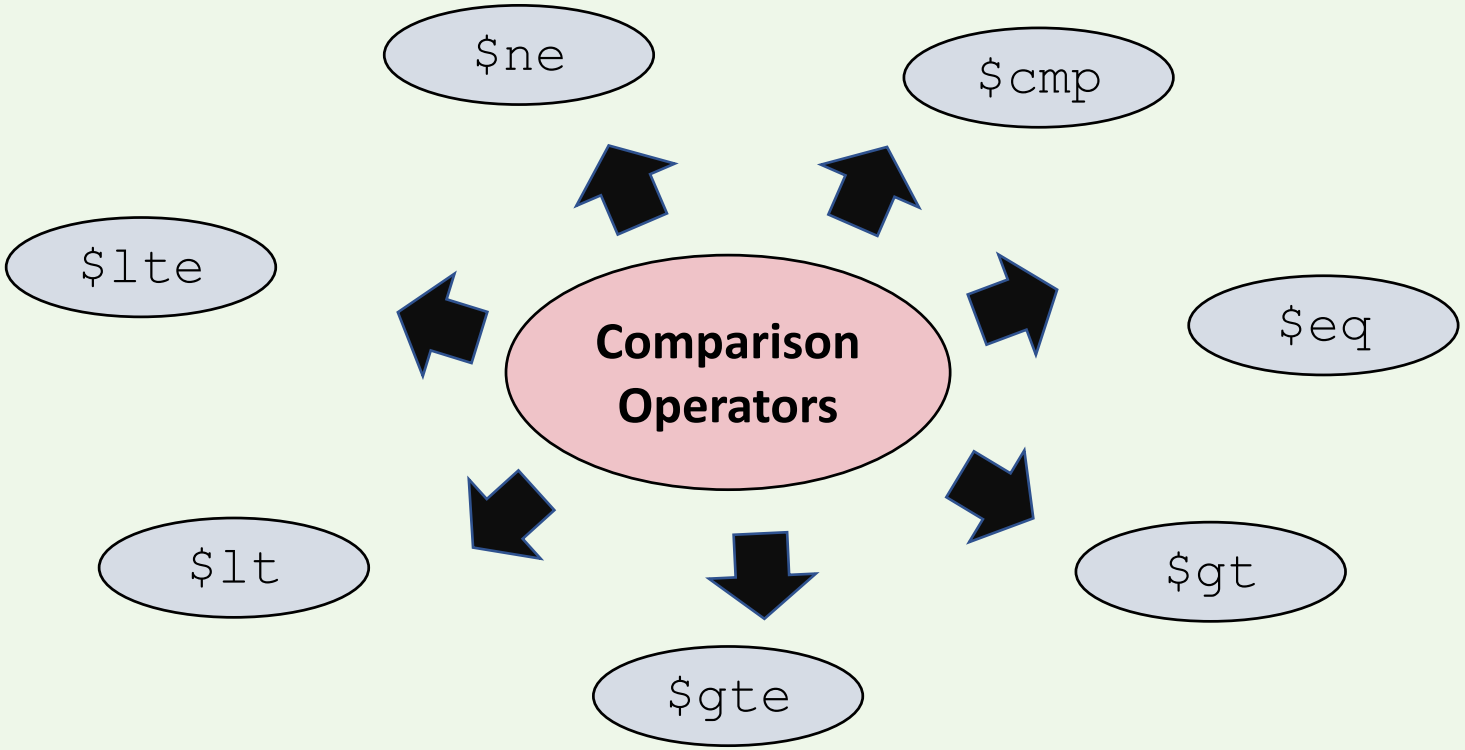
Boolean Operators:

`$and`	`$not`	`$or`
Returns true if all the specified expressions resolve to true; else, it returns false.	Returns true if the specified expression resolves to false and it returns false if the specified expression resolves to true.	Returns true if any one of the specified expressions evaluates to true.

```
{ $and: [ <expression1>, <expression2>, ... ] }
```

Comparison Operators

Comparison operators compare two expressions and return a Boolean value.




```
{ $gt/$lt: [ <expression1>, <expression2> ] }
```

String Operators

String operators work on strings.

$concat	$ltrim	$replaceAll	$strcasecmp
$dateFromString	$regexFind	$rtrim	$toLower
$datetoString	$regexFindAll	$split	$trim
$indexofBytes	$regexMatch	$strLenBytes	$toUpper
$indexofCP	$replaceOne	$strLenCP	$toString

SYNTAX

```
{ $concat: [ <expression1>, <expression2>, ... ] }
```

Summary

- Aggregation is the processing of many documents in a collection and giving out a result.
- Aggregation pipeline is a series of steps, where some processing is done at each step and the result is passed to the next stage.
- Some of the stages of the aggregation pipeline include `$project`, `$limit`, `$match`, `$skip`, `$group`, `$sort`, `$addFields`, `$out`, and `$merge`.
- Arithmetic operators, Boolean operators, string operators, and comparison operators are some of the operators in MongoDB that are used to manipulate data in the aggregation pipeline.

Session: 5

Database Commands

Managing Large Datasets Using MongoDB

Objectives

- Explain the importance of database commands in MongoDB
- Describe the various types of database commands with examples

Running a Command

Run a command against the current database using following syntax:

```
db.runCommand( { <command> } )
```

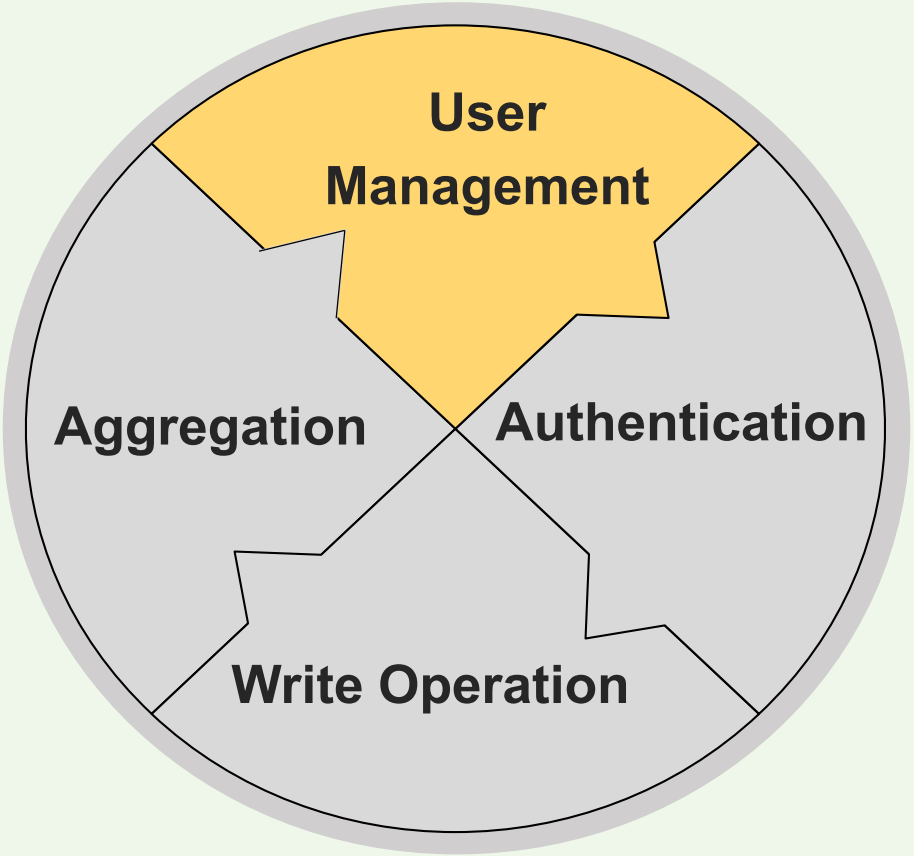
The `db.runCommand` command takes a database command as its parameter.

Introduction to MongoDB Commands

MongoDB offers database commands in the following categories:

User Management Commands

User management commands deal with operations that manage users.



`createUser`
`grantRolesToUser`
`usersinfo`
`dropUser`

createUser Command

To create a user in the currently active database:

```
db.runCommand(
{
    createUser: "<name>",
    pwd: passwordPrompt(),  // Or "<cleartext
    password>"
    customData: { <any information> },
    roles: [{ role: "<role>", db:
    "<database>" } | "<role>", ... ],
    writeConcern: { <write concern> },
    authenticationRestrictions: [
            { clientSource: [ "<IP|CIDR
    range>", ... ],
    serverAddress: [ "<IP|CIDR range>", ... ]
    }, ... ],
    mechanisms: [ "<scram-mechanism>", ... ],
    //Available starting in MongoDB 4.0
    digestPassword: <boolean>, comment: <any>
}
```

grantRolesToUser Command

To grant additional roles to a user:

```
db.runCommand(
{
    grantRolesToUser: "<user>",
    roles: [ <roles> ],
    writeConcern: { <write
    concern> },
    comment: <any>
}
)
```

`usersInfo` Command

To view information about users in the database:

```
db.runCommand(
{
    usersInfo: <various>,
    showCredentials: <Boolean>,
    showCustomData: <Boolean>,
    showPrivileges: <Boolean>,
    showAuthenticationRestrictions:
    <Boolean>,
    filter: <document>,
    comment: <any>
}
)
```

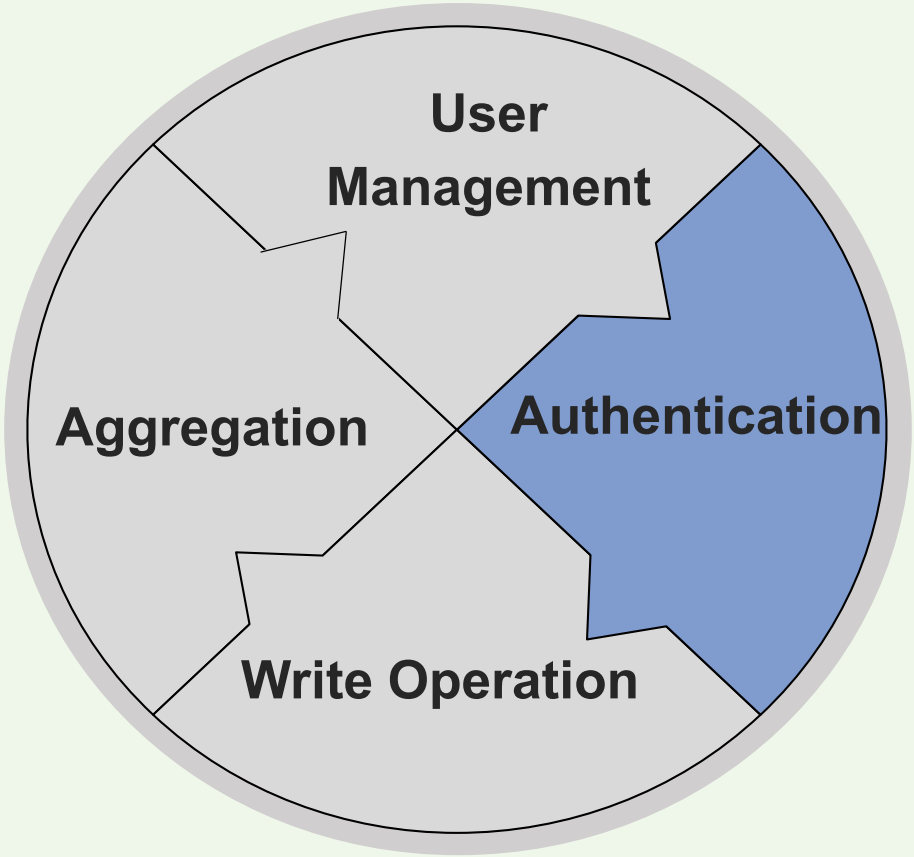
dropUser Command

To remove a user from the current database:

```
db.runCommand(
{
    dropUser: "<user>", writeConcern: { <write concern>
    },
    comment: <any>
})
```

Authentication Commands

Authentication commands use an authentication mechanism to validate users.



auth
getonce

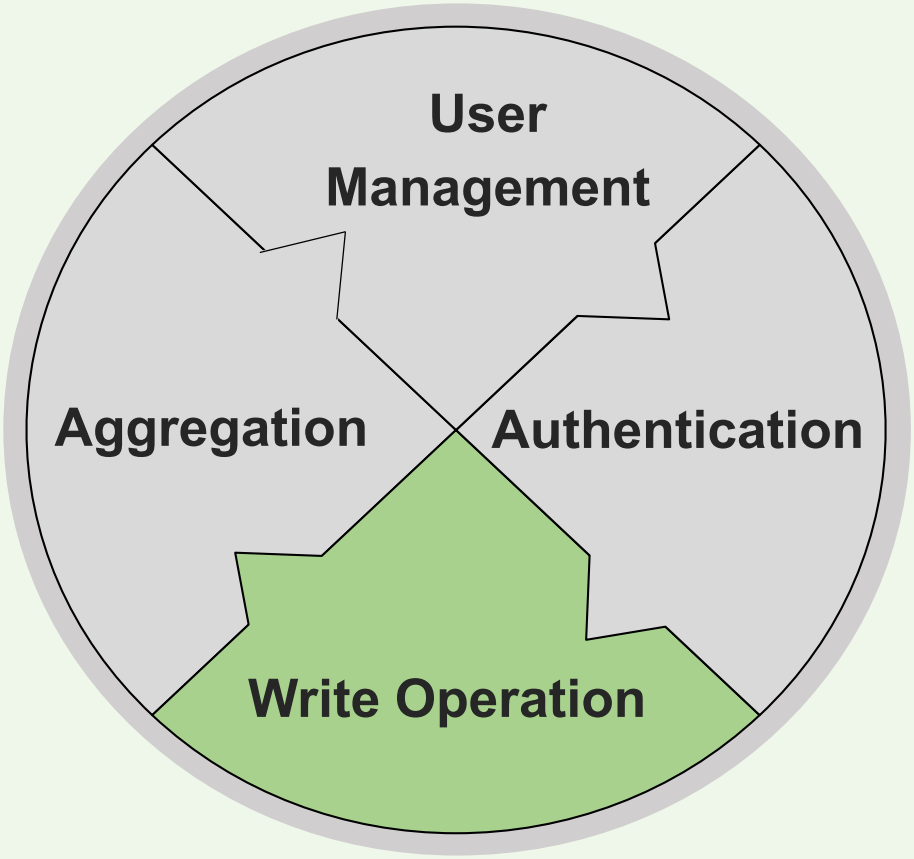
auth Command

To authenticate a user with username and password

```
db.auth
({user: <username>,
pwd: passwordPrompt(),
// Or "<cleartext password>"
mechanism: <authentication mechanism>,
digestPassword: <boolean> })
```

Write Operation Commands

Write Operation commands deal with inserting, updating, and deleting data in the database.



`insert`
`find`
`findAndModify`
`update`
`delete`

`insert` Command

To insert documents into a collection:

```
db.runCommand(
{
    insert: <collection>,
    documents: [ <document>,
    <document>, <document>, ... ],
    ordered: <boolean>,
    writeConcern: { <write
    concern> },
    bypassDocumentValidation:
    <boolean>,
    comment: <any>
}
)
```

`find` Command

To select documents that match your criteria and obtain a cursor to the selected documents:

```
db.runCommand(
{
  find: <string>, filter: <document>,
  sort: <document>, projection: <document>,
  hint: <document or string>,
  skip: <int>, limit: <int>,
  batchSize: <int>, singleBatch: <bool>,
  comment: <any>, maxTimeMS: <int>,
  readConcern: <document>, max: <document>,
  min: <document>, returnKey: <bool>,
  showRecordId: <bool>, tailable: <bool>,
  oplogReplay: <bool>, noCursorTimeout: <bool>,
  awaitData: <bool>, allowPartialResults: <bool>,
  collation: <document>, allowDiskUse : <bool>,
  let: <document> // Added in MongoDB 5.0
})
```

`findAndModify` Command

To search for a document in a collection and modify it:

```
db.runCommand(
{
  findAndModify: <collection-name>,
  query: <document>, sort: <document>,
  remove: <boolean>, update: <document or
  aggregation   pipeline>, new: <boolean>,
  fields: <document>, upsert: <boolean>,
  bypassDocumentValidation: <boolean>,
  writeConcern: <document>, collation:
  <document>,
  arrayFilters: <array>, hint: <document|string>,
  comment: <any>,
  let: <document> // Added in MongoDB 5.0
})
```

update Command

To modify documents in a collection:

```
db.runCommand(
{
  update: <collection>,
  updates: [
  {
   q: <query>,
   u: <document or pipeline>,
   c: <document>, // Added in MongoDB 5.0    upsert: <boolean>,
   multi: <boolean>, collation: <document>, arrayFilters:
   <array>, hint: <document|string> }, ...],
   ordered: <boolean>, writeConcern: { <write concern> },
   bypassDocumentValidation: <boolean>, comment: <any>,
   let: <document> // Added in MongoDB 5.0 } )
```

`delete` Command

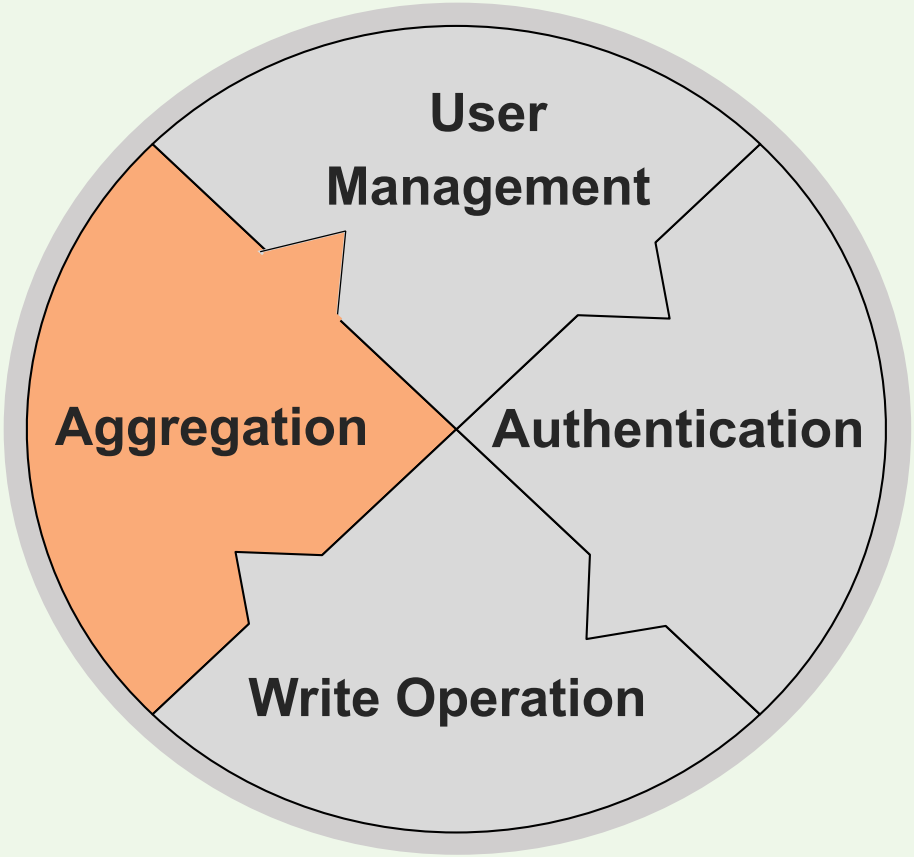
To delete one or more documents from a collection:

```
db.runCommand(
{
 delete: <collection>,
 deletes: [
 {
  q : <query>,
  limit : <integer>, collation: <document>,
  hint: <document|string> }, ...],
  comment: <any>,
   let: <document>, // Added in MongoDB 5.0
   ordered: <boolean>, writeConcern: {
   <write concern> } } )
```

Aggregation Commands

Aggregation commands deal with grouping data and performing operations on the grouped data.

`Count`
`aggregate`
`distinct`



`count` Command

To count the number of documents in a collection:

```
db.runCommand
(
    {
      count: <collection or view>,
      query: <document>,
      limit: <integer>,
      skip: <integer>,
      hint: <hint>,
      readConcern: <document>,
      collation: <document>,
      comment: <any>
    }
)
```

`aggregate` Command

To aggregate using an aggregation pipeline:

```
db.runCommand(
{
  aggregate: "<collection>" || 1,
  pipeline: [ <stage>, <...> ],
  explain: <boolean>,
  allowDiskUse: <boolean>, cursor: <document>,
  maxTimeMS: <int>,
  bypassDocumentValidation: <boolean>,
  readConcern: <document>,
  collation: <document>,
  hint: <string or document>,
  comment: <any>,
  writeConcern: <document>,
  let: <document> // Added in MongoDB 5.0
  }
)
```

distinct Command

To identify the distinct values for a specified field in a single collection:

```
db.runCommand(
{
    distinct: "<collection>",
    key: "<field>",
    query: <query>,
    readConcern: <read concern document>,
    collation: <collation document>,
    comment: <any>
}
)
```

Summary

- Database commands allow users to create and edit databases.
- MongoDB offers database commands for user management, authentication, write operations, and aggregation.
- User management commands include user creation, user deletion, viewing of user information, and granting user roles.
- Authentication happens using the SCRAM mechanism in MongoDB.
- Write operation commands in MongoDB allows insertion of data to a collection, modifying existing data, and deleting data from a collection.
- Aggregation commands perform operations on grouped values from multiple documents to return a single result.

Session: 6

MongoDB Shell Methods

Managing Large Datasets Using MongoDB

Objectives

- 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

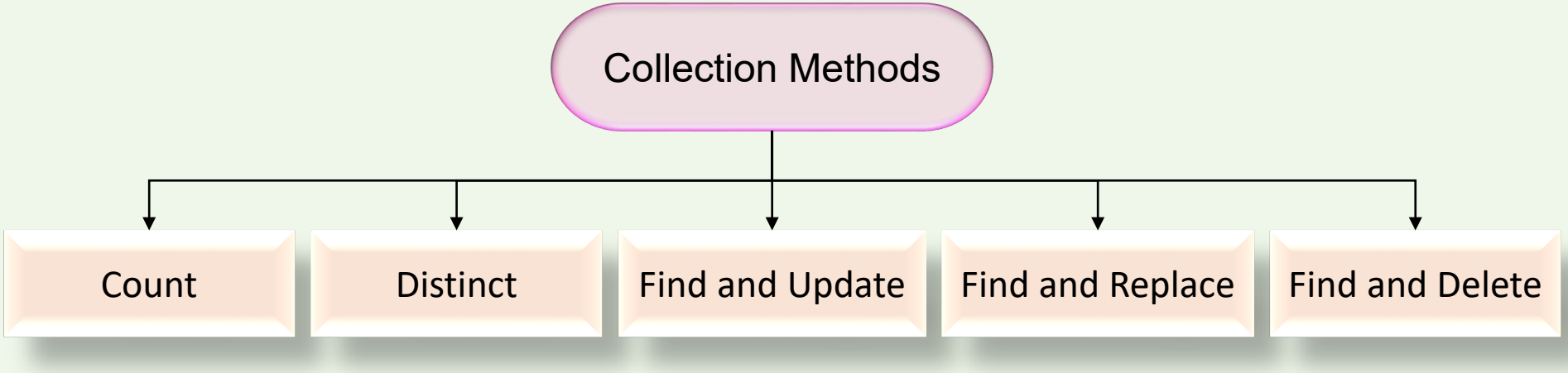
Introduction to MongoDB Shell Methods

MongoDB Shell Methods

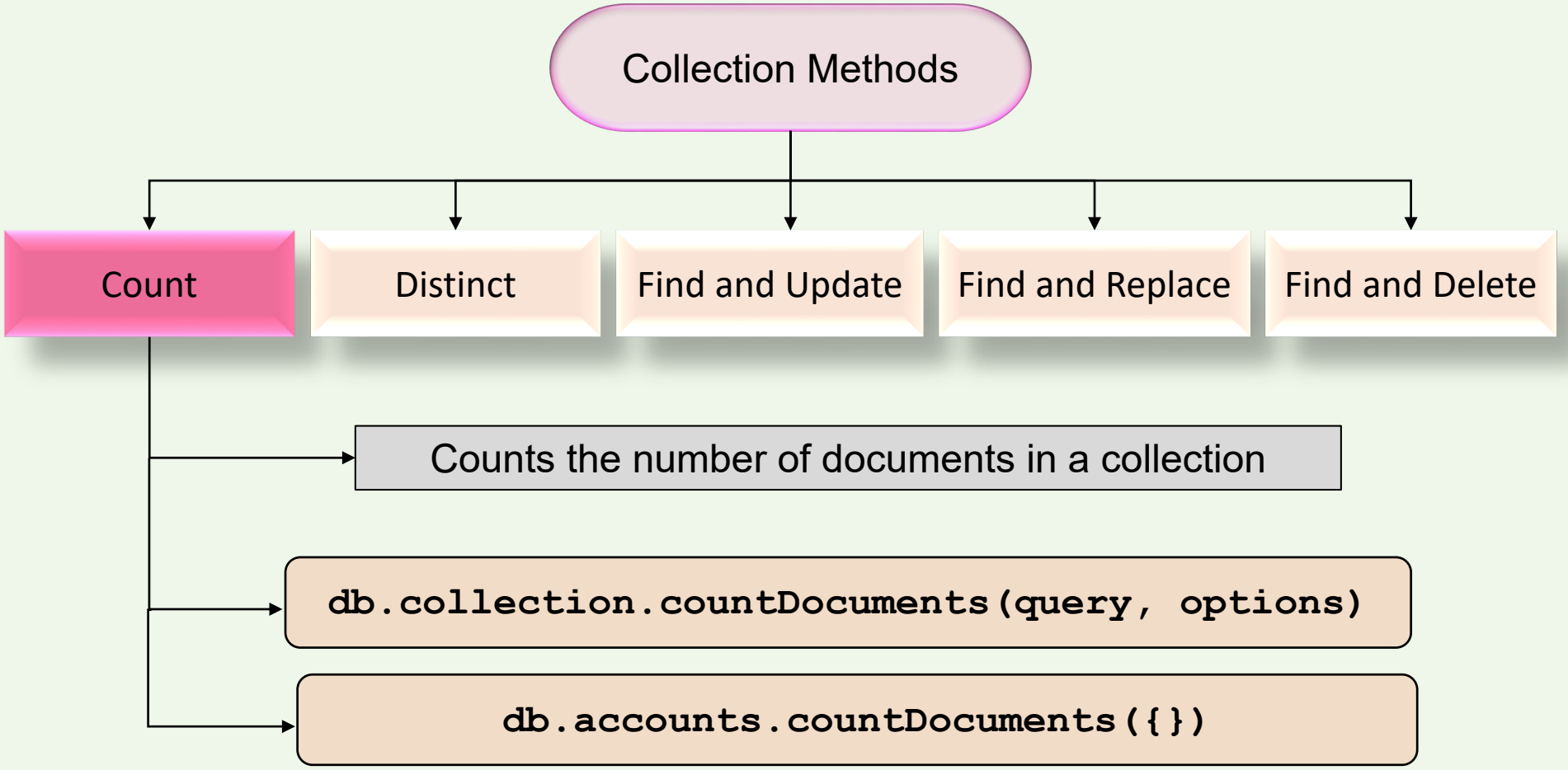
- Create, insert, update, and remove databases, collections, and documents
- Change the way a query is run
- Manage user authentication
- Provide built-in roles and role-based authorization
- Perform operations in bulk
- Replicate data across multiple MongoDB servers
- Distribute data from large datasets across multiple machines

Collection Methods

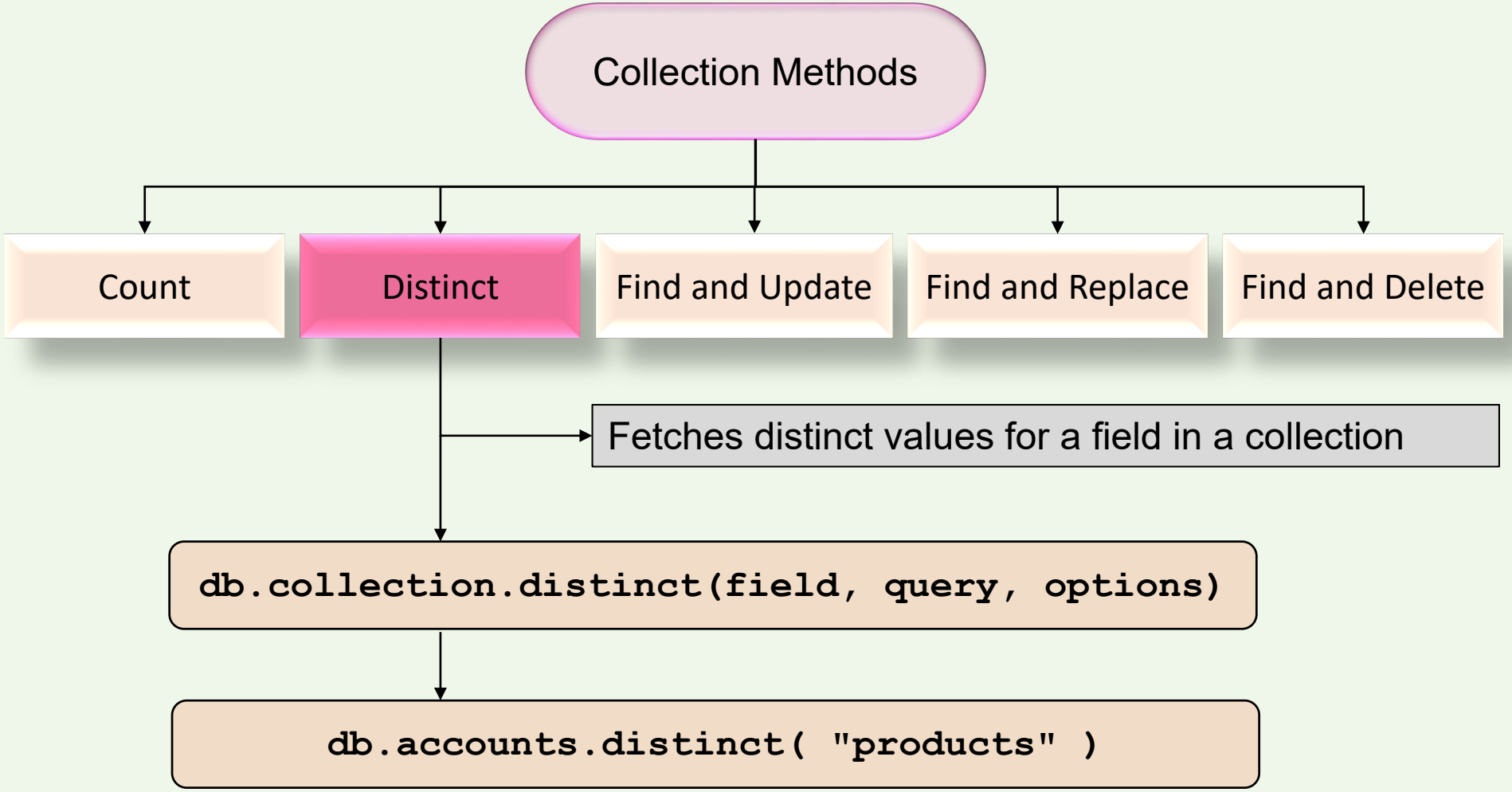
Collection methods help to create, read, update, delete, rename, and manage the collections in a database.



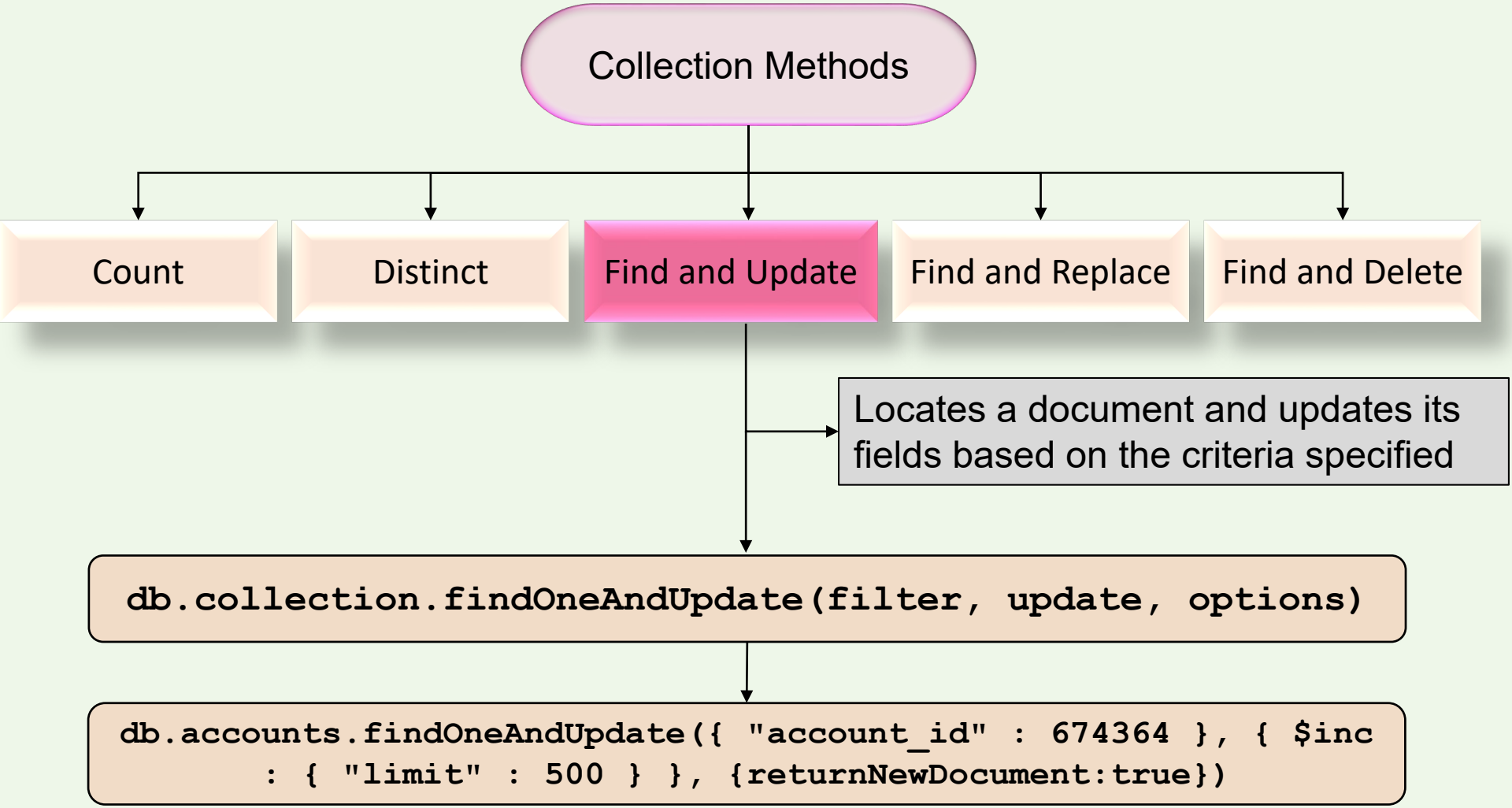
Count Method



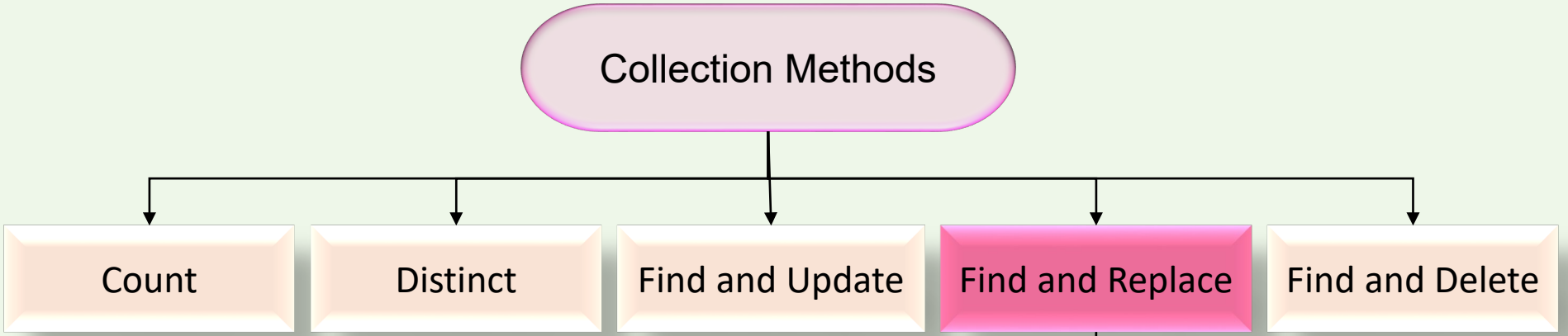
Distinct Method



Find and Update Method



Find and Replace Method

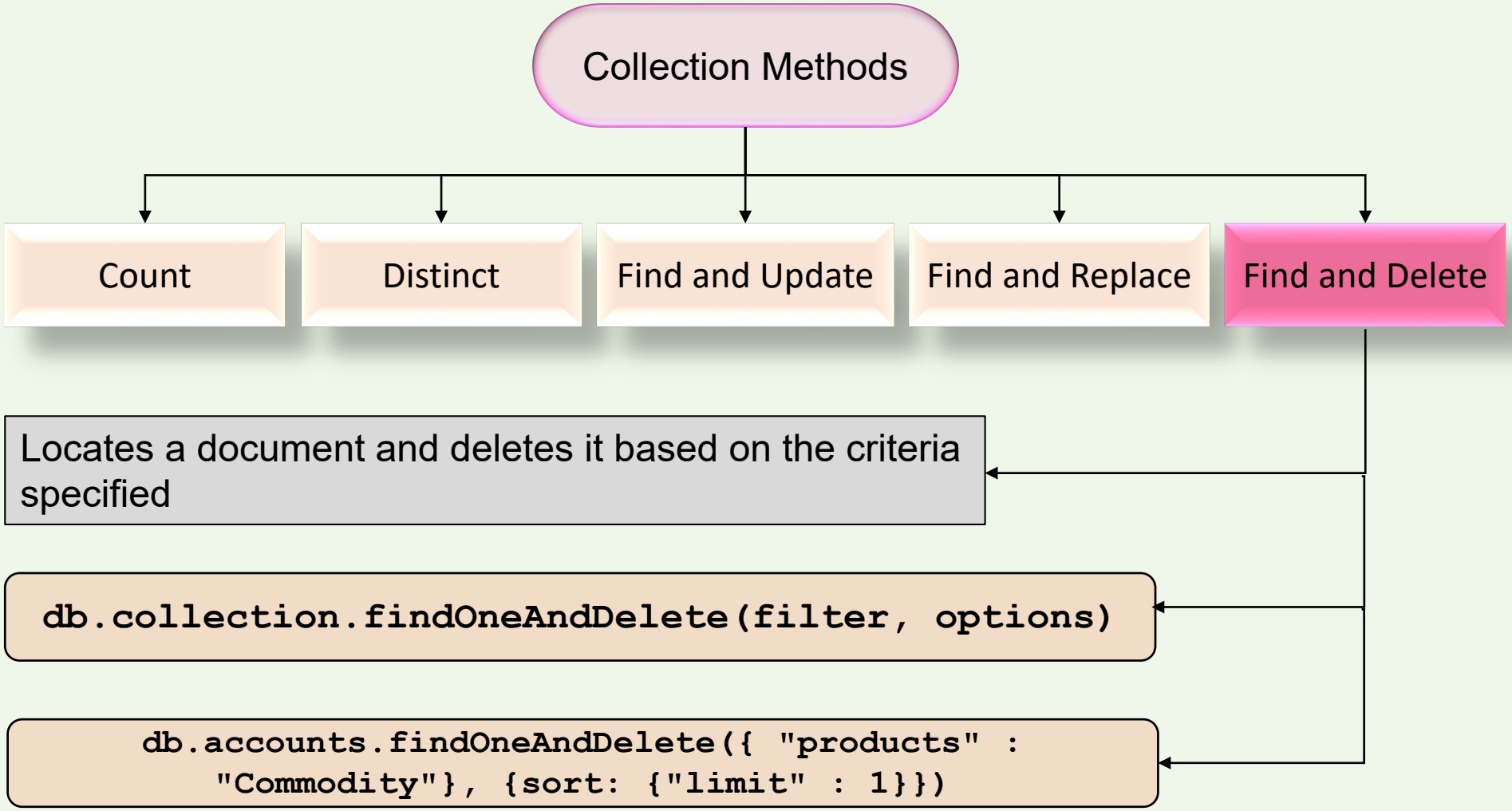


Locates a document and replaces it based on the conditions specified

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

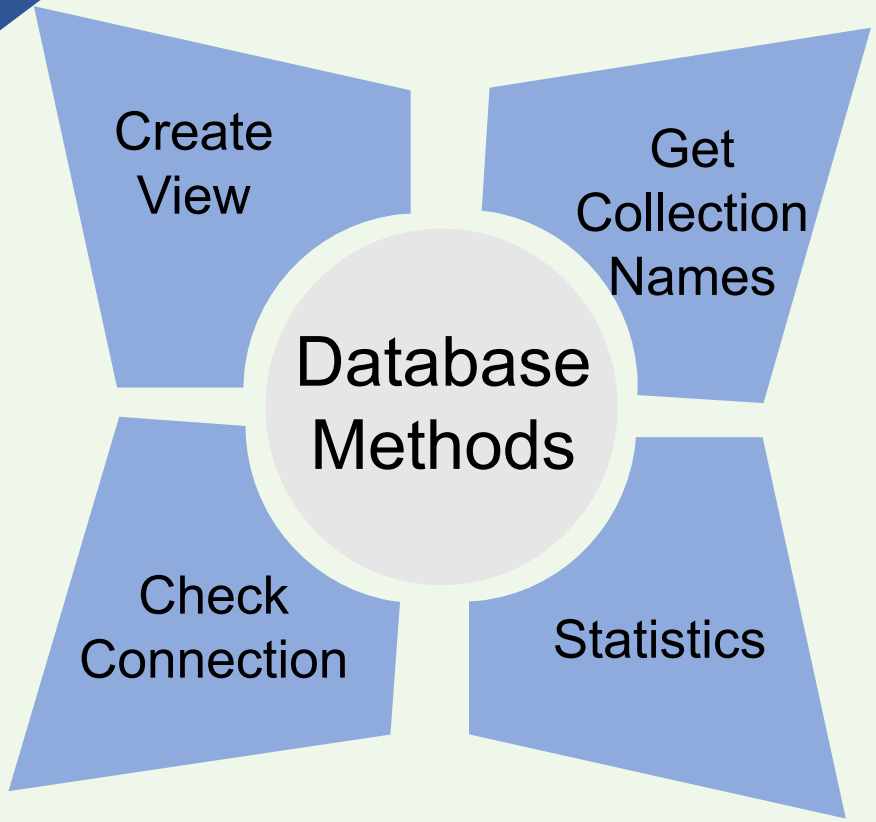
```
db.accounts.findOneAndReplace({ "limit" : { $gt : 5000 } },
{"product" : "Commodity", "limit" : 2500 }, {returnNewDocument:true})
```

Find and Delete Method



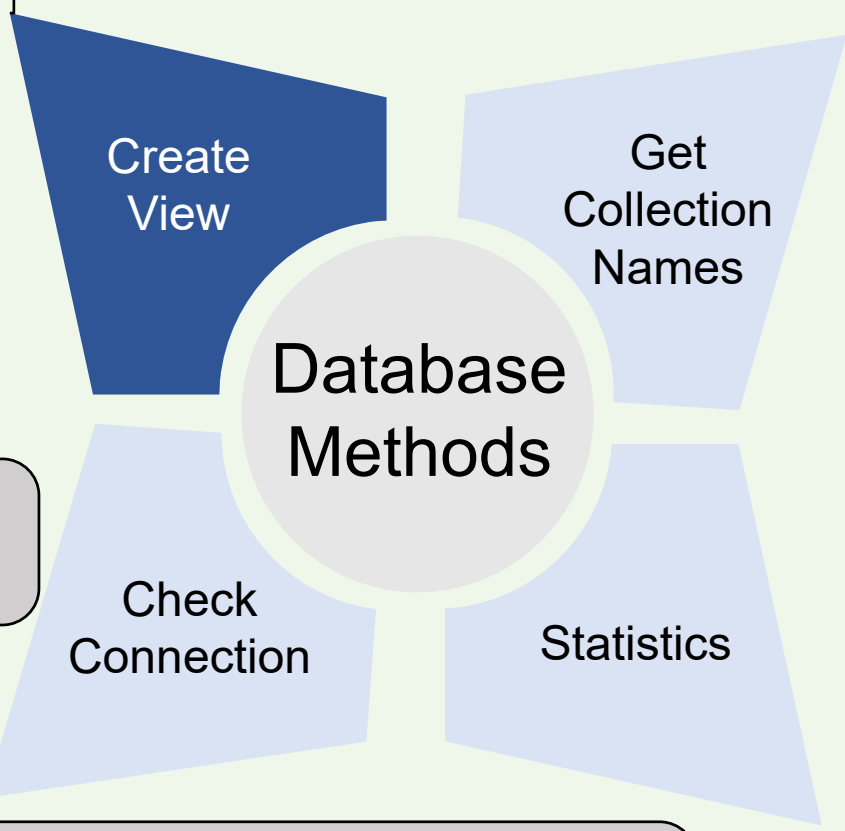
Database Methods

Database methods help to create, run, drop, and fetch data from collections and views in a database.



Create View Method

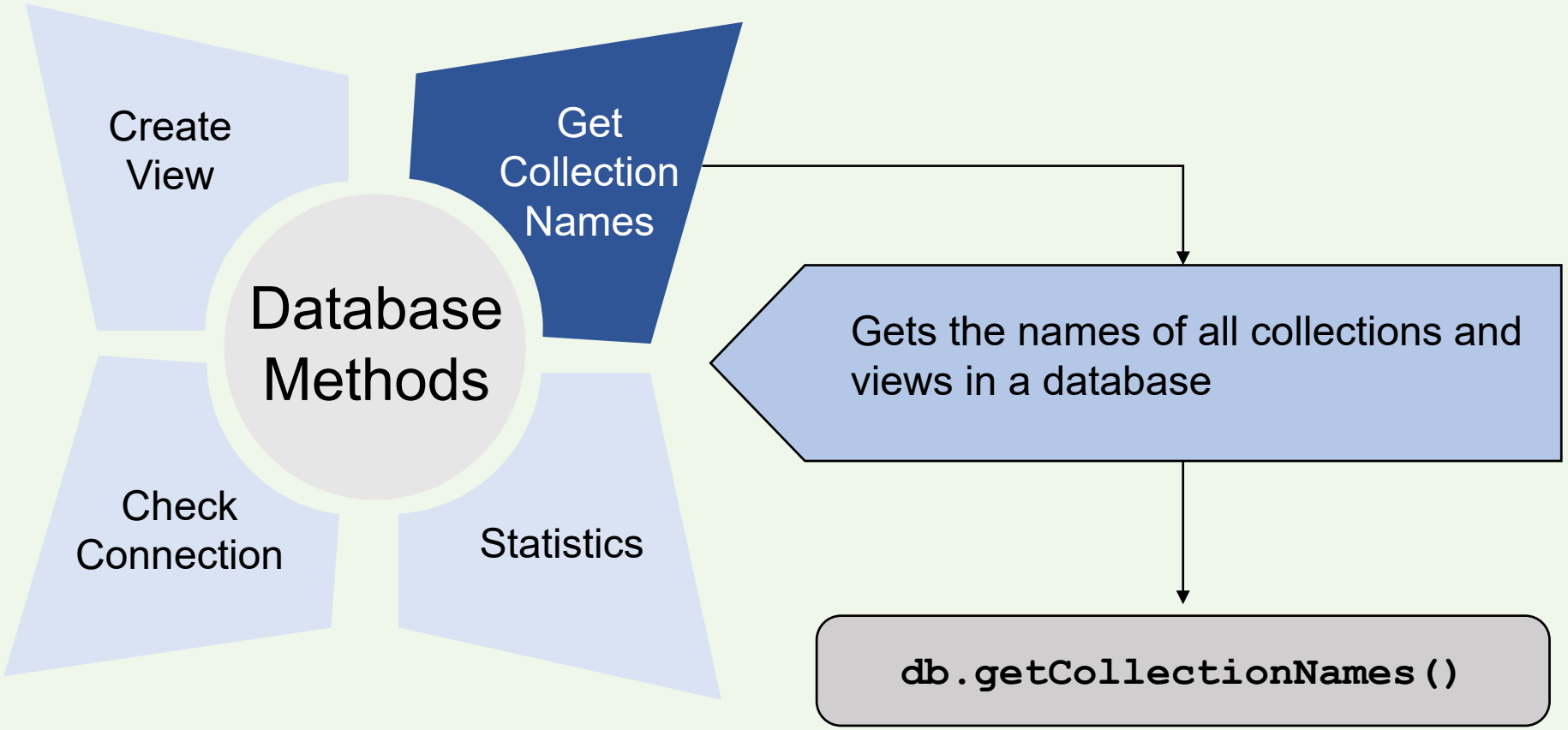
Creates a view based on a collection or another view



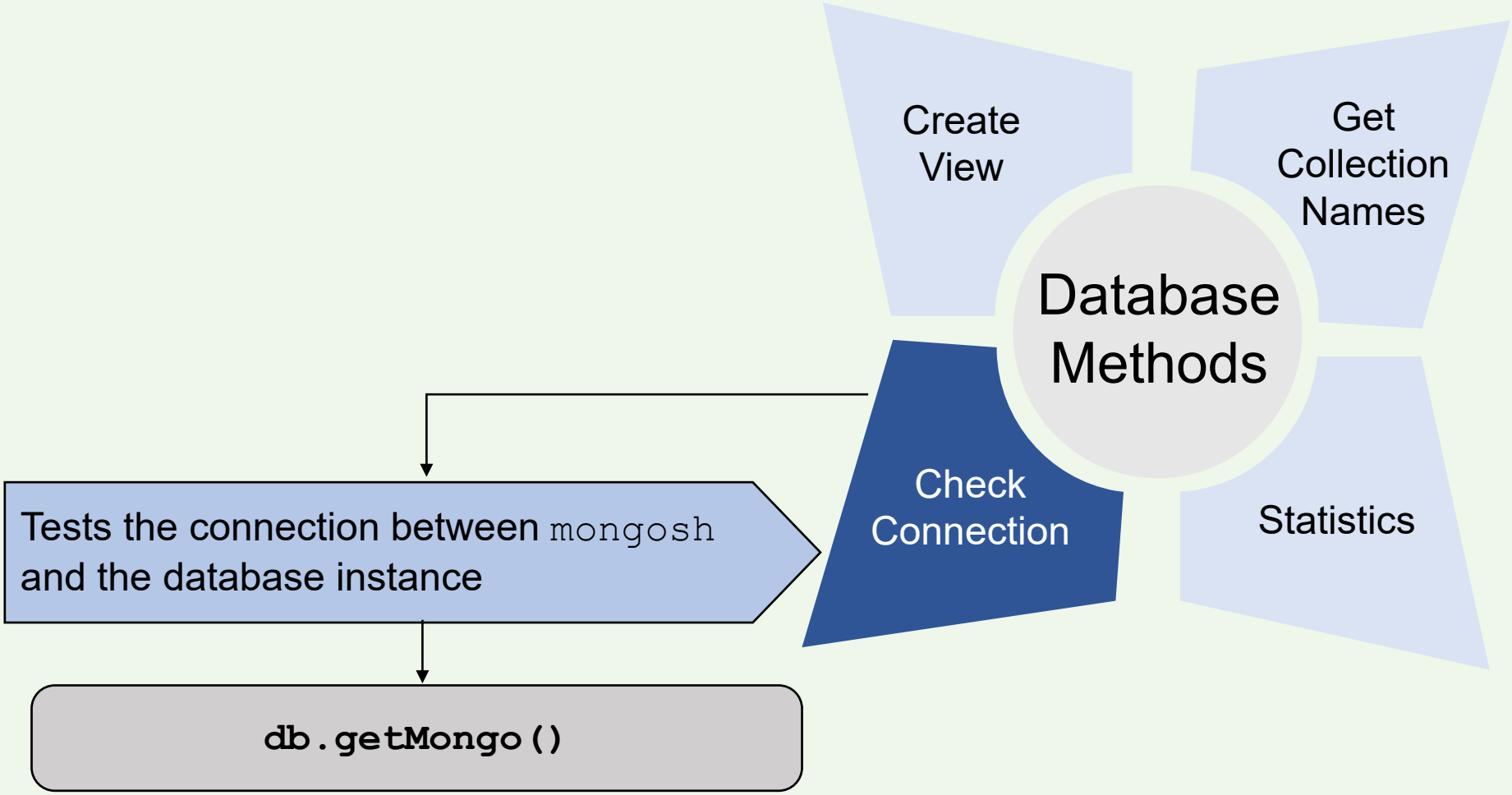

```
db.createView(view, source, pipeline, collation)
```

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

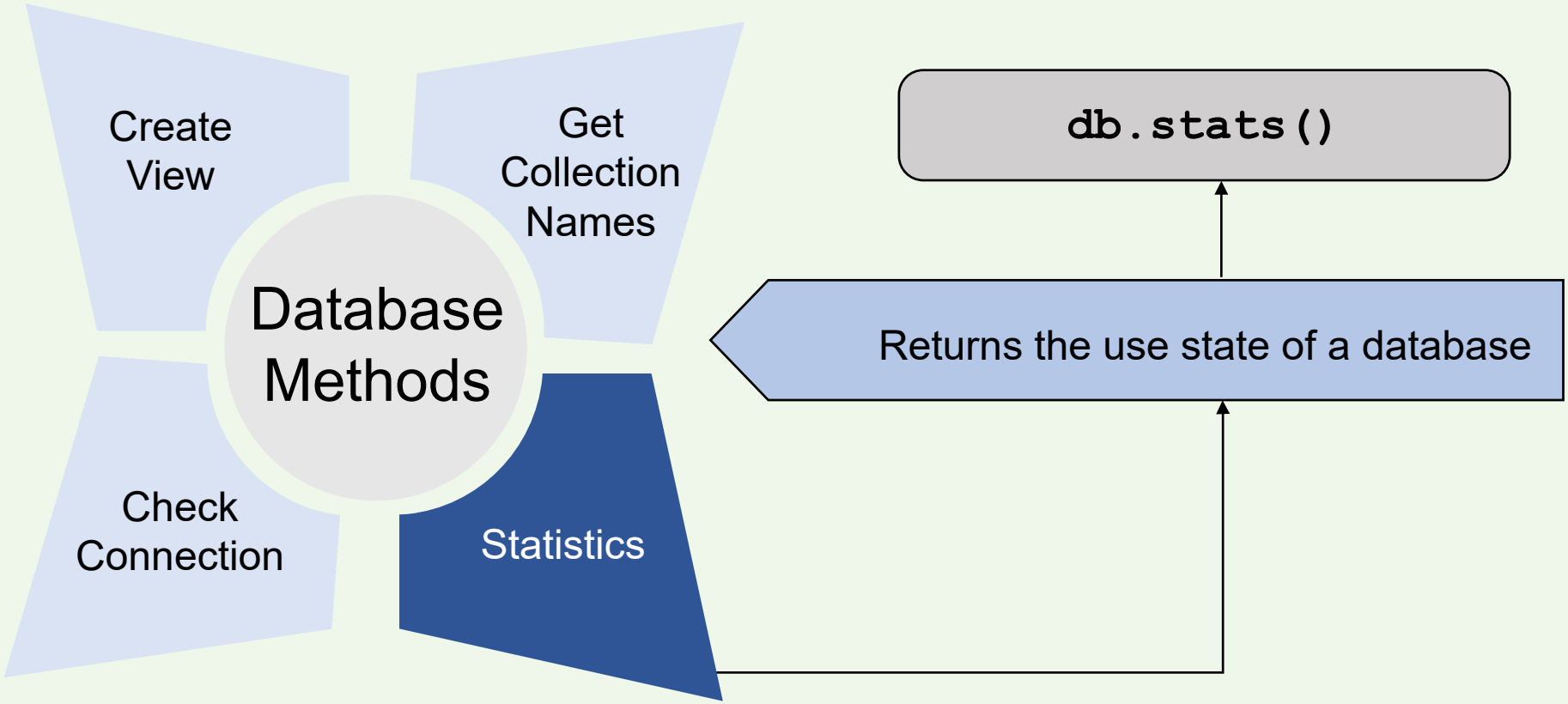
Get Collection Names Method



Check Connection Method



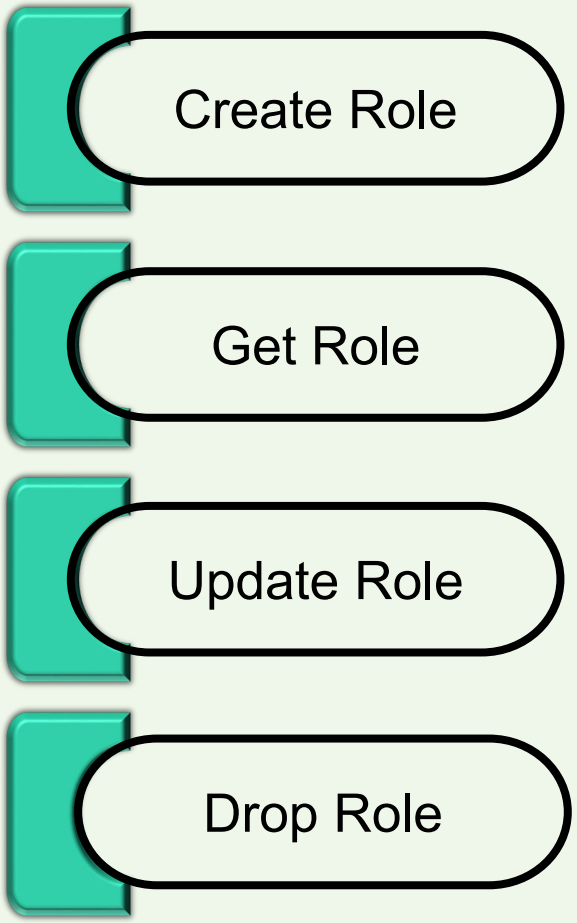
Statistics Method



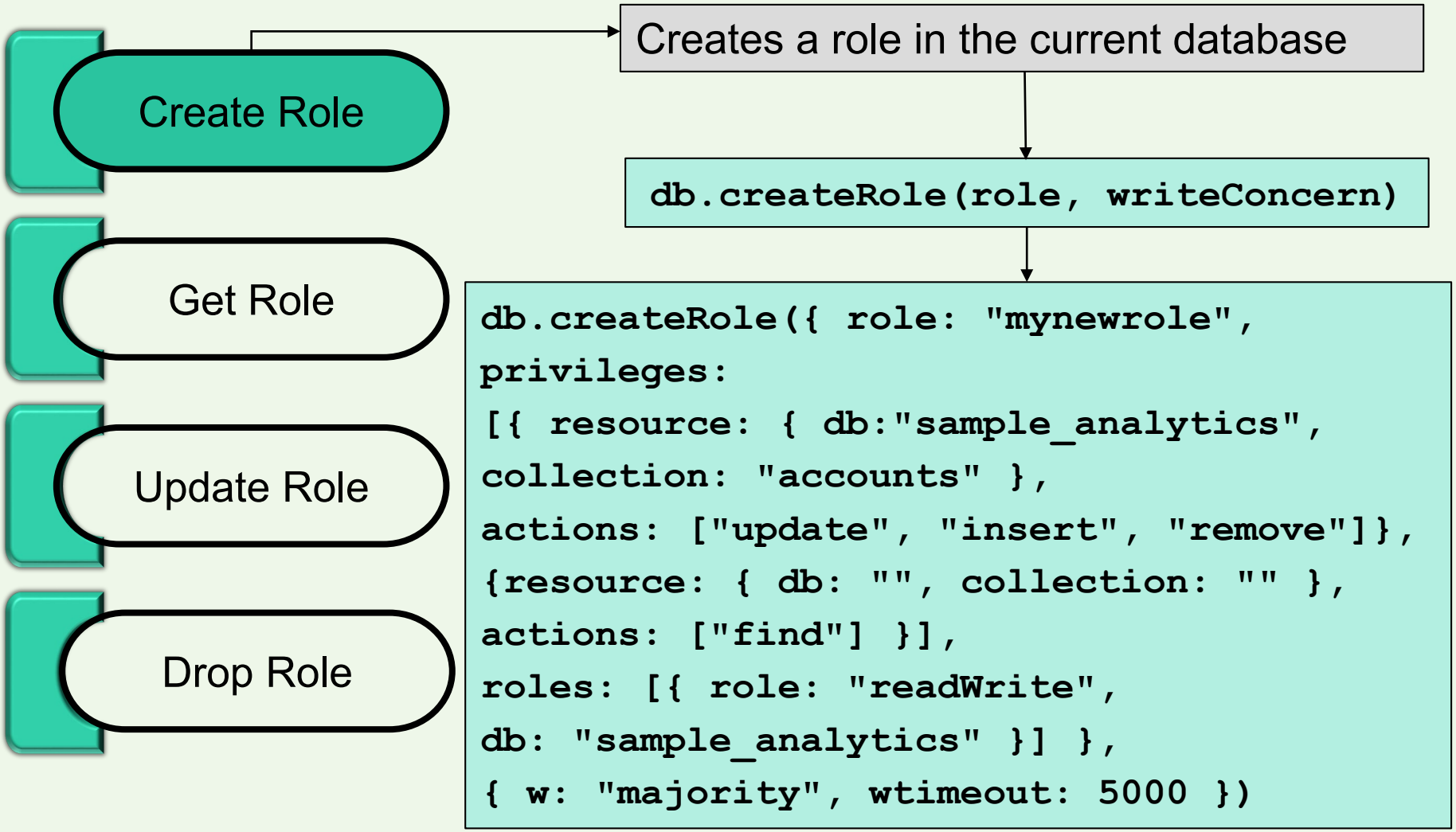
Role Management Methods

Role management methods

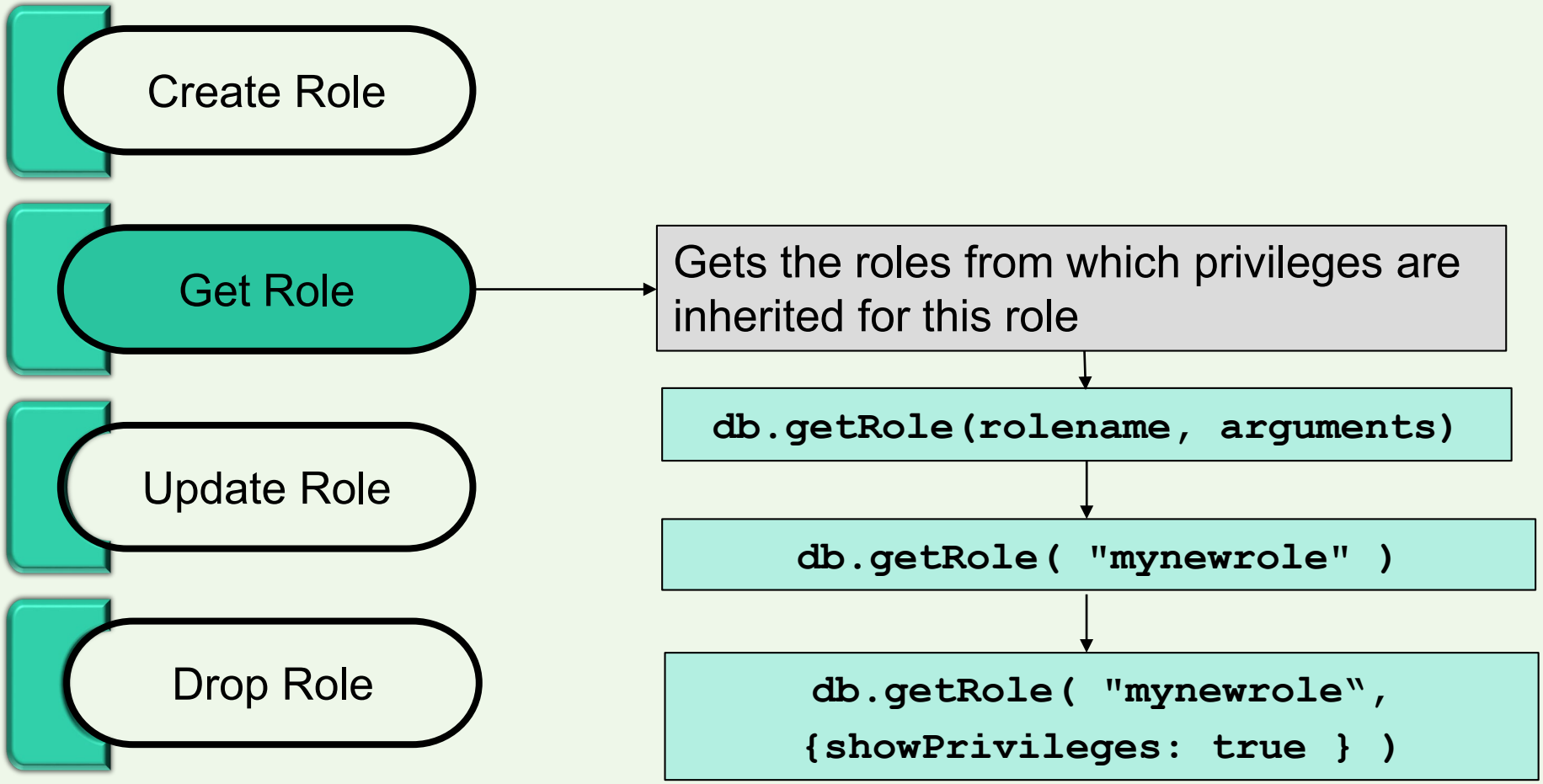
- Help to create, update, and drop roles
- Facilitate users to grant and revoke privileges to roles



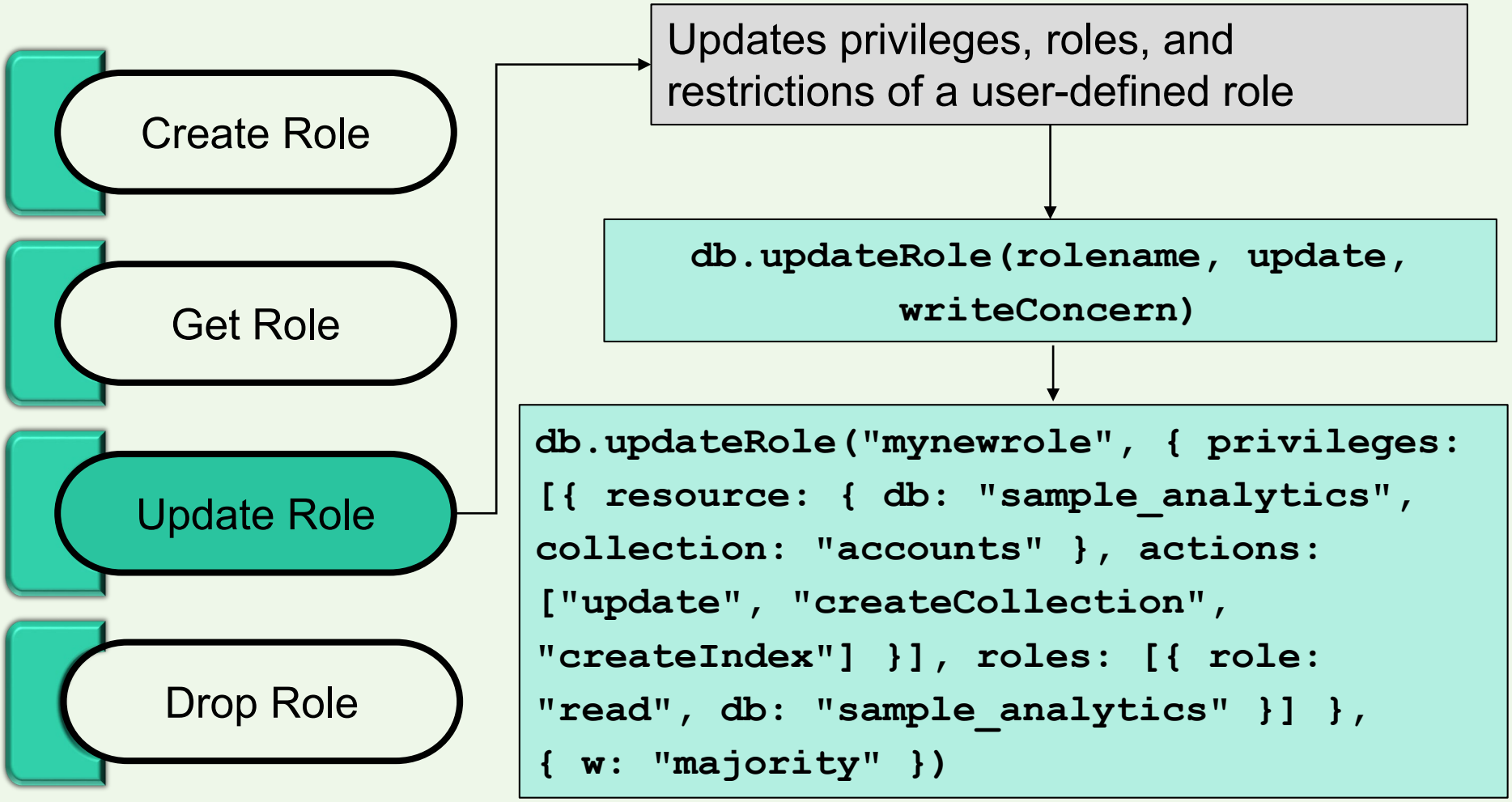
Create Role Method



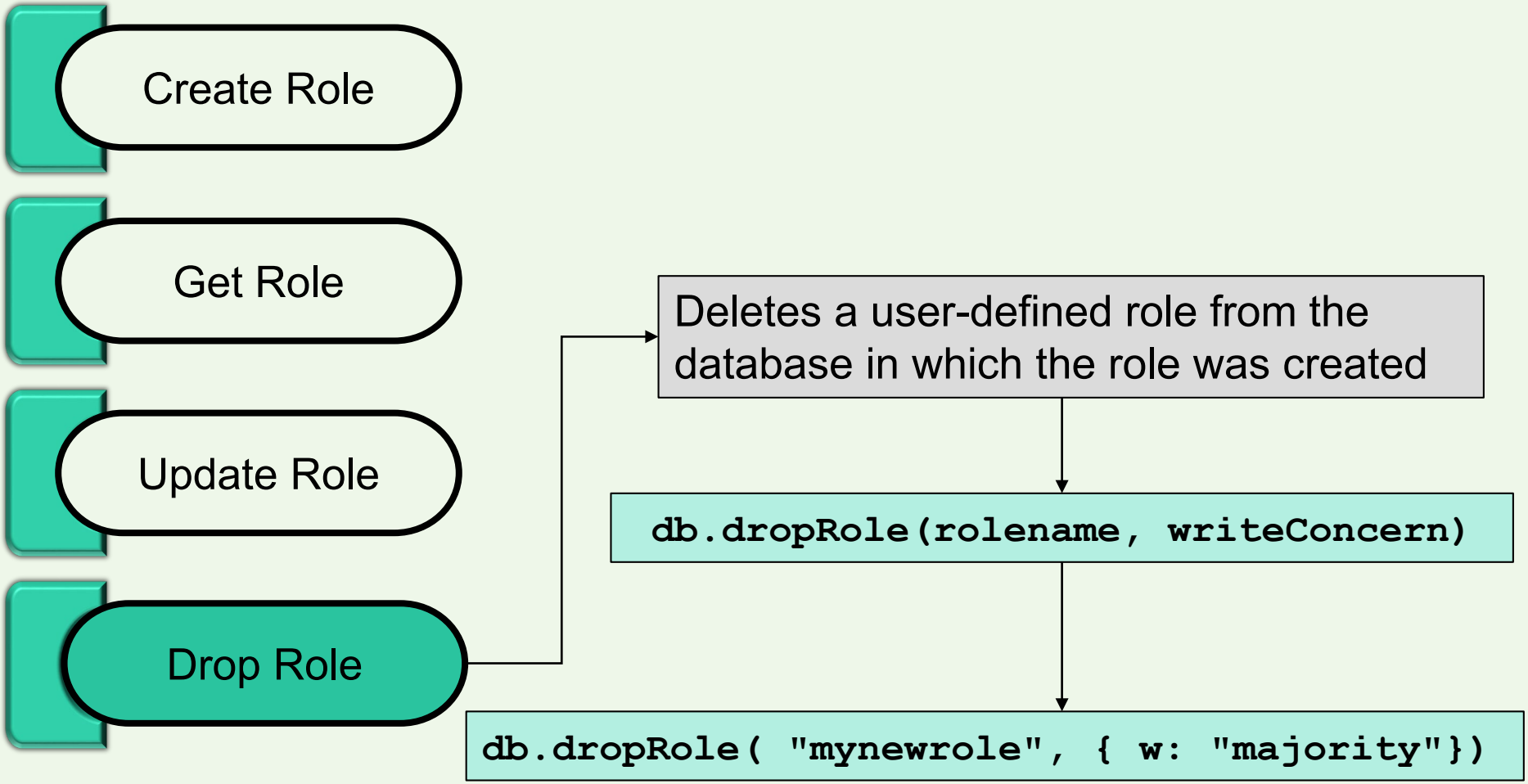
Get Role Method



Update Role Method

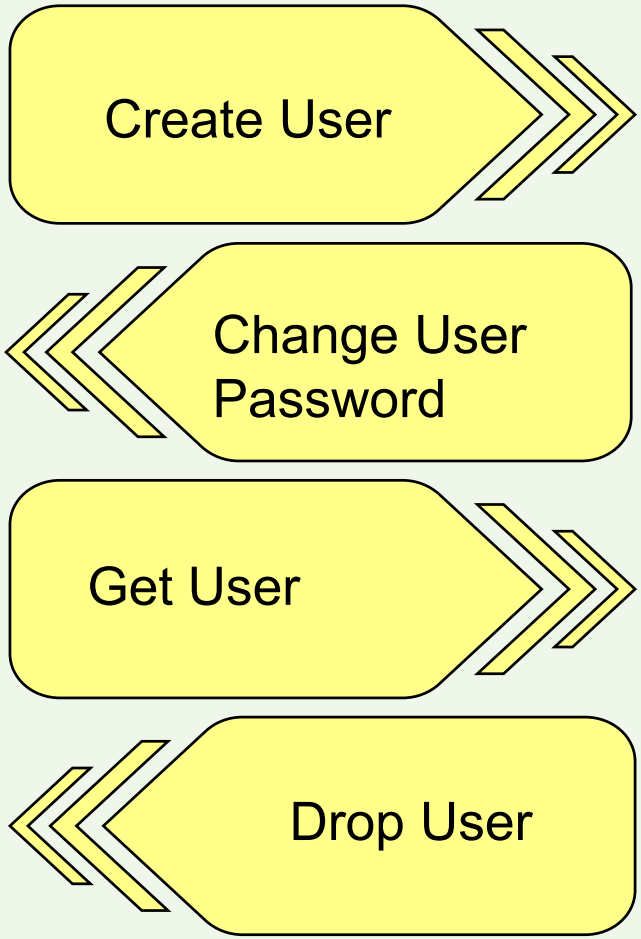


Drop Role Method

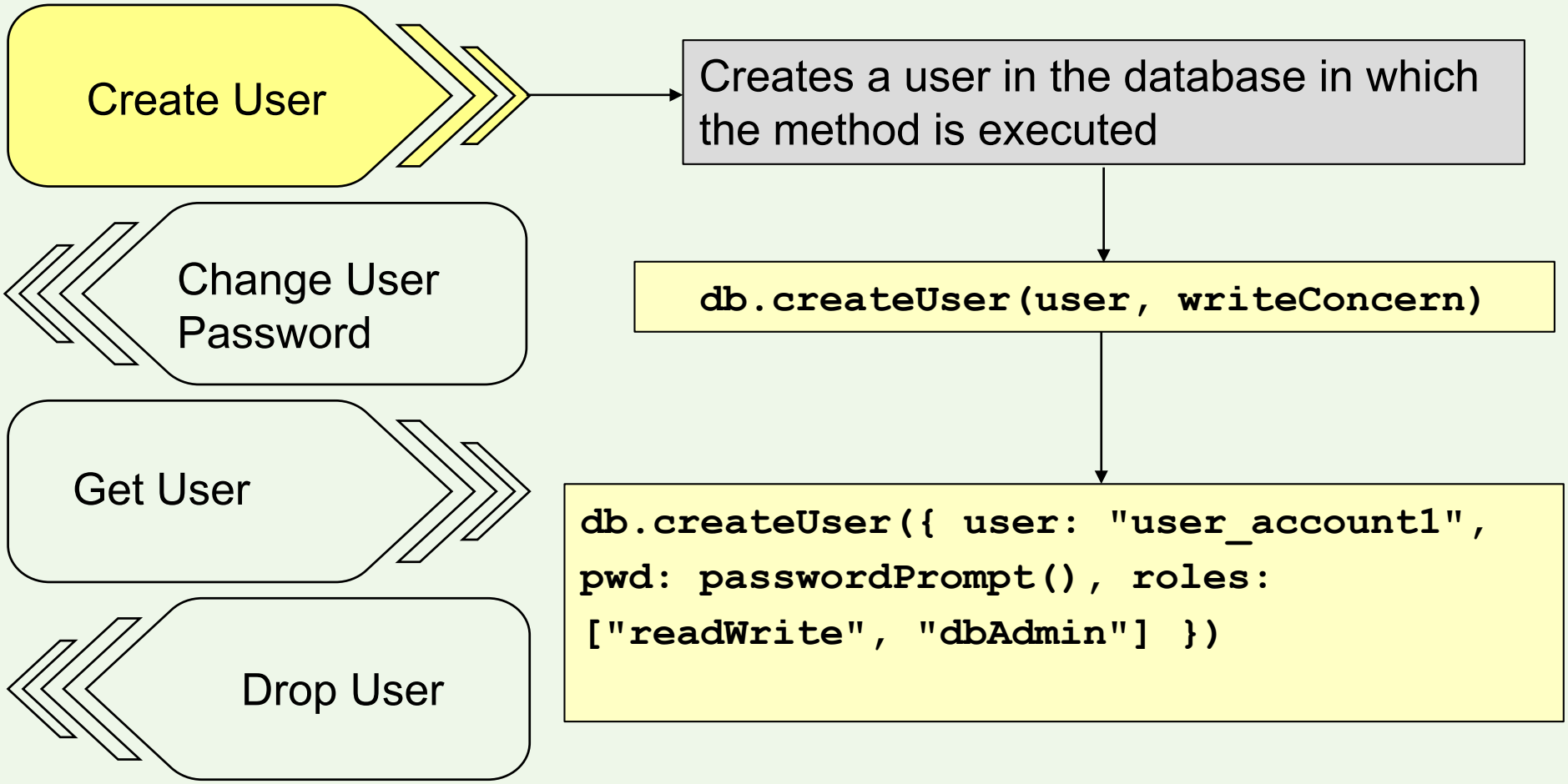


User Management Methods

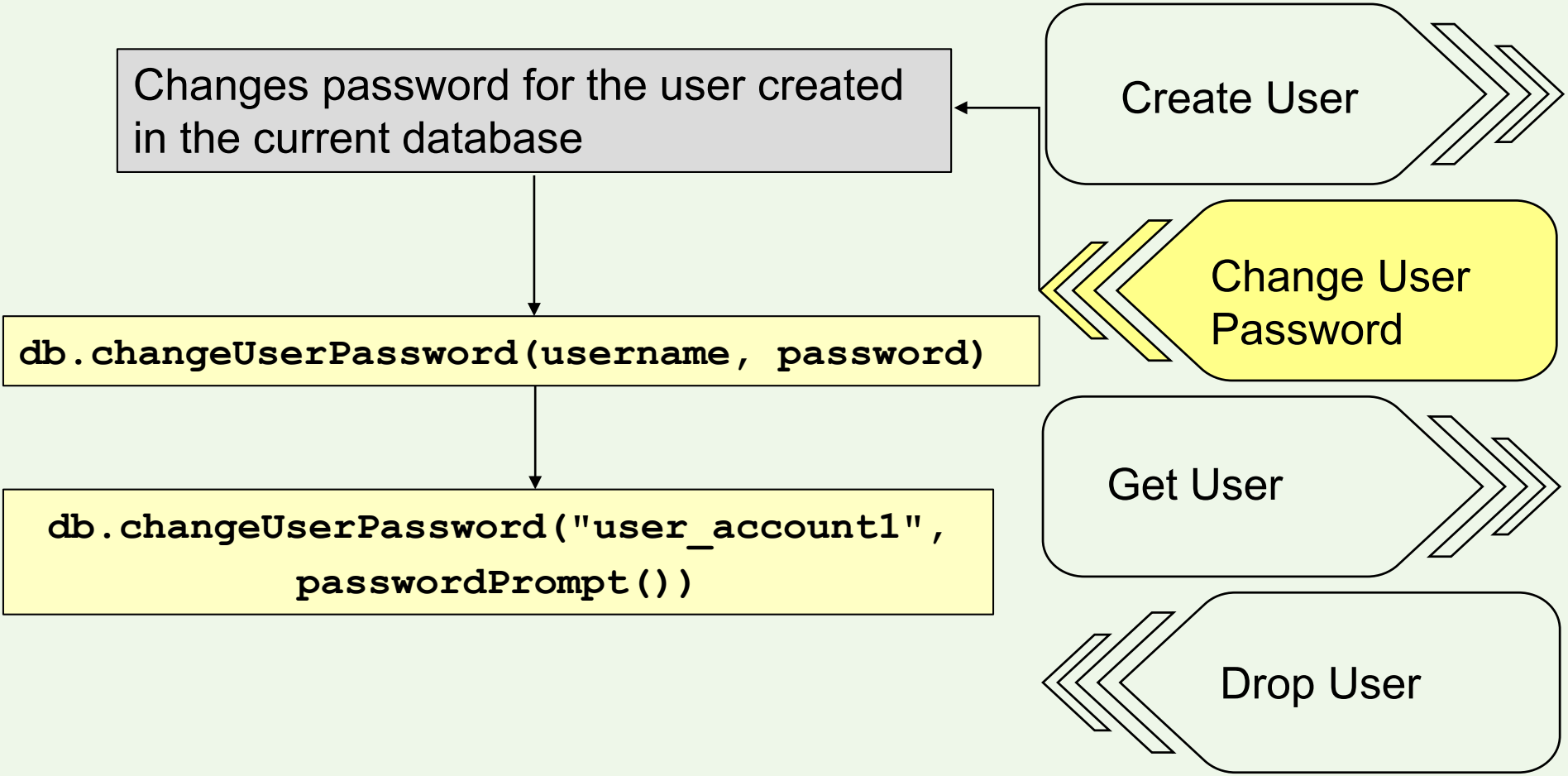
User management methods help to create a user, get roles to the user, change user password, and drop the user.



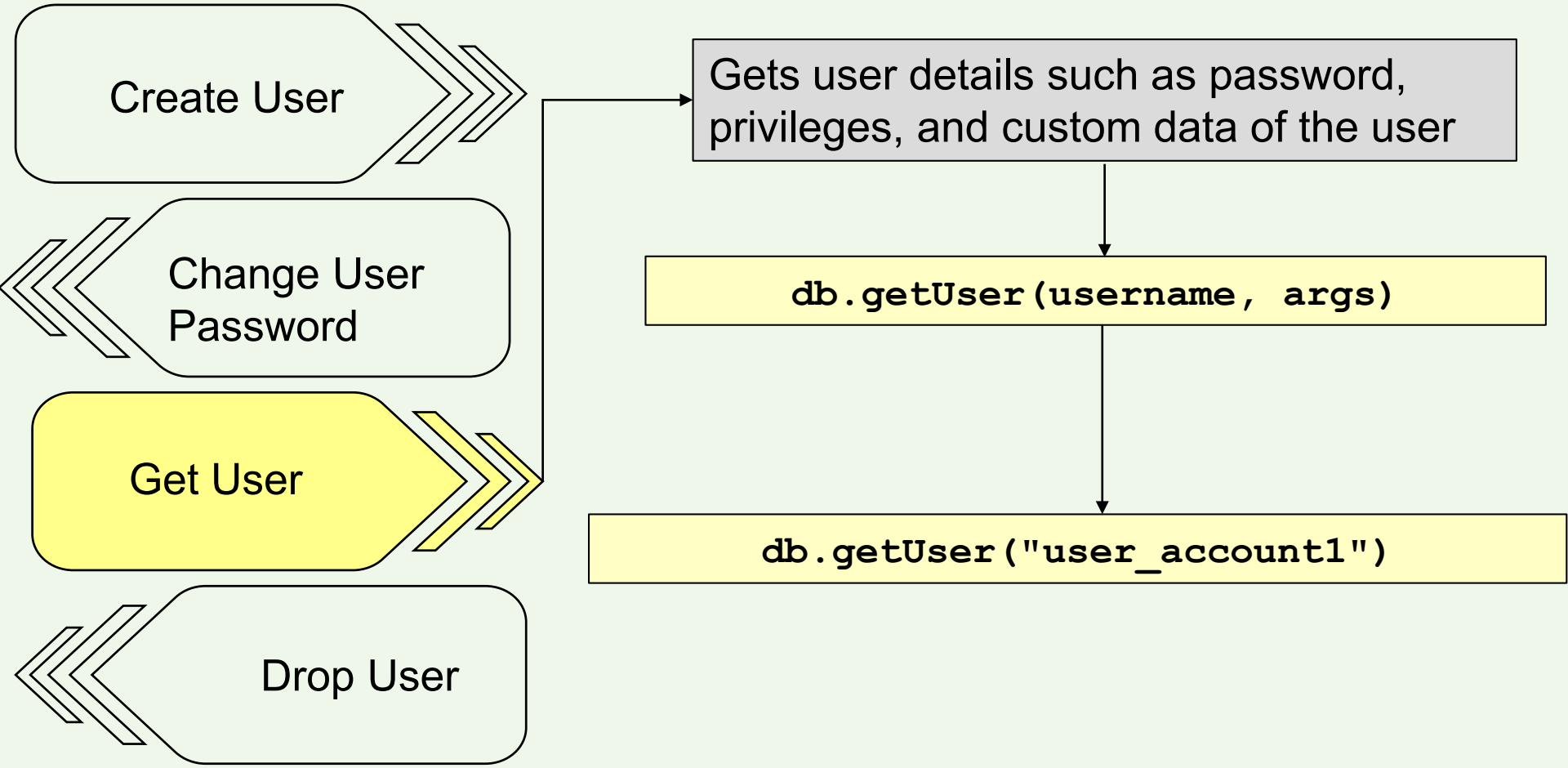
Create User Method



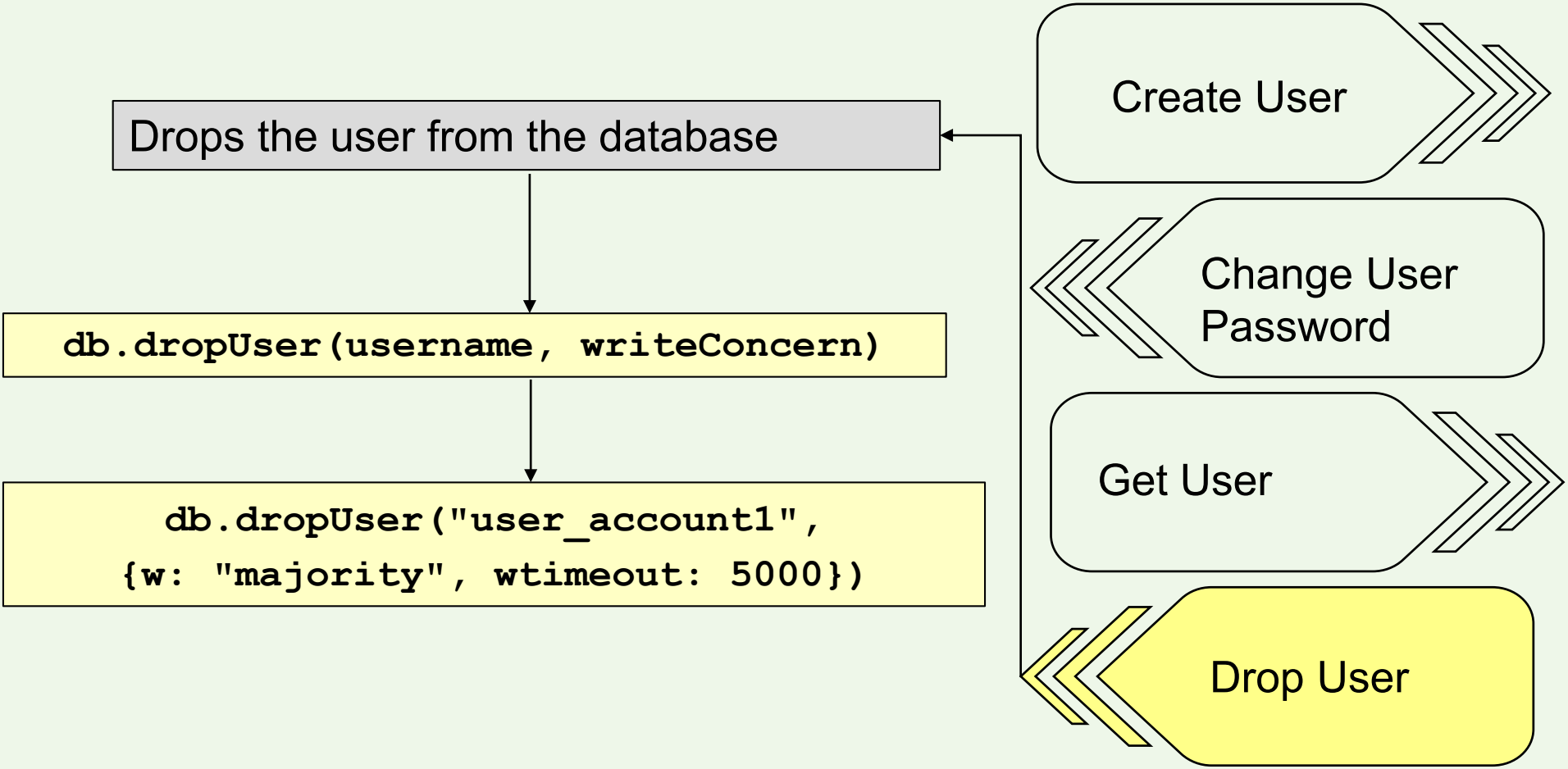
Change User Password Method



Get User Method



Drop User Method



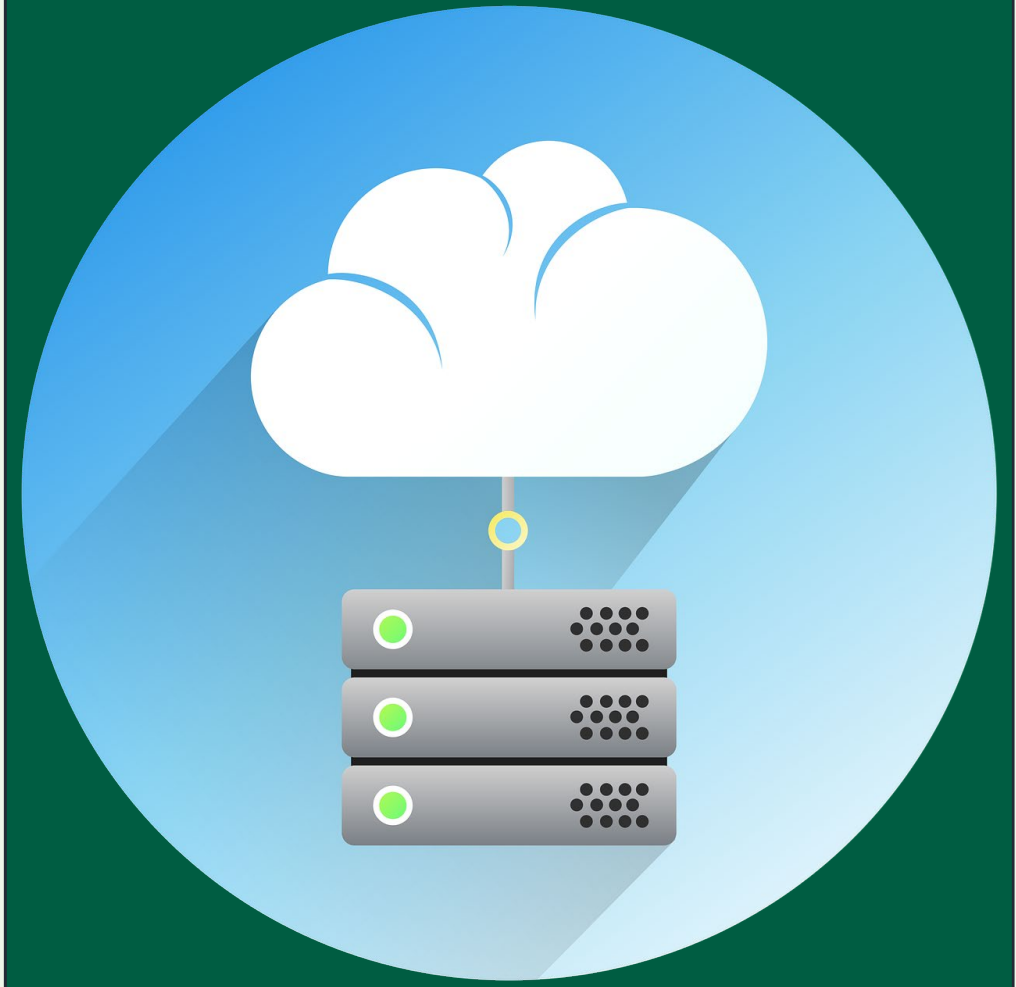
Summary

- 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.

Session: 7

MONGODB INDEXING

Managing Large Datasets Using MongoDB



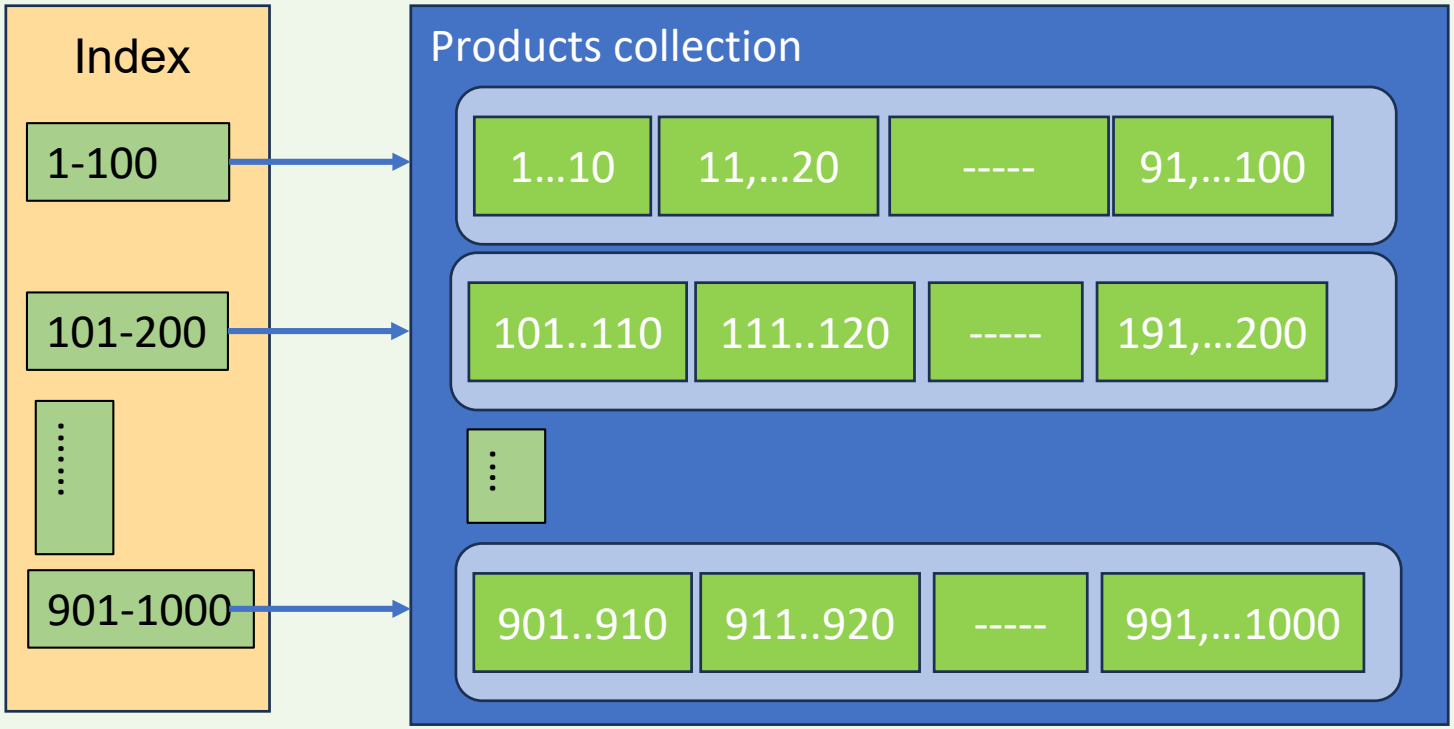
Objectives

- Explain indexes
- Describe the types of indexes
- List the advantages of using the indexes
- Explain how to create and drop indexes

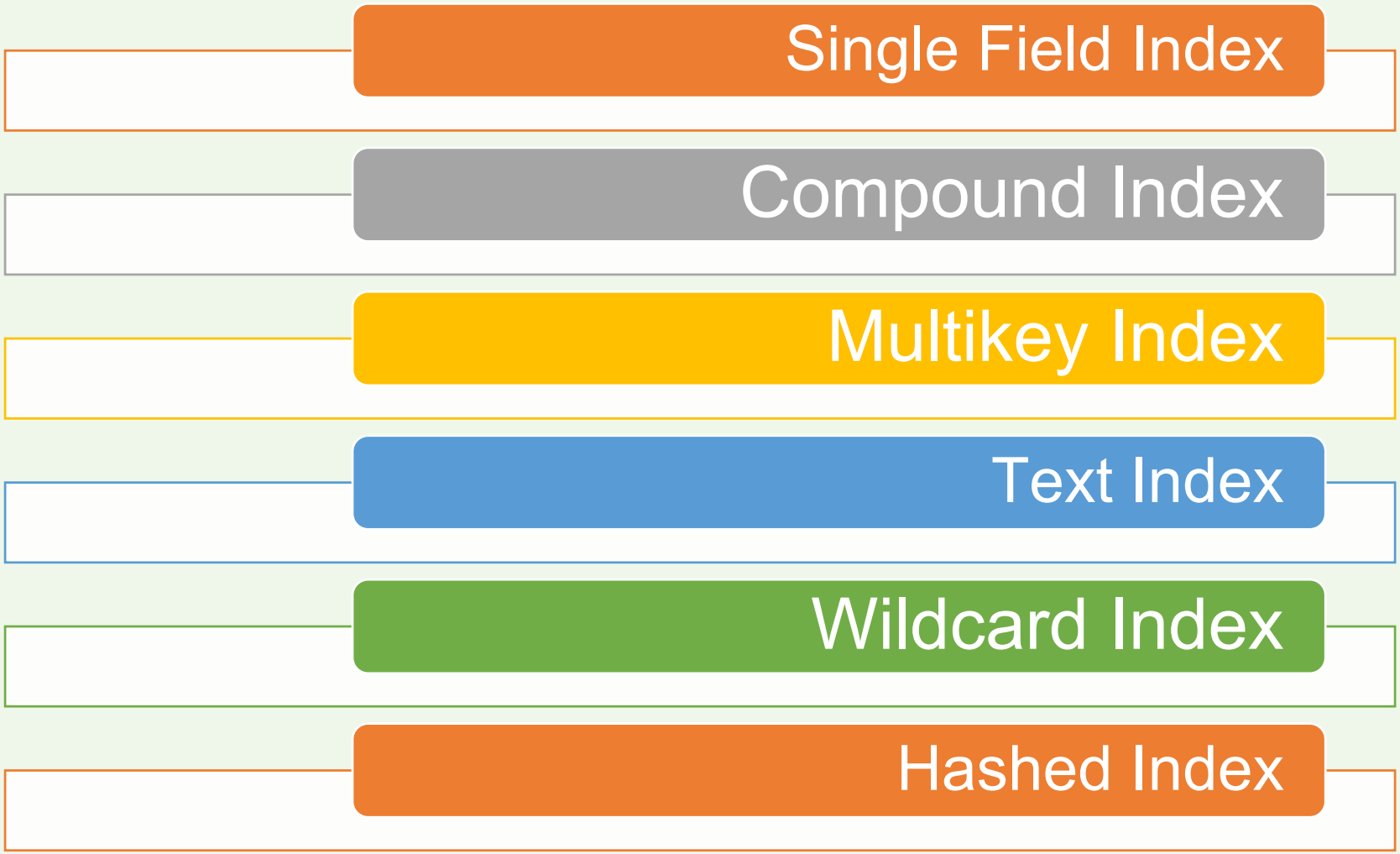
What are Indexes?

- Special data structures that help to speed up query performance
- Pointers to the database to provide faster access to data

Index sorted in ascending order



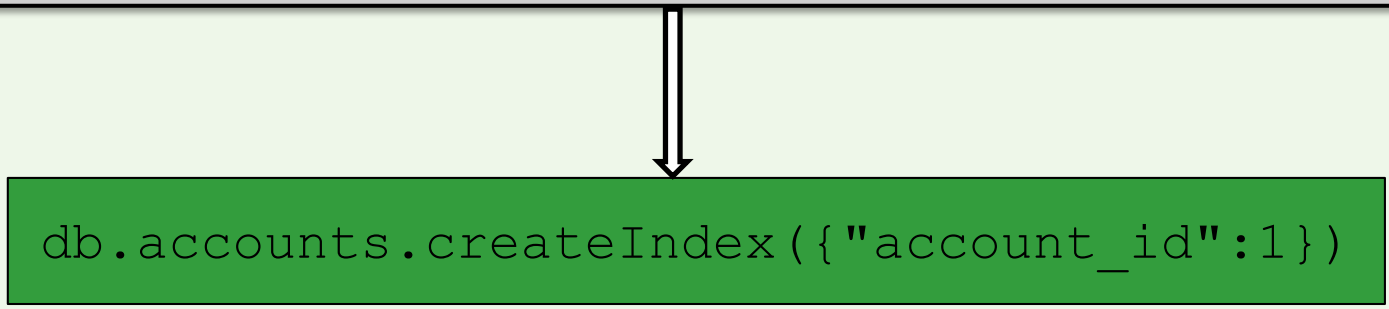
Types of Indexes



Single Field Index

Single index is created on only one field of the document in a collection.

```
db.Collection_name.createIndex(key,option,commitQuorum)
```

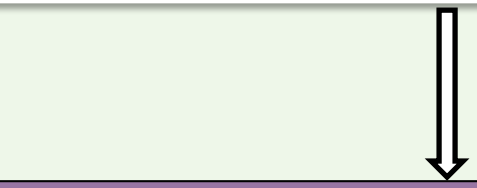


```
db.accounts.createIndex({"account_id":1})
```

Compound Field Index

Compound field index is created on multiple fields of the documents in a collection.

```
db.collections.createIndex({fieldName1:ord1,
fieldName2:ord2, ...} )
```



```
db.grades.createIndex({class_id: 1,
student_id:-1}, {name: "compoundIndex"})
```

Multikey Index

Multikey indexes are created on each element of an array.

```
db.collections.createIndex
   (<arrayname : <type>)
```

```
db.accounts.createIndex({products:1},
        {name: "multiKeyIndex"})
```

Types of Multikey Index

Compound Multikey Index

- Includes only one array field as an indexed field
- Does not allow insertion of a document that violates the index

Index on Fields Embedded in Arrays

- Includes indexes created on embedded fields in an array
- Helps to run queries that include one of the indexed fields or both the indexed fields

Text Index

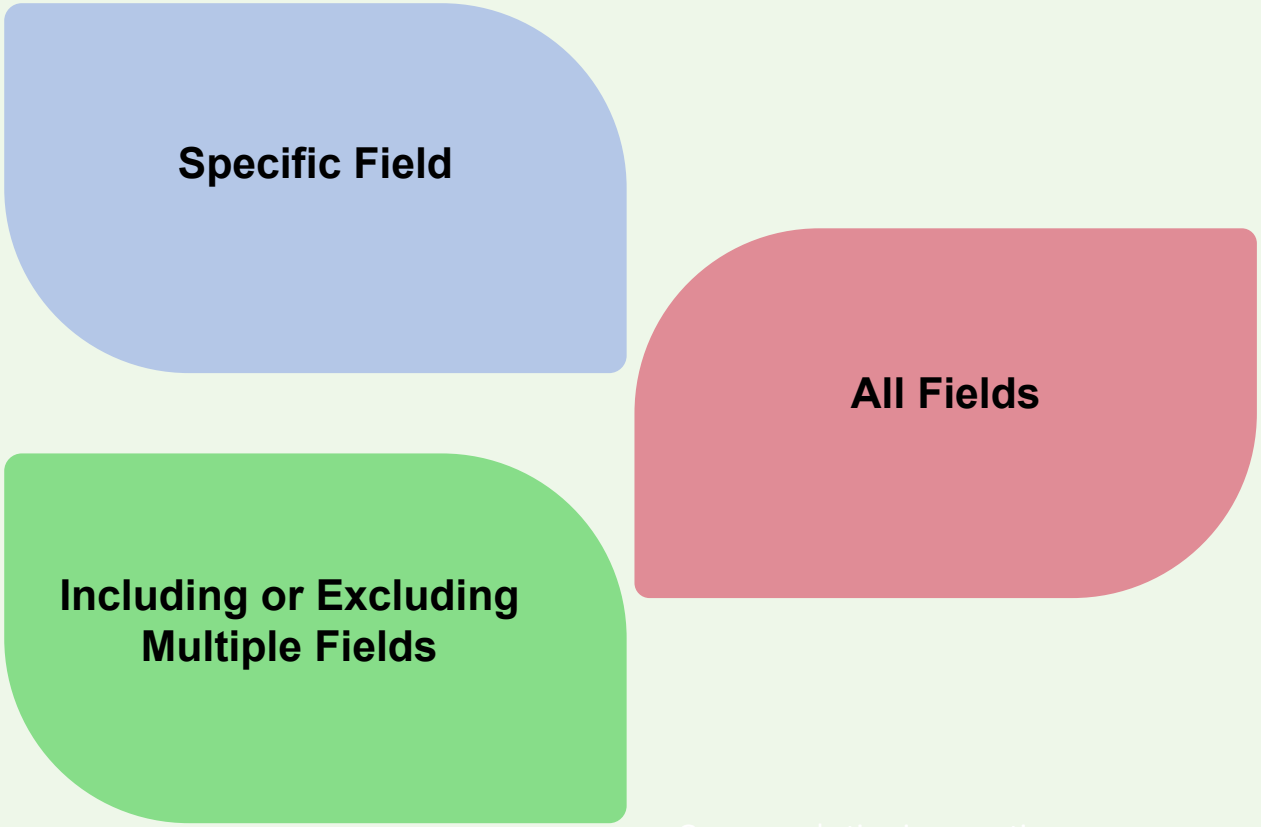
Text indexes are created on a text field or an array of text elements.

```
db.collections.createIndex(key, options, commitQuorum))
```

```
db.inspections.createIndex({"business_name":1})
```

Wildcard Index

Wildcard index supports queries against unknown or arbitrary fields. Wildcard indexes can be created on:



Wildcard Index – Specific Field

```
db.collections.createIndex
    ({"fieldA.$**" : 1})
```

Specific Field

All Fields

Including or Excluding Multiple Fields

```
db.inspections.create
        Index( {
"address.$**" : 1 } )
```

Wildcard Index – All Fields

Specific Field

All Fields

Including or Excluding Multiple Fields

```
db.collection.createIndex(
{ "$**" : 1 } )
```

```
db.inspections.create
Index( { "$**" : 1 }
)
```

Wildcard Index – Multiple Fields

```
db.collection.createIndex(
      { "$**" : 1 },
  { "wildcardProjection" :
      { "fieldA" : 1,
"fieldB.fieldC" : 1 } } )
```

Specific Field

All Fields

Including or Excluding Multiple Fields

```
db.sales.createIndex
   ({ "$**" : 1 },
 {"wildcardProjection" :
      { "name" : 1,
"customer.gender" : 1 } })
```

Hashed Index

Hashed index is created on hashed values of the indexed field.

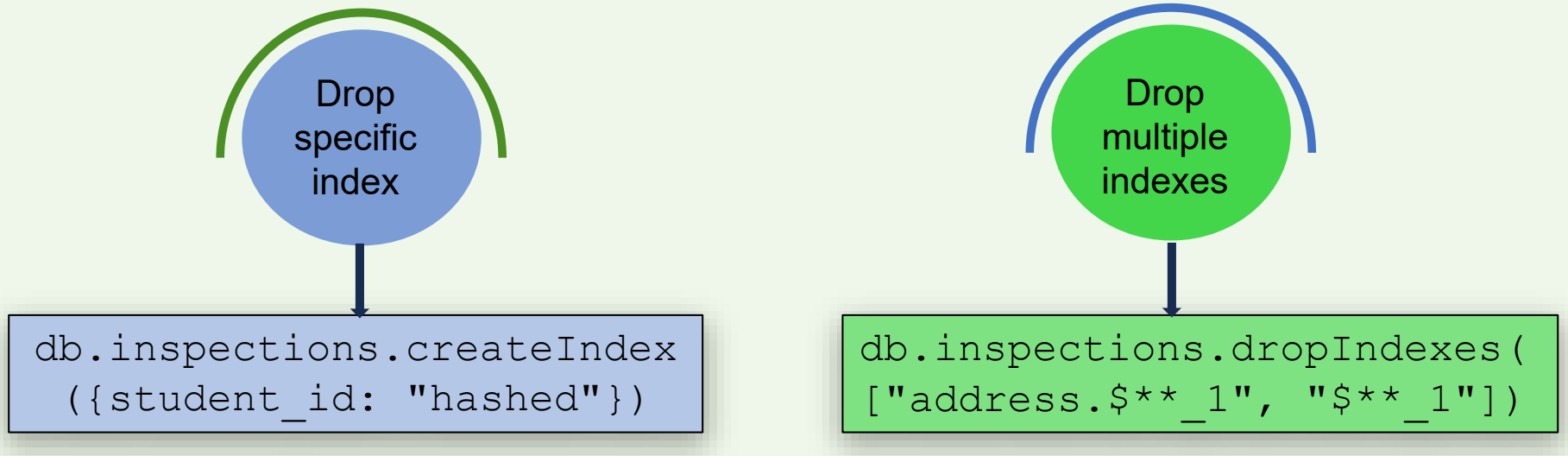
```
db.collections.createIndex({fieldname: "hashed"})
```

```
db.inspections.createIndex({student_id: "hashed"})
```

Drop Indexes

Drops the created indexes

```
db.collection.dropIndexes({indexes})
```



Summary

- Indexes are nothing, but data structures that store the values of a field or multiple fields in ascending or descending order.
- Single key index is created on a single field by specifying the order of sorting.
- Compound index is the index created on more than one field.
- Multikey index is an index created on an array field.
- Wildcard index helps in building indexes on unknown fields.
- Hashed indexes are the indexes created by hashing the value in the indexed field.
- All indexes created for a collection can be dropped except the default index created on the `_id` field.

Session: 8

MongoDB Replication and Sharding

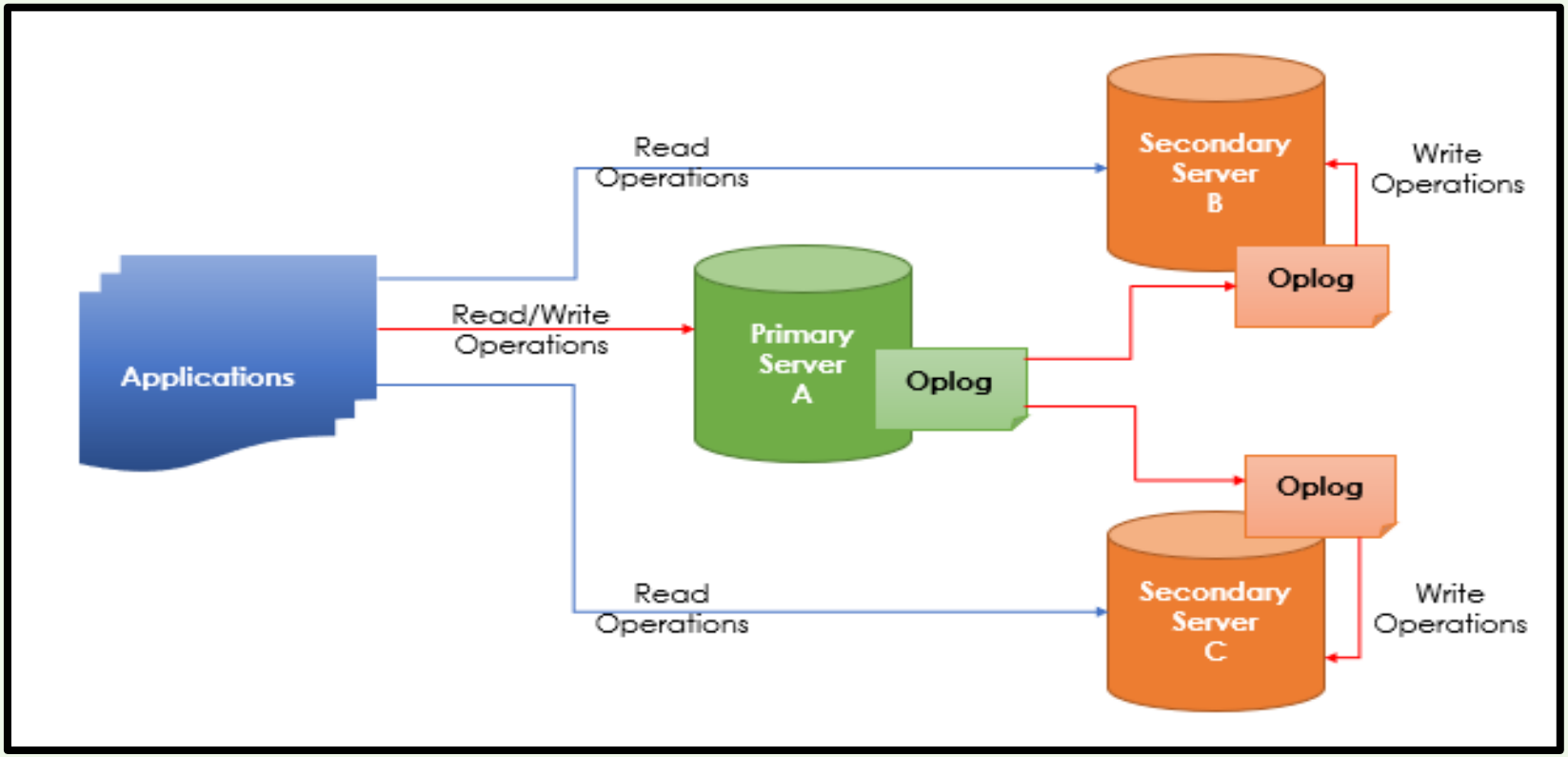
Managing Large Datasets Using MongoDB

Objectives

- Explain replication
- Describe the ways to implement replication
- Explain sharding
- Describe the ways to implement sharding

Replication Architecture

Replication:
- Is building duplicates of existing data on different servers.
- Improves data availability during server failure.



Primary-Secondary-Secondary (P-S-S) Configuration

A replica set is a group of primary and secondary servers.

Each server has its own `MongoDB` instance.

Data is replicated to secondary servers from a primary server.

The main server can both read and write data.

Replica servers or secondary servers read data from the primary server.

The secondary servers can only be used for read operations and do not support write operations.

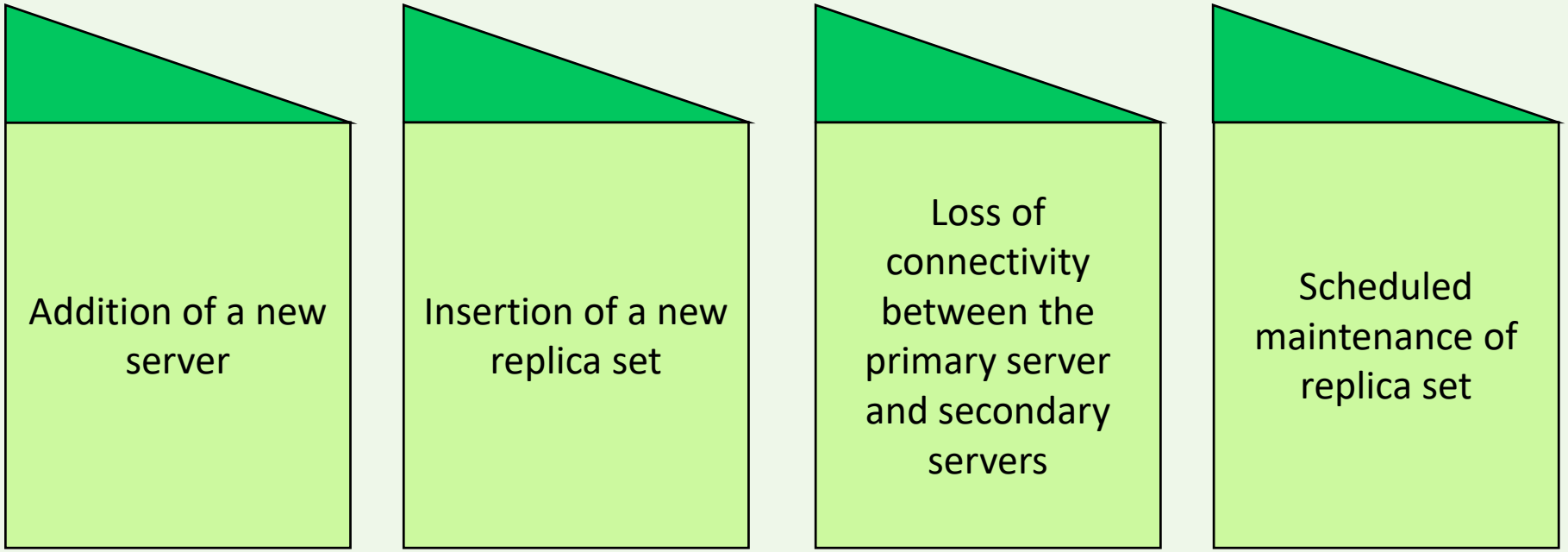
Advantages of Replication

- High data availability
- No waiting time in case of server failure

Replica Set Election

When the primary server fails, an election is held between the secondary servers.

Events that trigger an election:



The newly elected primary server turns into a secondary server when the original primary server becomes operational.

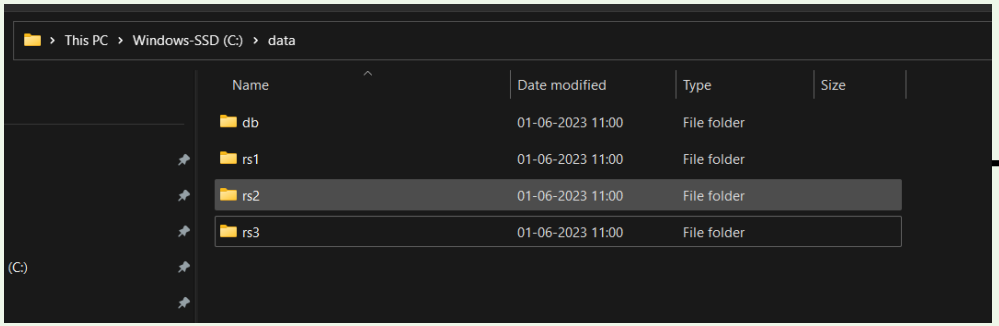
Replication Methods

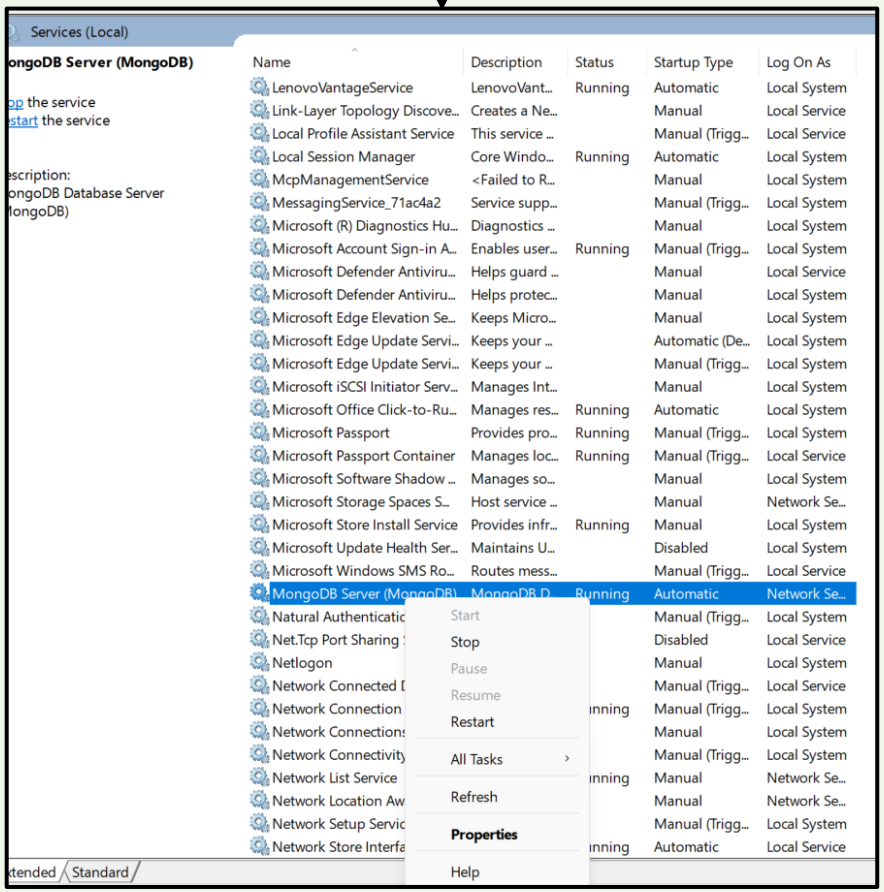
rs.add()	rs.syncFrom()	rs.stepDown()
rs.addArb()	Methods for Replication	rs.status()
rs.conf()		rs.remove()
rs.freeze()		rs.reconfigForPSASet()
rs.help()		rs.reconfig()
rs.initiate()	rs.printReplicationInfo()	rs.printSecondaryReplicationInfo()

Replication Implementation

Command

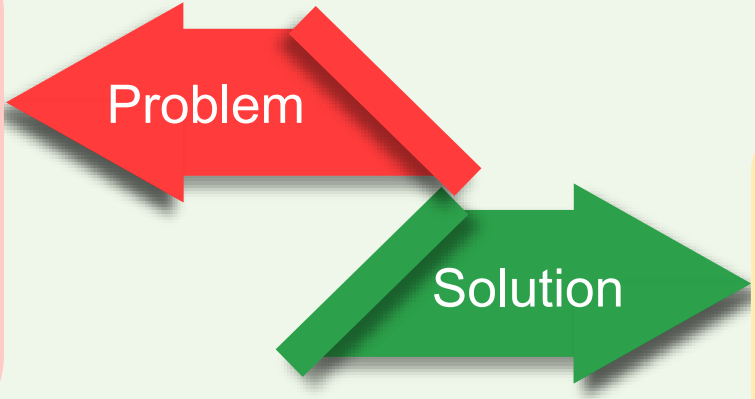
```
mongod --replSet replicaset
```





Sharding

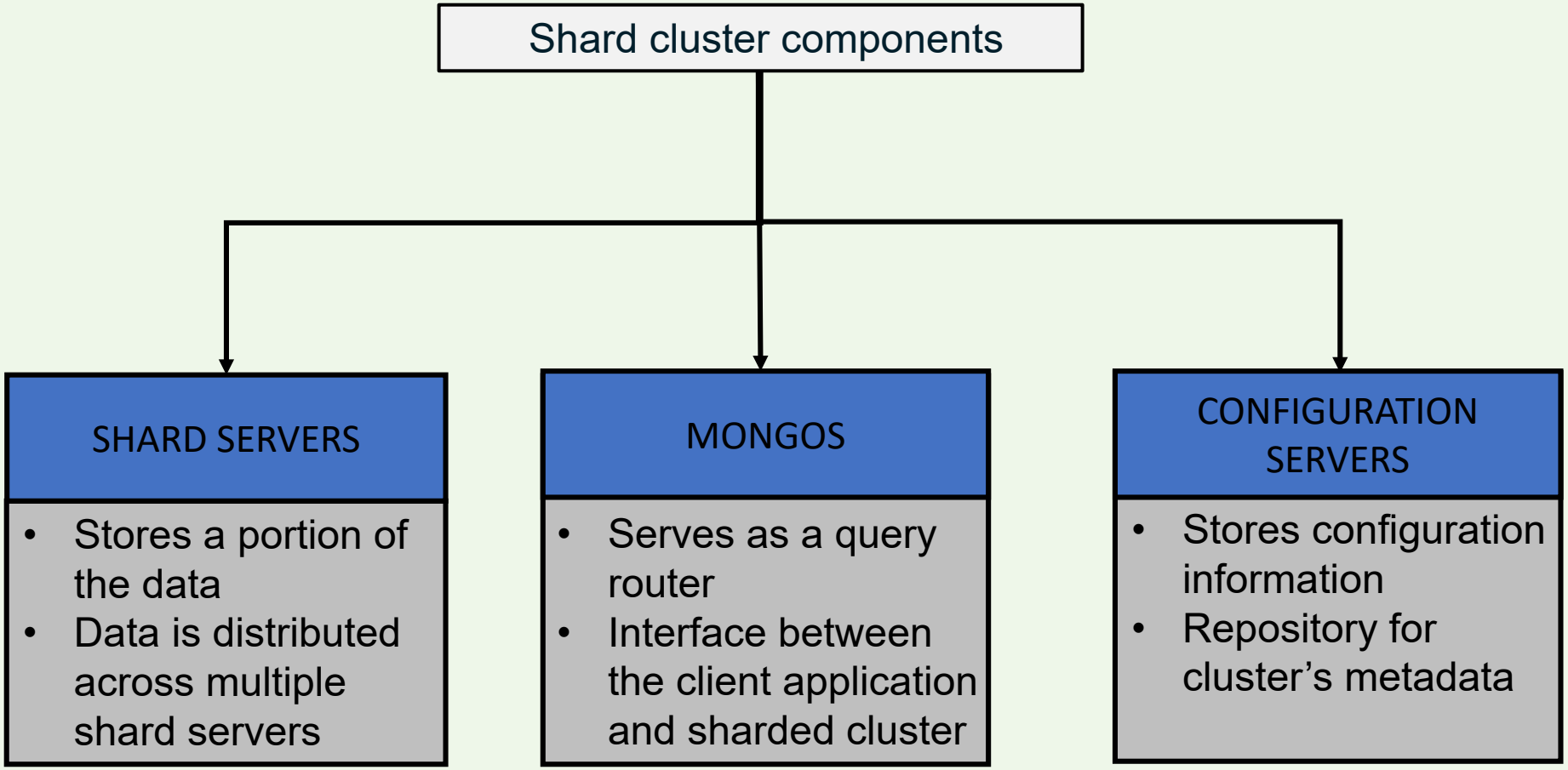
- Databases with huge volume of data
- Depletion of resources - CPU, RAM, and input-output devices



- Vertical scaling
- Horizontal scaling - Sharding

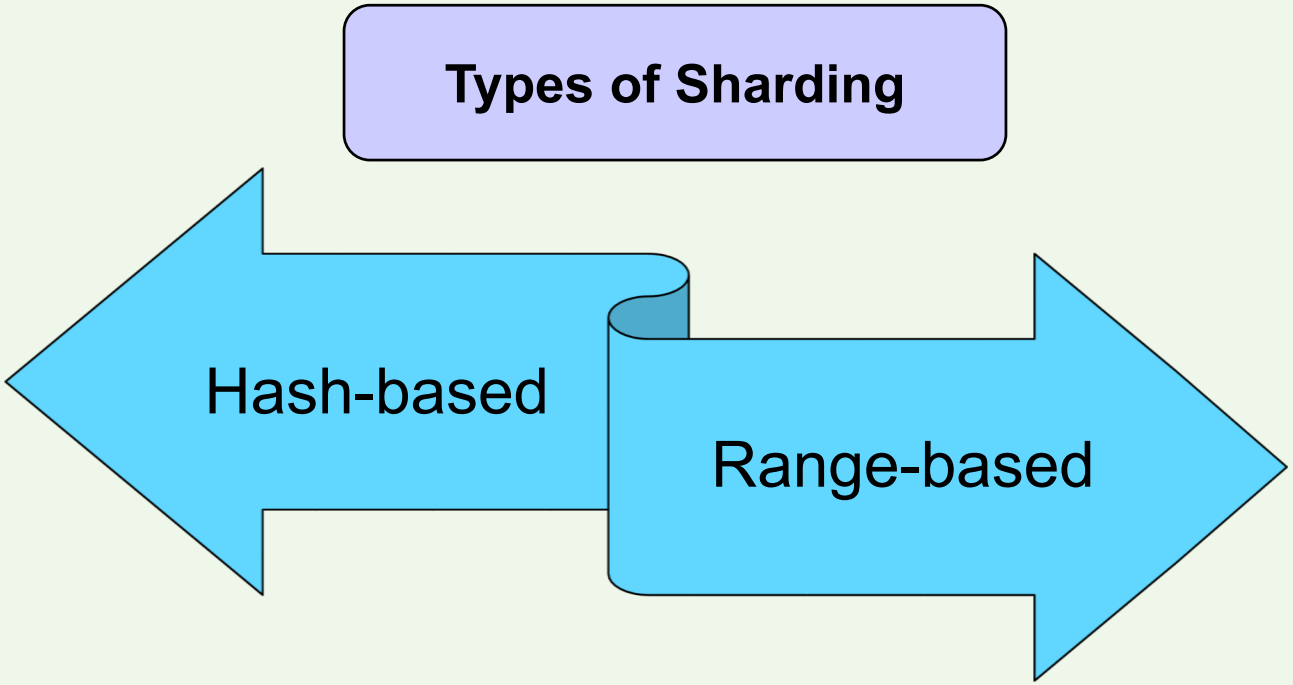
MongoDB supports horizontal scaling through sharding which involves distribution of data collections across multiple servers.

Components in a Shard Cluster



Shard Keys

Shard keys are single or compound index keys that determine the distribution of data in a shard cluster.



Hash-Based Sharding

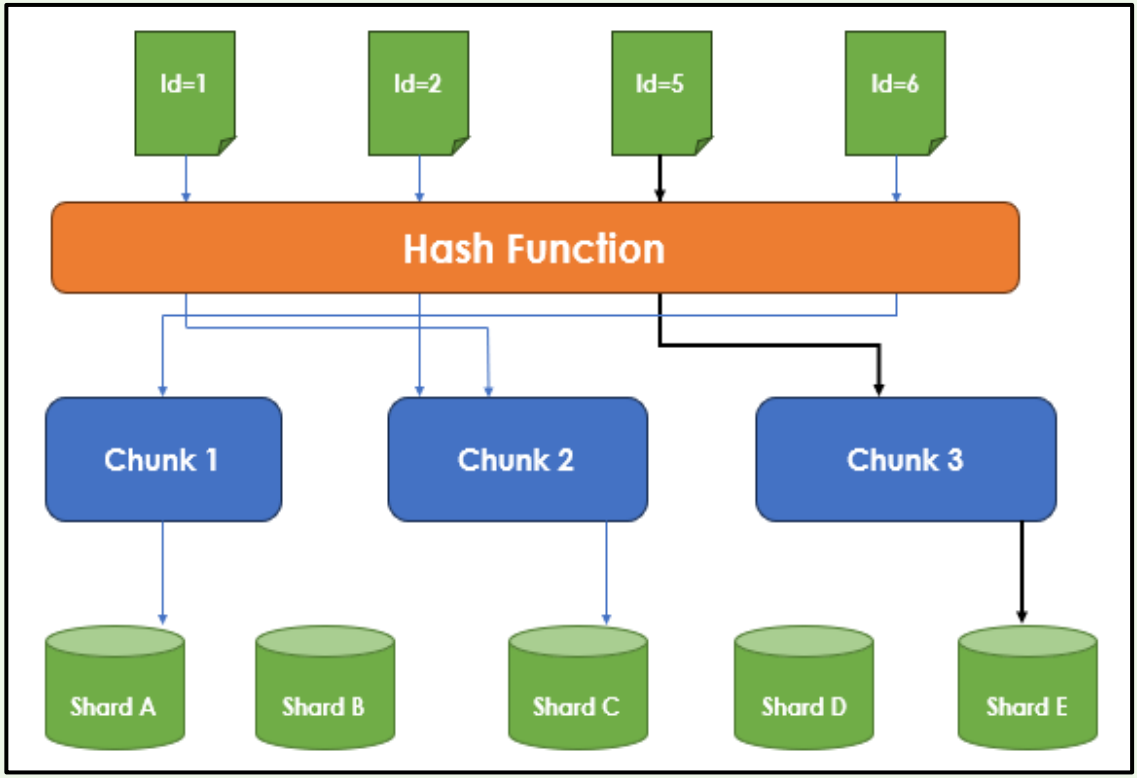
Hash values are computed based on shard keys.

Hash values determine how data is distributed among different shards.

Data is divided into chunks and stored in shards based on their hash values.

The default size of a chunk is 64 MB.

For every user query, MongoDB computes the hash value to locate the shard where the required data is stored.



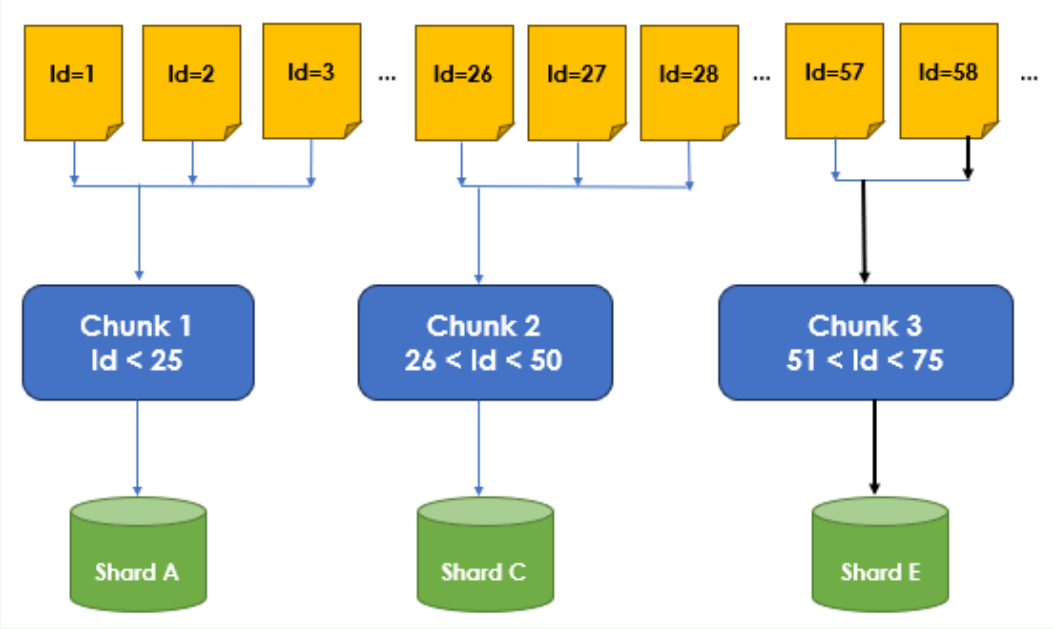
Range-Based Sharding

Indexes are arranged in an order and divided into chunks.

Each chunk represents a continuous range of indexes.

Chunks are evenly distributed across shards.

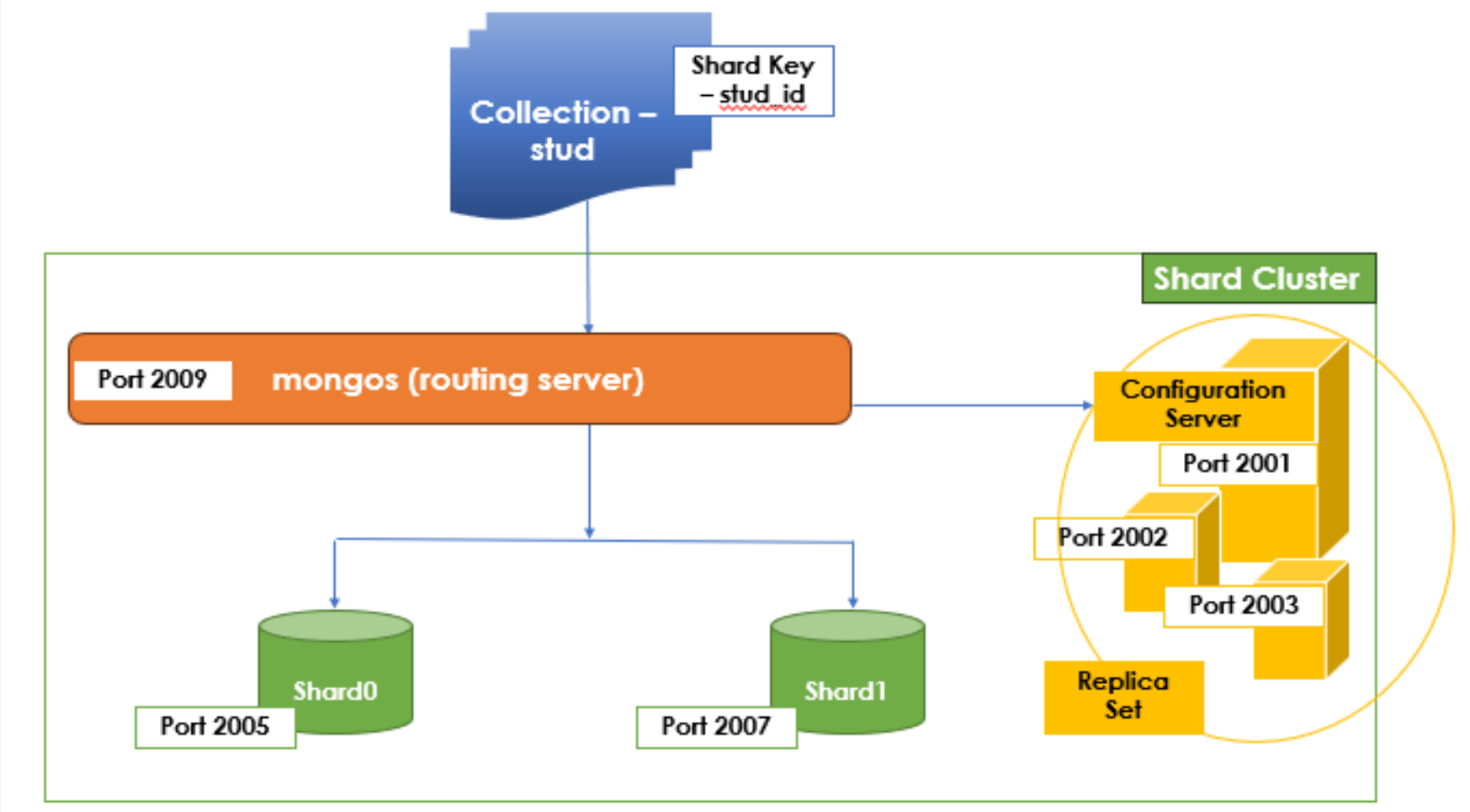
This strategy provides efficient querying as documents with closer shard keys are mostly placed in the same shard.



Sharding Methods

`sh.addShard()`	Adds a shard to the cluster
`sh.status()`	Reports on the status of a sharded cluster
`sh.enableSharding()`	Creates a database
`sh.shardCollection()`	Enables sharding for a collection
`sh.moveChunk()`	Migrates a chunk to a cluster

Sharding Implementation



Configuration Server Replica Set 1-2

Create three directories `cs1`, `cs2`, and `cs3`.

Create a `mongod` instance to act as a primary configuration server.

```
mongod --configsvr --replSet rshard --logpath
\data\cs1\1.log --dbpath \data\cs1 --port 2001
```

Start communication with `cs1` through port `2001`.

```
mongosh --port 2001
```

Configure `cs1` as primary shard server.

```
shardconfig={_id:"rshard",members:[{_id:0,host
:"localhost:2001"}]}
rs.initiate(shardconfig)
```

Configuration Server Replica Set 2-2

Configure the secondary configuration servers.

```
mongod --configsvr --replSet rshard --logpath
\data\cs2\2.log --dbpath \data\cs2 --port 2002

mongod --configsvr --replSet rshard --logpath
\data\cs3\3.log --dbpath \data\cs3 --port 2003
```

Add the secondary servers to the replica set.

```
rs.add("localhost:2002")
rs.add("localhost:2003")
```

Check the status of the servers.

```
rs.status()
```

Shard Servers 1-3

Create two folders `shard0` and `shard1`.

Create two log folders one each inside the folders `Shard0` and `Shard1`.

Create a shard server to listen to port `2005`.

```
mongod --shardsvr --replSet shardrep --
logpath \data\Shard0\log\Shard0.log --
dbpath \data\Shard0 --port 2005
```

Start communication with `shard0` through port `2005`.

```
mongosh --port 2005
```

Shard Servers 2-3

Configure `shard0` as the primary shard server.

```
sconfig={_id:"shardrep",members:[{_id:0,host:"localhost:2005"}]}

rs.initiate(sconfig)
```

Create a shard server to listen to port `2007`.

```
mongod --shardsvr --replSet shard0rep --logpath
\data\Shard1\log\Shard1.log --dbpath \data\Shard1 --port 2007

mongosh --port 2007

s2config={_id:"shard0rep",members:[{_id:0,host:"localhost:2007"}]}

rs.initiate(s2config)
```

Shard Servers 3-3

Start the routing server to listen to port 2009.

```
mongos --port 2009 --configdb
rshard/localhost:2001
```

Establish communication with the routing server.

```
mongosh --port 2009
```

Add the shard server to the routing server.

```
sh.addShard("shardrep/localhost:2005")

sh.addShard("shard0rep/localhost:2007")
```

Enable Sharding

Enable sharding for the `studentdb` database.

```
sh.enableSharding("studentdb")
```

Enable sharding for the `stud` collection in `studentdb` database.

```
sh.shardCollection("studentdb.stud",{"stud_id":"hashed"})
```

Insert records into the collection.

```
for (i = 1; i <= 100; i++) { db.stud.insertMany([{
stud_id: i, class: "Grade-5" }]);}
```

Check if the documents are sharded.

```
db.stud.getShardDistribution()
```

Summary

- Replication is making the data available in more than one place which enables easy querying and high data availability.
- PSS and PSA are some of the ways to implement replication.
- If the primary server fails, the secondaries hold an election, and the elected member takes the place of the primary server.
- Sharding is distributing data between shard servers and making them available for queries from the application.
- The query router helps in routing the data amongst the shards.
- Maintaining the shard servers as replica sets helps in ensuring the high availability of data.

Session: 9

Transaction Management

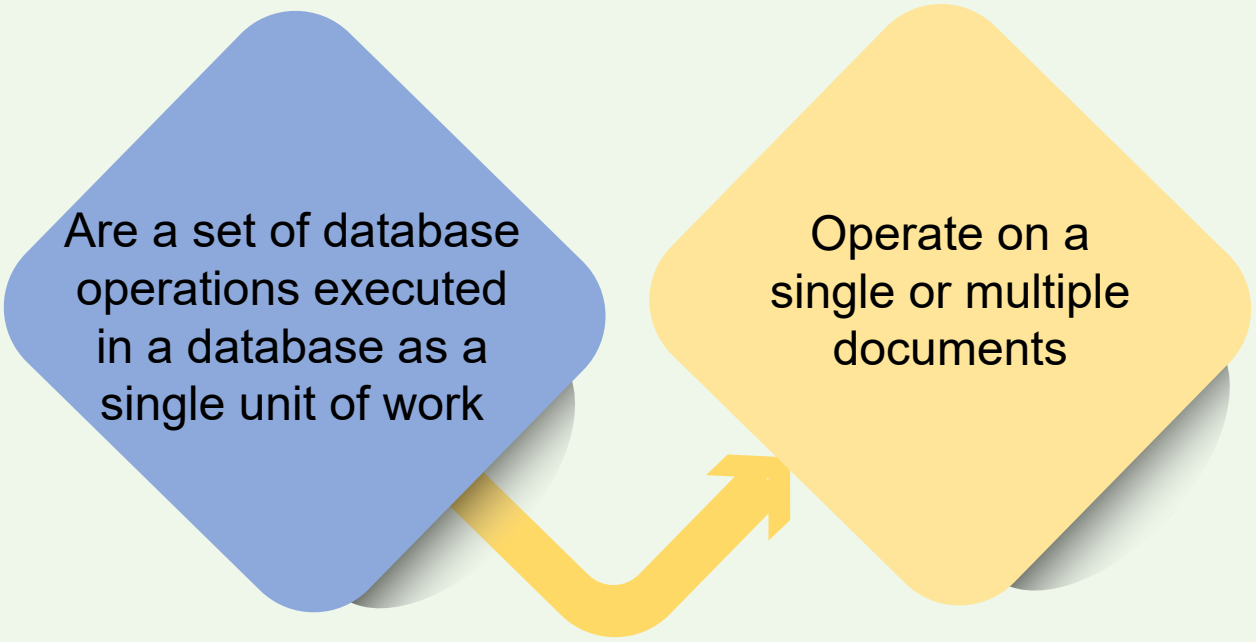
Managing Large Datasets Using MongoDB

Objectives

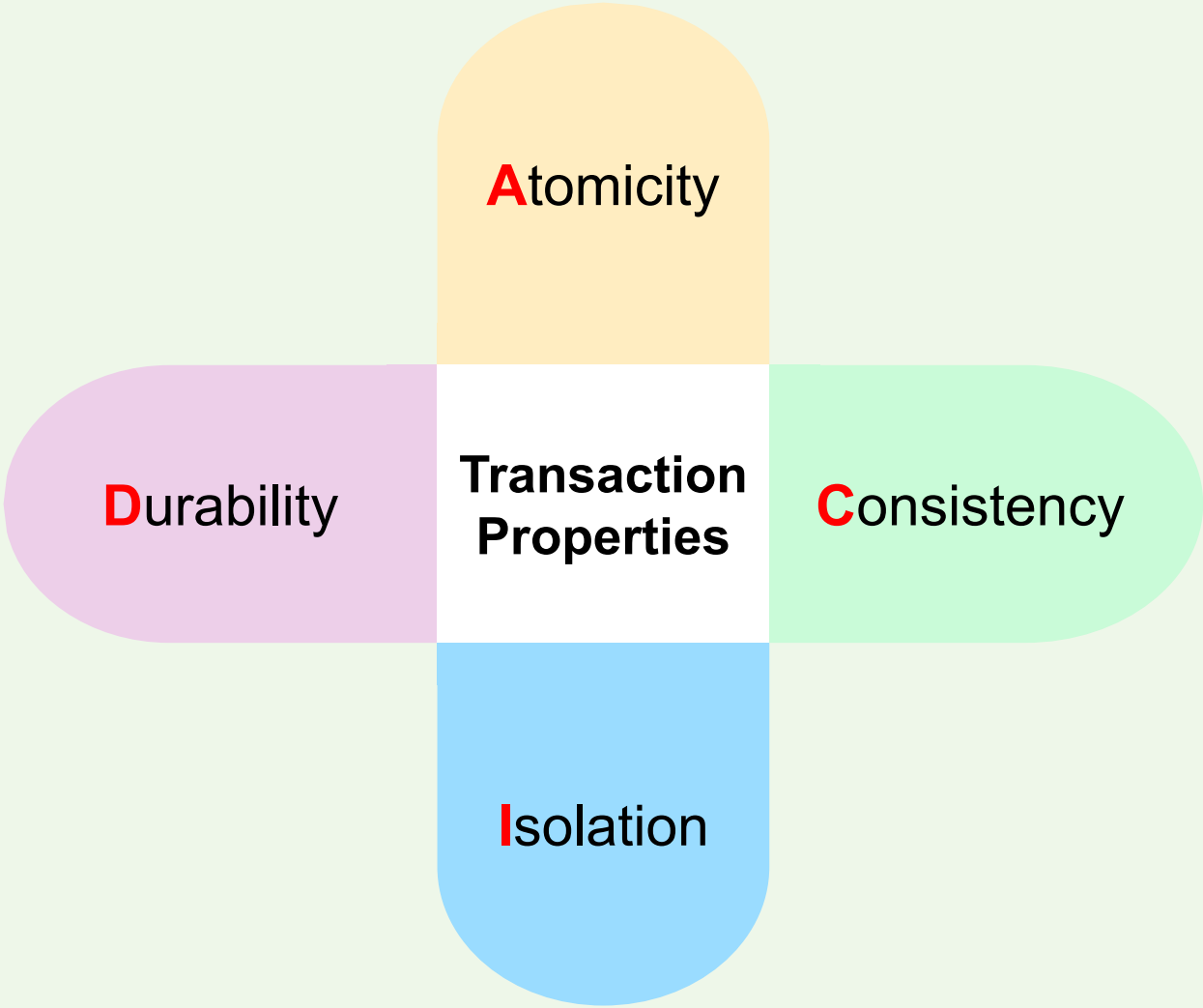
- Explain transactions in MongoDB
- Describe the transaction Application Programming Interfaces (API) in MongoDB
- Explain various error labels in MongoDB
- Describe properties of transactions in MongoDB
- Explain sessions and transactions within the sessions in MongoDB

Introduction to MongoDB Transactions

Transactions:

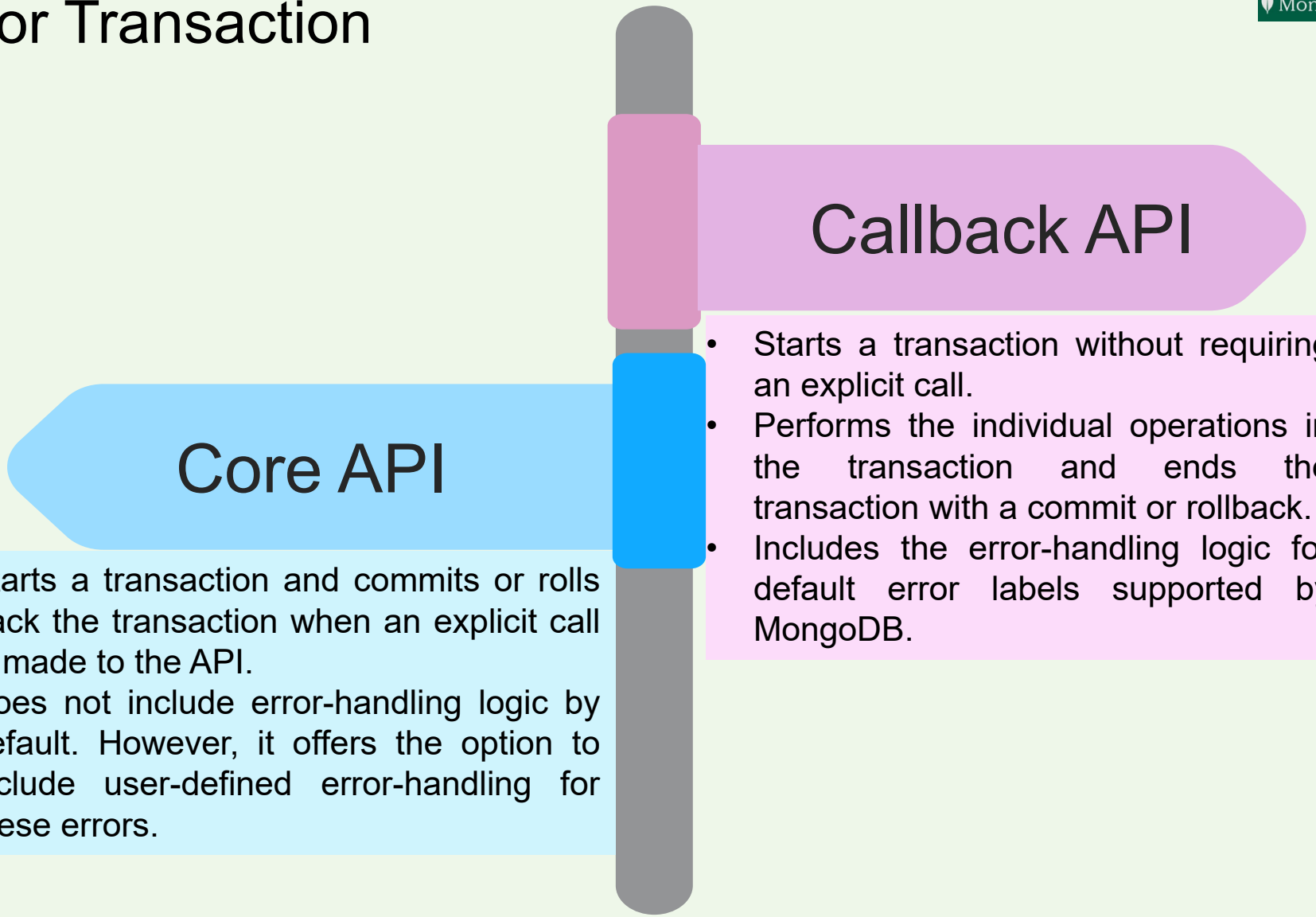


Properties of MongoDB Transactions



Use of Replica Sets in Transactions

Application Programming Interfaces (APIs) for Transaction



- Starts a transaction without requiring an explicit call.
- Performs the individual operations in the transaction and ends the transaction with a commit or rollback.
- Includes the error-handling logic for default error labels supported by MongoDB.

- Starts a transaction and commits or rolls back the transaction when an explicit call is made to the API.
- Does not include error-handling logic by default. However, it offers the option to include user-defined error-handling for these errors.

Transaction Error Handling

Capturing errors and specifying actions on the errors

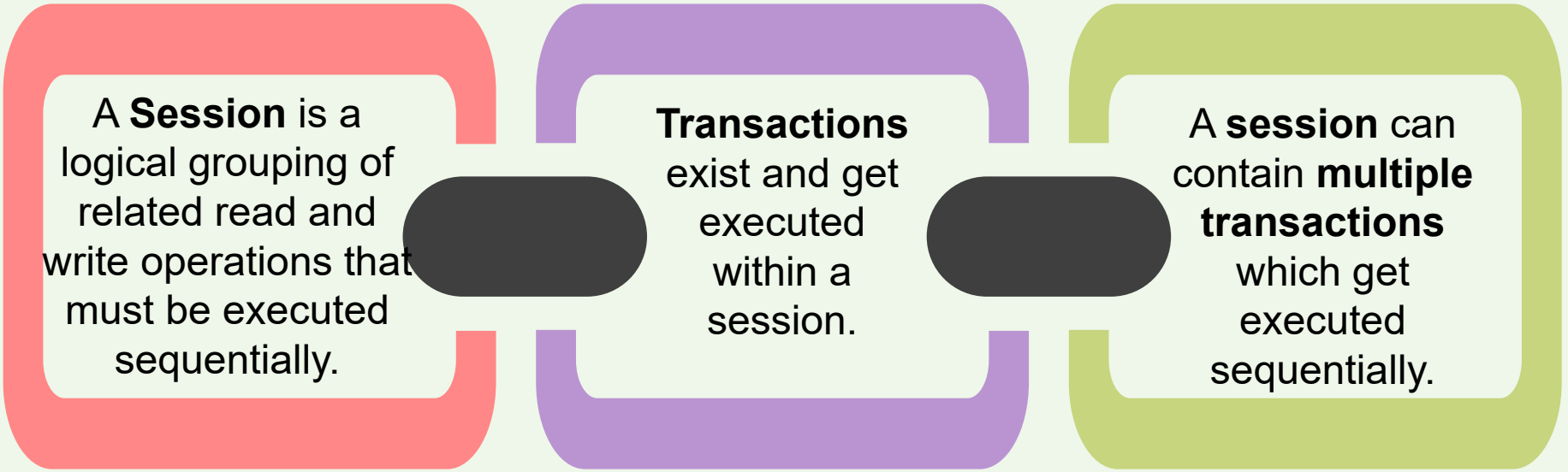
`TransientTransactionError`

Indicates that a temporary error has occurred in the transaction and execution of the transaction can be retried for a successful completion

`UnknownTransactionCommitResult`

Indicates that an error has occurred while committing the transaction and the driver can retry the commit operation

Transactions and Sessions



Methods to Manage Transactions

`startTransaction`

Start a transaction within a session

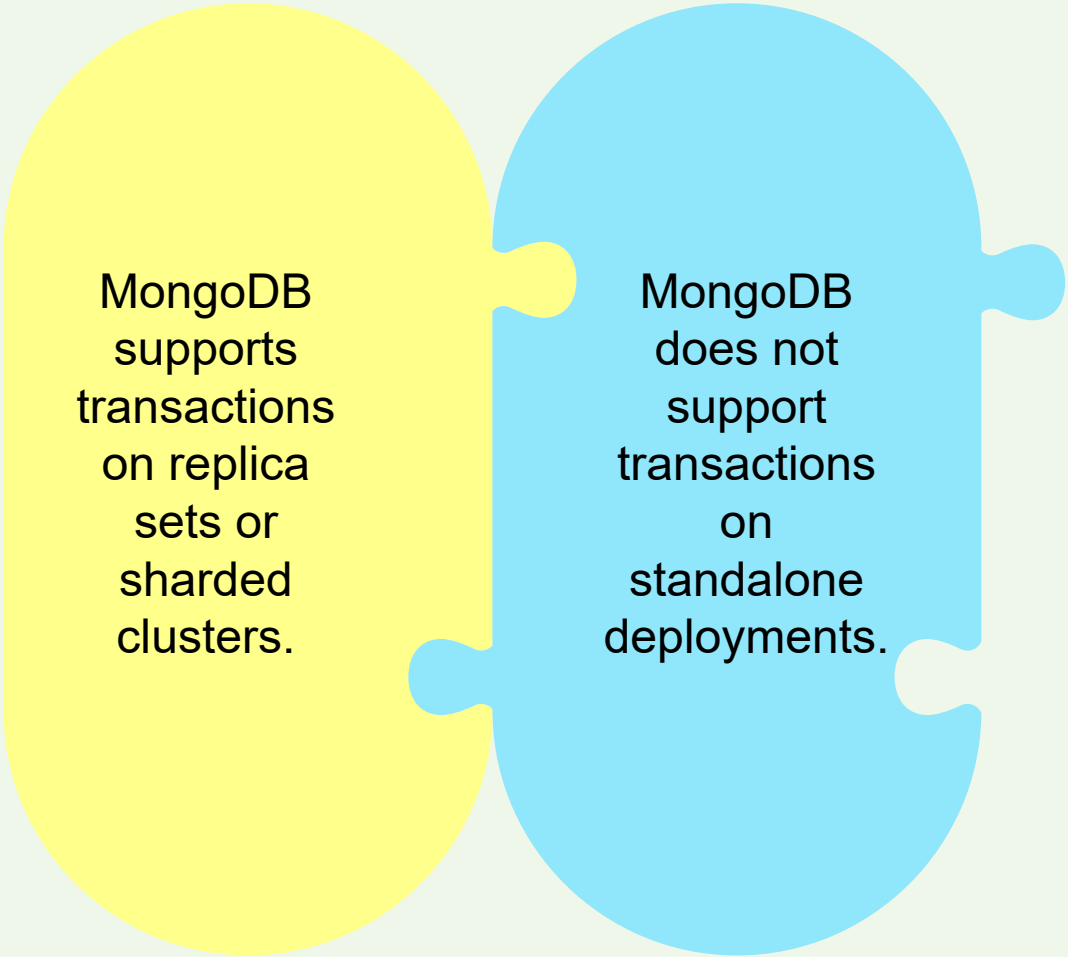
`commitTransaction`

Permanently save the changes made by the operations in a transaction

`abortTransaction`

Rollback the changes made by the operations in a transaction

Requirements for Transactions



Working with Transactions 1-2

Documents in the `EmpDetail` Collection

```
replicaset [direct: primary] Employee> db.EmpDetail.find()
[
  {
    _id: ObjectId("647778437e0714d84f821a47"),
    Emp_ID: 101,
    Emp_Name: 'Oliver Smith',
    Gender: 'Male',
    Designation: 'Software Engineer'
  },
  {
    _id: ObjectId("647778437e0714d84f821a48"),
    Emp_ID: 103,
    Emp_Name: 'Richard Franklin',
    Gender: 'Male',
    Designation: 'HR Manager'
  },
  {
    _id: ObjectId("647778437e0714d84f821a49"),
    Emp_ID: 105,
    Emp_Name: 'Linda Michael',
    Gender: 'Female',
    Designation: 'Accountant'
  }
]
```

Working with Transactions 2-2

The user runs commands to:

Avoid duplicates in the `Emp_ID` field

```
db.EmpDetail.createIndex( { "Emp_ID": 1 },
                          { "unique": true } )
```

Allow read operation on the secondary nodes

```
rs.secondaryOk()
```

Verify that the documents reflect in the secondary node connected at port `27019`

```
use Employee
db.EmpDetail.find()
```

Transactions in a Session

Create a Transaction Within a Session and Commit the Transaction

Create a Transaction Within a Session and Abort the Transaction

Create a Multi-Document Transaction Within a Session

Create a Transaction Within a Session and Commit the Transaction

Start a session named `session`.

```
var session = db.getMongo().startSession()
```

Start a transaction.

```
session.startTransaction({"readConcern": { "level":"snapshot" },
writeConcern": { "w": "majority" }})
```

Create a variable that represents `EmpDetail` within the transaction.

```
var empdetail=session.getDatabase('Employee').getCollection('EmpDetail')
```

Insert a new document into `EmpDetail`.

```
empdetail.insertOne({Emp_ID:102, Emp_Name:"John Edward", Gender:"Male",
Designation:"Software Engineer"})
```

Commit the transaction.

```
session.commitTransaction()
```

Create a Transaction Within a Session and Abort the Transaction

Update the designation of the employee.	`empdetail.updateOne({Emp_ID:105}, {$set:{Designation:"Project Lead"}})`
Delete a document.	`empdetail.deleteOne ({Emp_ID:103})`
Abort the transaction.	`session.abortTransaction()`

Create a Multi-document Transaction Within a Session 1-2

Documents in the `DesignationDetail` Collection

```
[
  {
    _id: ObjectId("648ff438c48f7416b9f41ba0"),
    Designation_ID: 10003,
    Designation_Title: 'Accountant',
    Department: 'Finance'
  },
  {
    _id: ObjectId("648ff438c48f7416b9f41b9e"),
    Designation_ID: 10002,
    Designation_Title: 'HR Manager',
    Department: 'Human Resources'
  },
  {
    _id: ObjectId("648ff438c48f7416b9f41b9c"),
    Designation_ID: 10001,
    Designation_Title: 'Software Engineer',
    Department: 'Product Development'
  },
  {
    _id: ObjectId("648ff438c48f7416b9f41b9f"),
    Designation_ID: 10012,
    Designation_Title: 'HR Executive',
    Department: 'Human Resources'
  },
  {
    _id: ObjectId("648ff438c48f7416b9f41b9d"),
    Designation_ID: 10011,
    Designation_Title: 'Project Manager',
    Department: 'Product Development'
  },
  {
    _id: ObjectId("648ff438c48f7416b9f41ba1"),
    Designation_ID: 10013,
    Designation_Title: 'Finance Manager',
    Department: 'Finance'
  }
]
```

Create a Multi-document Transaction Within a Session 2-2

Create a variable that represents the `DesignationDetail` collection.

```
var designation =
session.getDatabase('Employee').getCo
llection('DesignationDetail')
```

Update the Designation field of the employee with `Emp_ID` as `105` to `Finance Executive`.

```
empdetail.updateOne({Emp_ID:105},
{$set:{Designation:"Finance
Executive"}})
```

Update the `Designation_Title` with `Designation_ID` as `10003` to `Finance Executive`.

```
designation.updateOne({Designation_ID
:10003},{$set:{Designation_Title:
"Finance Executive"}})
```

Commit the transaction.

```
session.commitTransaction()
```

Summary

- Transactions in MongoDB are a single logical unit of work that involves single or multiple operations.
- The integrity of data in the database is protected by the four properties of transactions—Atomicity, Consistency, Isolation, and Durability (ACID).
- MongoDB provides two transaction APIs—Callback API and Core API.
- MongoDB provides methods to create, commit, and abort transactions in a session.
- Two major error labels in MongoDB are `TransientTransactionError` and `UnknownTransactionCommitResult`.
- MongoDB sessions can contain multiple transactions. Only one of these transactions remains active at a time.

Session: 10

Working With MongoDB Tools

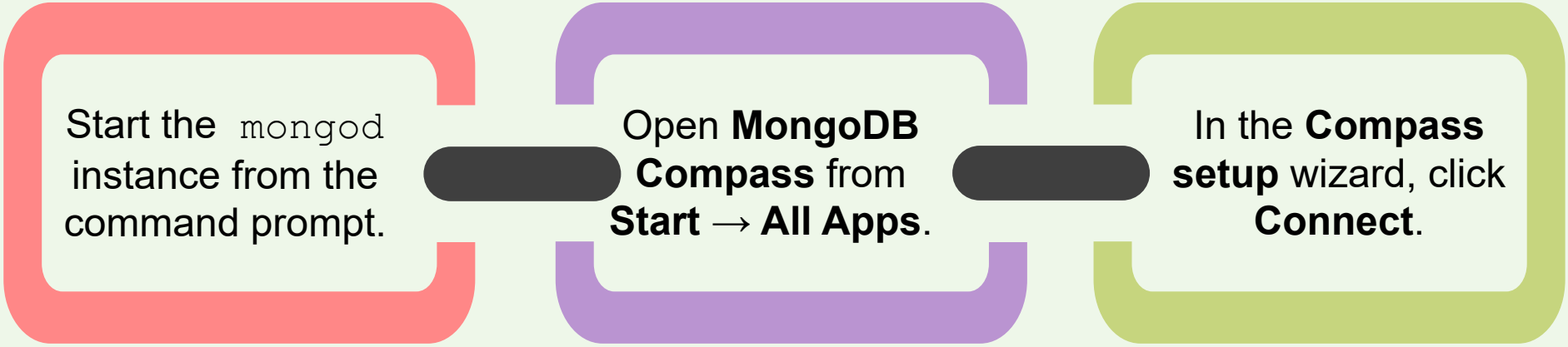
Managing Large Datasets Using MongoDB

Objectives

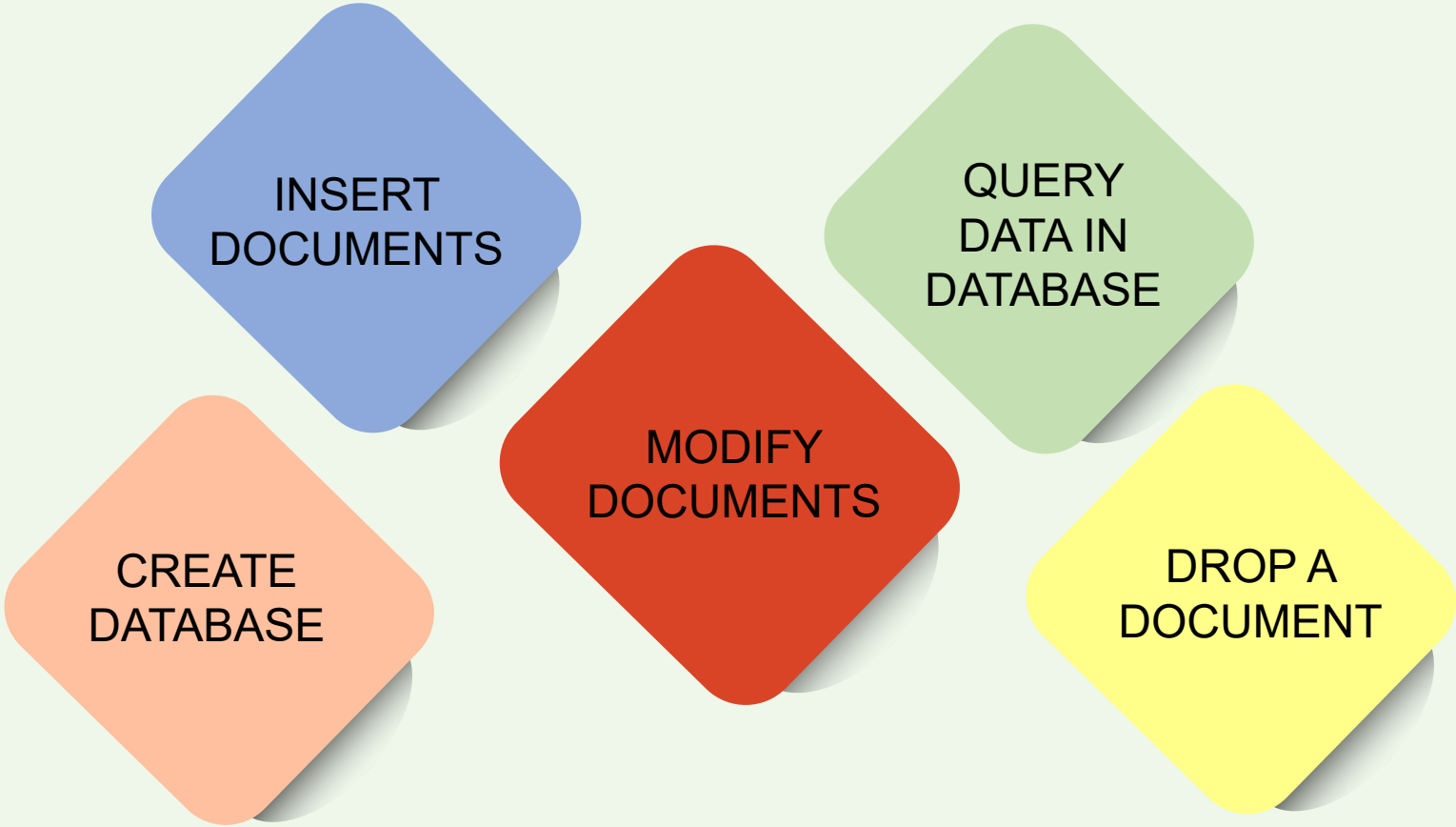
- Explain the method to connect MongoDB Compass with MongoDB Deployment
- Describe how to use MongoDB Compass to create, insert, update, select, and drop data in a database
- Explain the installation of MongoDB Business Intelligence (BI) Connector
- Describe the process for integrating the MongoDB BI Connector with the MongoDB Deployment
- Describe how to create a system Data Source Name (DSN)

Connect MongoDB Compass with MongoDB Deployment

To connect **Compass** with a `mongod` instance:



Manage Database Using Compass

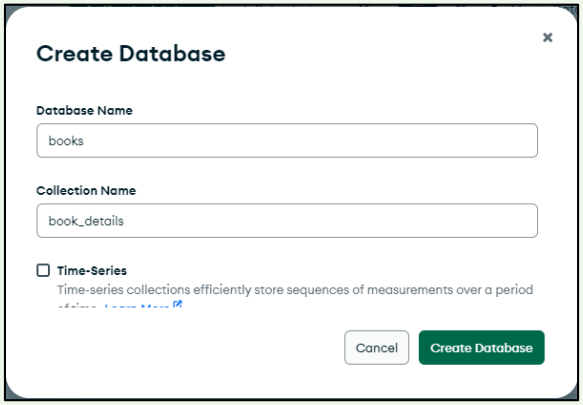


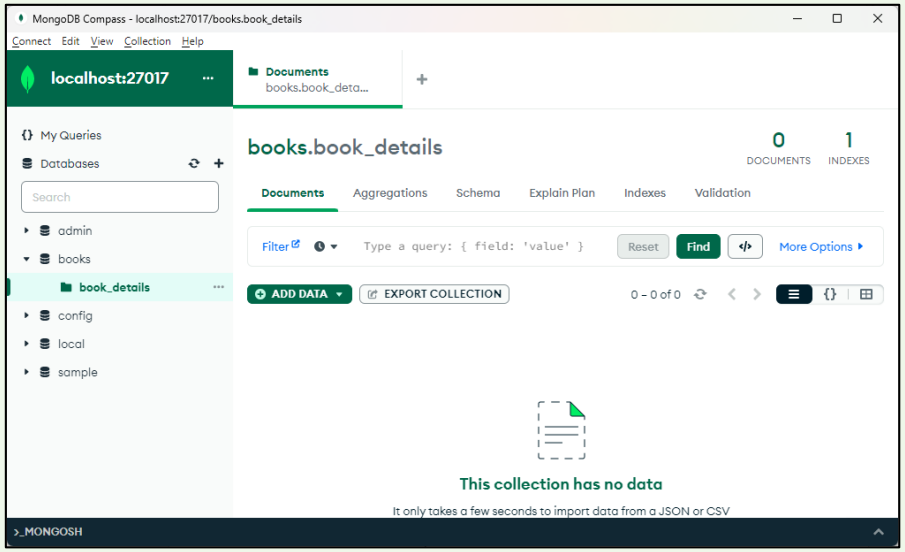
Create Database Using Compass

In **MongoDB Compass** home page, click the **Databases** tab.

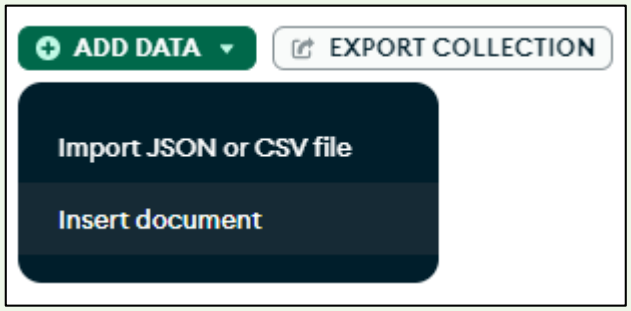
On the **Database** tab, click **Create Database**.

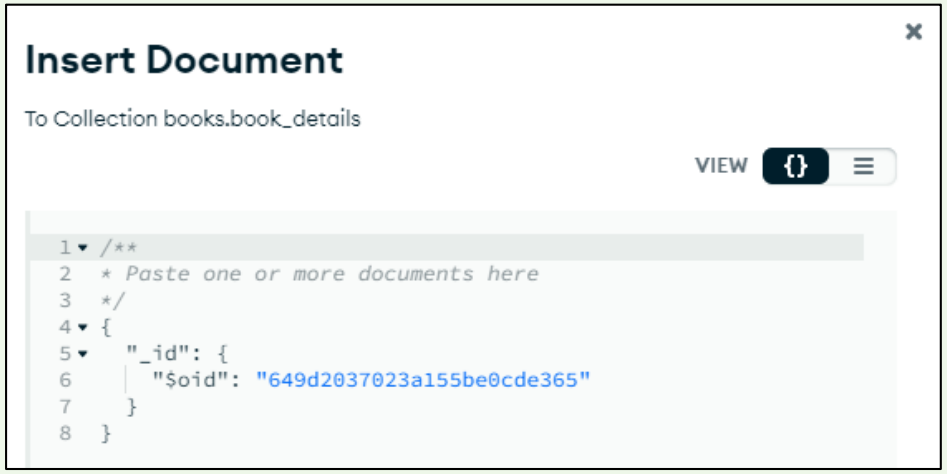
Type the database name as `books` and the collection name as `book_details`.





Insert Documents Using Compass

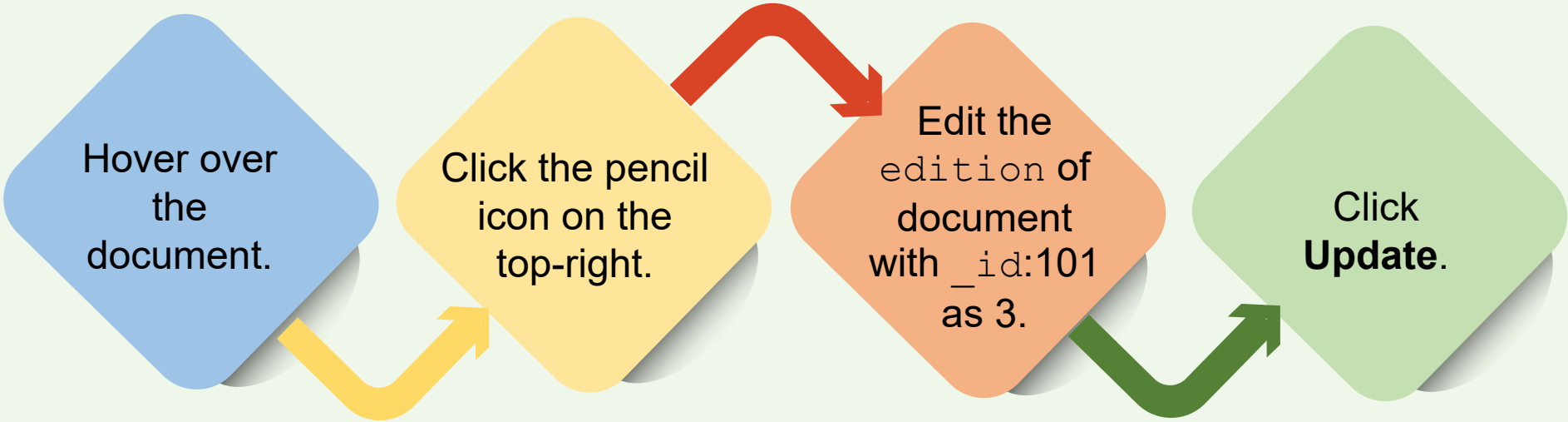




To import JSON files, select the { } on the top-right is and type the document(s) you want to insert into the collection.

To insert multiple documents, enter a comma-separated array of JSON documents in the { }.

Modify Documents Using Compass



Query Data Using Compass

In the **Filter** box, enter the name of the field, to be fetched, along with its value.

Click **Find** to run the query.

Using the **More Options** button, you can:
1. Project the fields in the document.
2. Sort the documents.
3. Skip the number of documents.
4. Limit the number of documents.

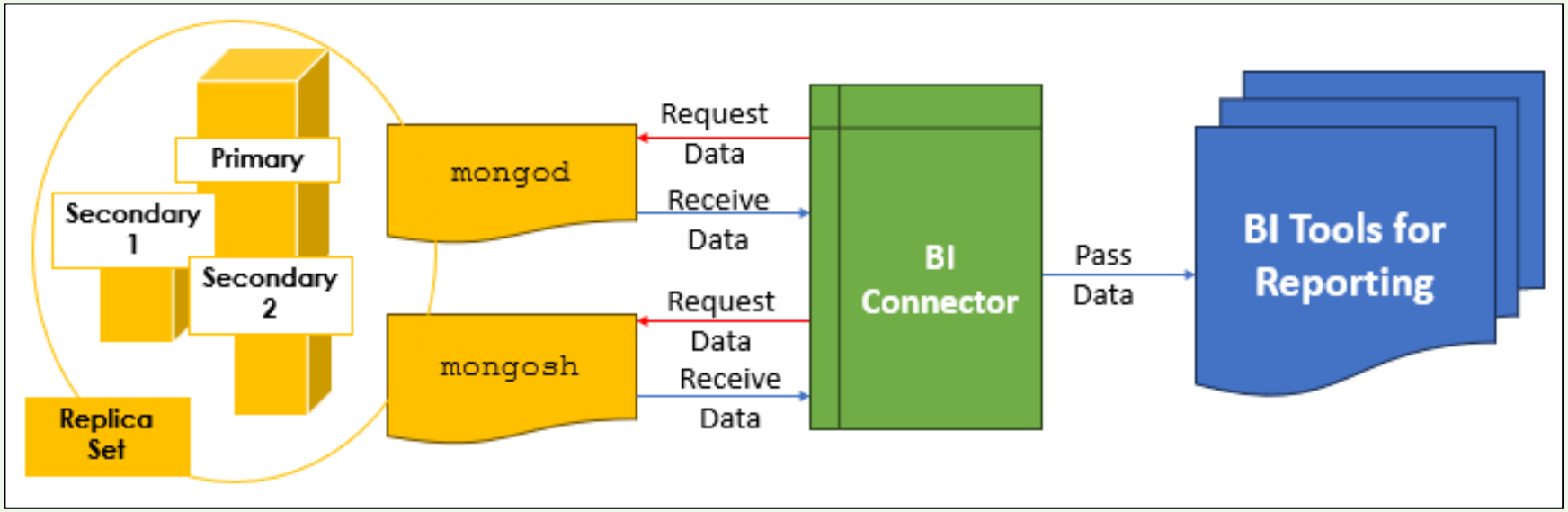
Drop a Document Using Compass

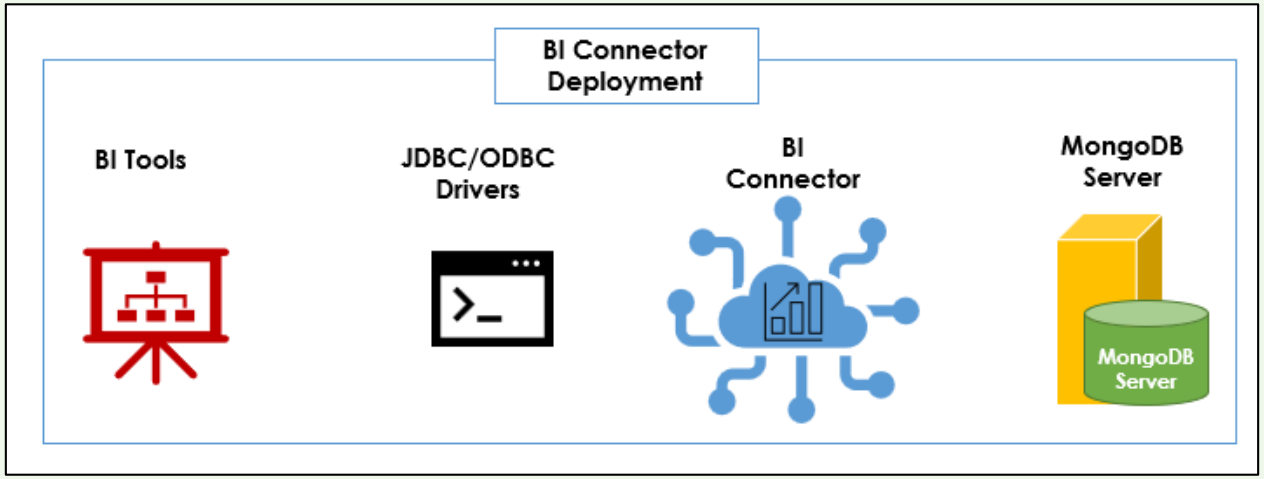
01 Hover the cursor over the document.

02 Click the **Remove** icon.

03 Click **Delete**.

MongoDB BI Connector and its System Components



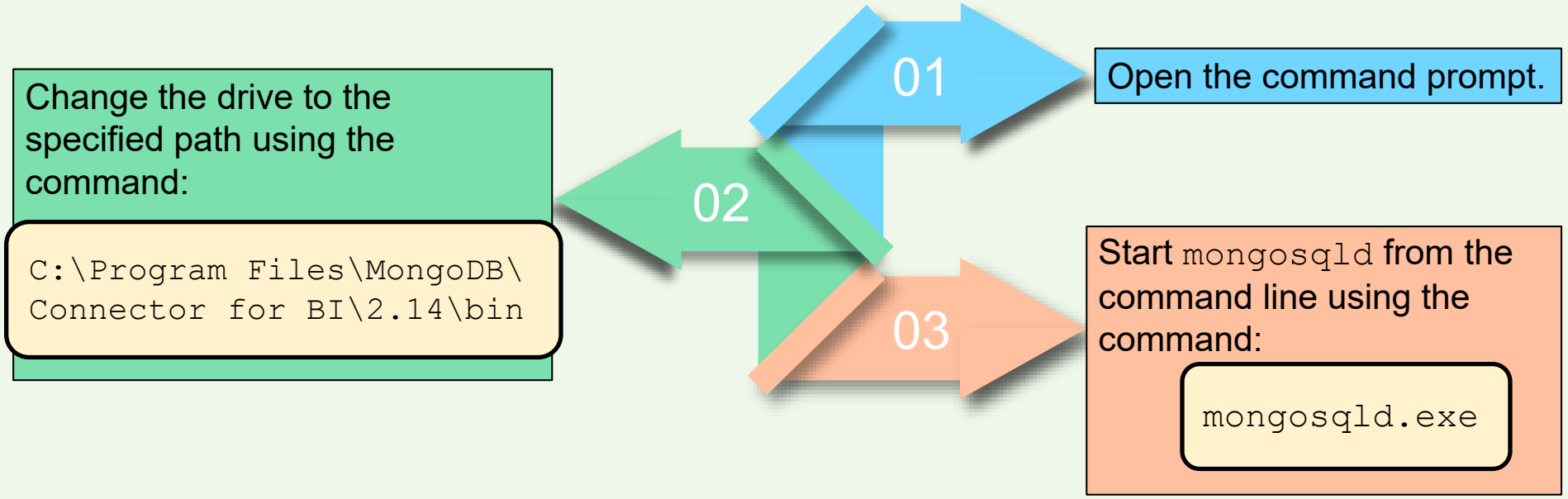


How to Install MongoDB BI Connector?

1. Open the browser and visit https://www.mongodb.com/download-center/bi-connector/releases
2. On this page, under **2.14.7**, scroll down to the **Windows x64** section and download the file mongodb-bi-win32-x86_64-v2.14.7.msi
3. After the download is complete, run the installer.
   The **MongoDB BI Connector Setup** wizard opens.
4. Click **Next**. The **End-User License Agreement** page opens.
   Select **I accept the terms in the License Agreement** check box.
   Click **Next**. The **Custom Setup** page opens. Click **Next.**
5. The **Ready to Install Connector for BI** page opens. Click **Install**.
   The **Completed the Connector for BI** page opens. Click **Finish**.

Connect MongoDB BI Connector with MongoDB Deployment

To launch `mongosqld`:



Connect MongoDB Database to BI Tools Using Open Database Connectivity (ODBC)

To install MongoDB BI Connector ODBC Driver:

1. Open the browser and navigate to the Website, https://github.com/mongodb/mongo-bi-connector-odbc-driver/releases/
To download the **MongoDB BI Connector ODBC Driver**, scroll down and select mongodb-connector-odbc-1.4.3-win-64-bit.msi

2. Run the installer after the download is complete. The **MongoDB ODBC 1.4.3 Setup** wizard opens.
.

3. Click **Next**. The **End-User License Agreement** page opens. Select the **I accept the terms in the License Agreement** check box.

4. Click **Next**. The **Custom Setup** page opens.

5. Click **Next**. The **Ready to Install MongoDB ODBC** page opens.

6. Click **Install**. The **Completed MongoDB ODBC** page opens.

7. Click **Finish**. Now, **MongoDB BI Connector ODBC driver** is installed.

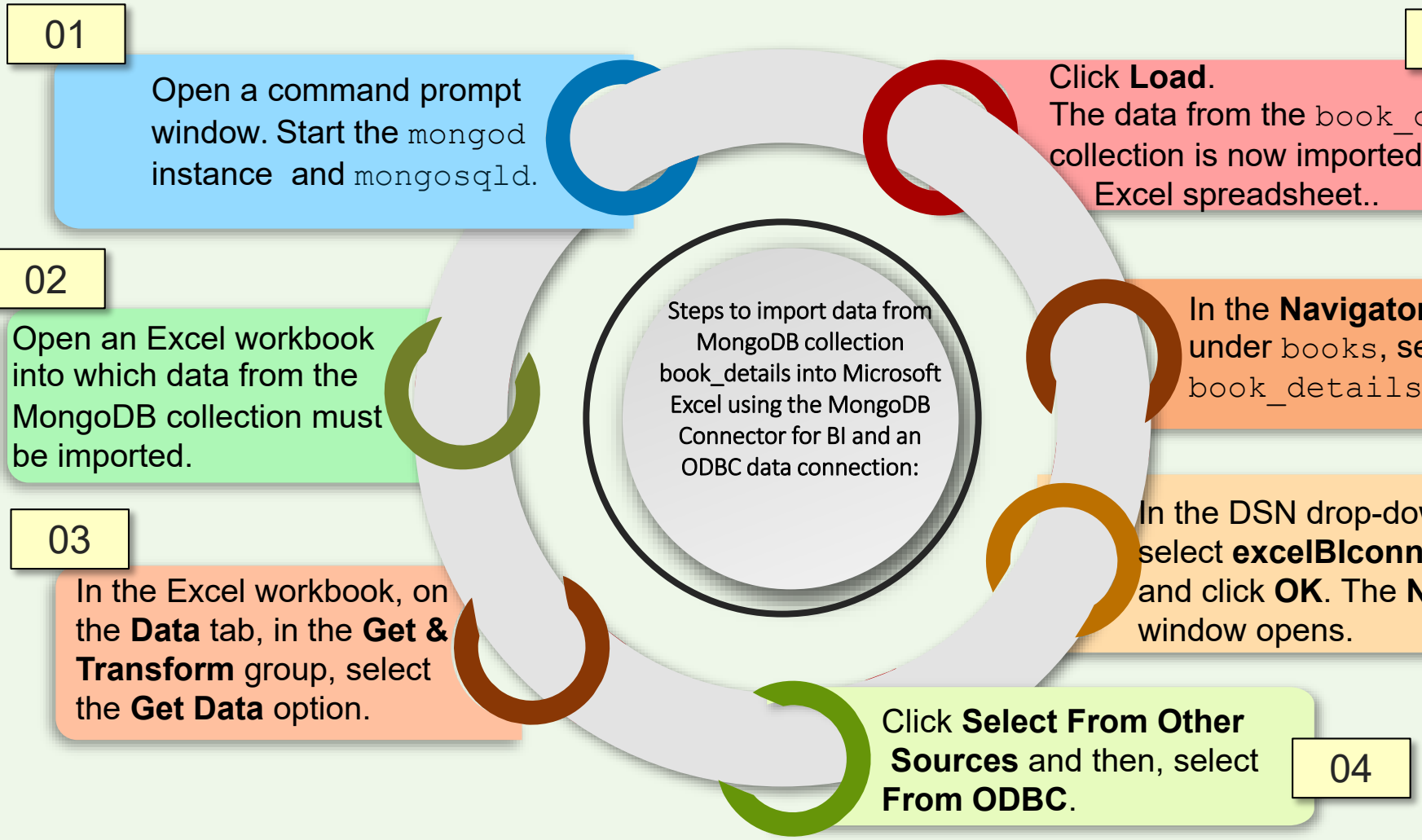
Create a Data Source Name (DSN)

01 Open a command prompt window.
Start the `mongod` instance and `mongosqld`.
Open the **Control Panel** window and select the **System and Security** option.
Select the **Windows Tools** option.

02 In the **ODBC Data Source Administrator (64 bit)** wizard, click the **System DSN** tab and click the **Add** button.
In the **Create New Data Source** window that opens, select either the **MongoDB ODBC ANSI Driver** or the **MongoDB ODBC Unicode Driver** and click **Finish.**
The **MongoDB Data Source Configuration** dialog box page opens.

03 In the dialog box, enter the name of the data source, specify the port to connect to the data source, and select the database to connect to. Click **Test.**
In the **Test Results** message box that appears, click **OK.**

Import Data from MongoDB to Microsoft Excel



Summary

- MongoDB Compass is a visual interactive tool that helps users to manage documents in the database efficiently.
- Users can connect to Compass from a `mongod` instance, any of the replica set, or any of the sharded cluster servers.
- Relational business intelligence tools such as Tableau and Power BI can be used to envision, chart, and review three-dimensional MongoDB data by connecting them with the help of the MongoDB BI Connector.
- A `mongod` instance and BI tool are connected by the BI Connector program, `mongosqld`. MongoDB collections and databases must be mapped to a data schema by `mongosqld`.
- The MongoDB BI Connector ODBC driver allows SQL clients to connect to MongoDB Connector for BI.

Session: 11

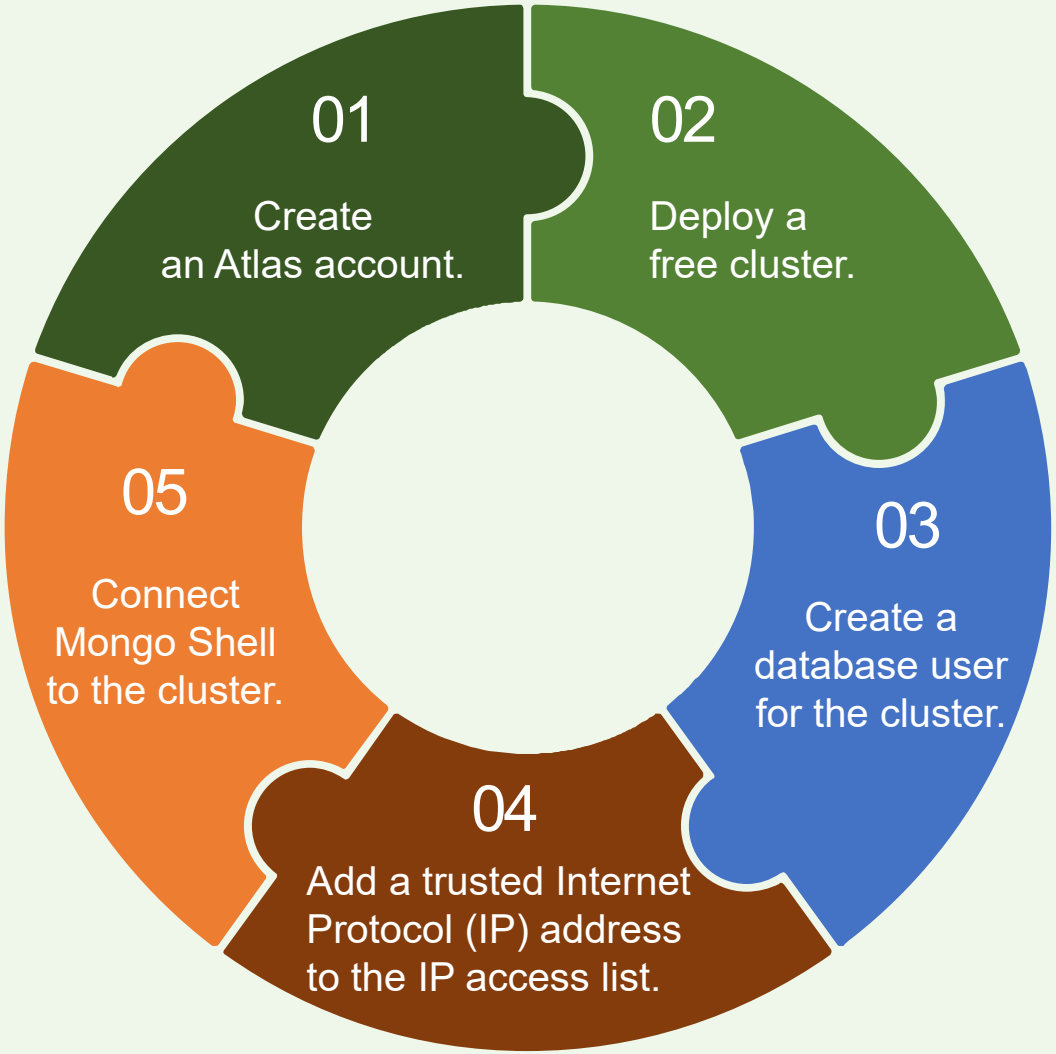
Managing Large Datasets Using MongoDB

MongoDB Cloud

Objectives

- Explain how to create an Atlas account and set up a cluster
- Describe how to access the MongoDB Atlas cluster
- Explain how to import data into the Atlas cluster from a MongoDB Instance and manage the imported data
- Explain how to export data from the Atlas cluster into a MongoDB Instance and manage the exported data
- Describe how to perform administrative tasks in the MongoDB cluster

Get Started with MongoDB Atlas 1-6



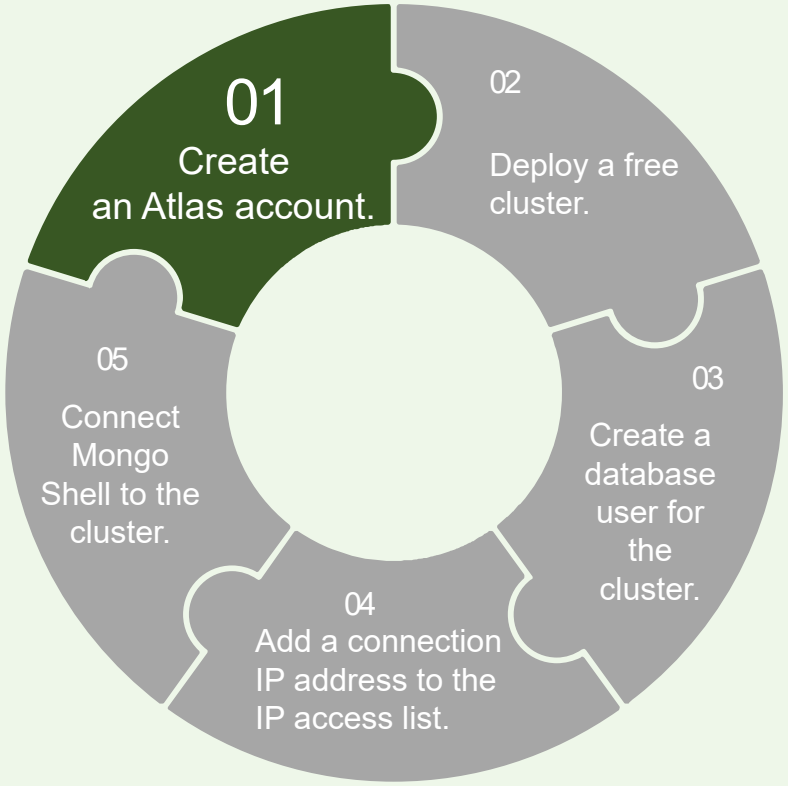
Get Started with MongoDB Atlas 2-6

Create an Atlas account.

Register for an Atlas account using any one of these:

- GitHub account
- Google account
- Email address

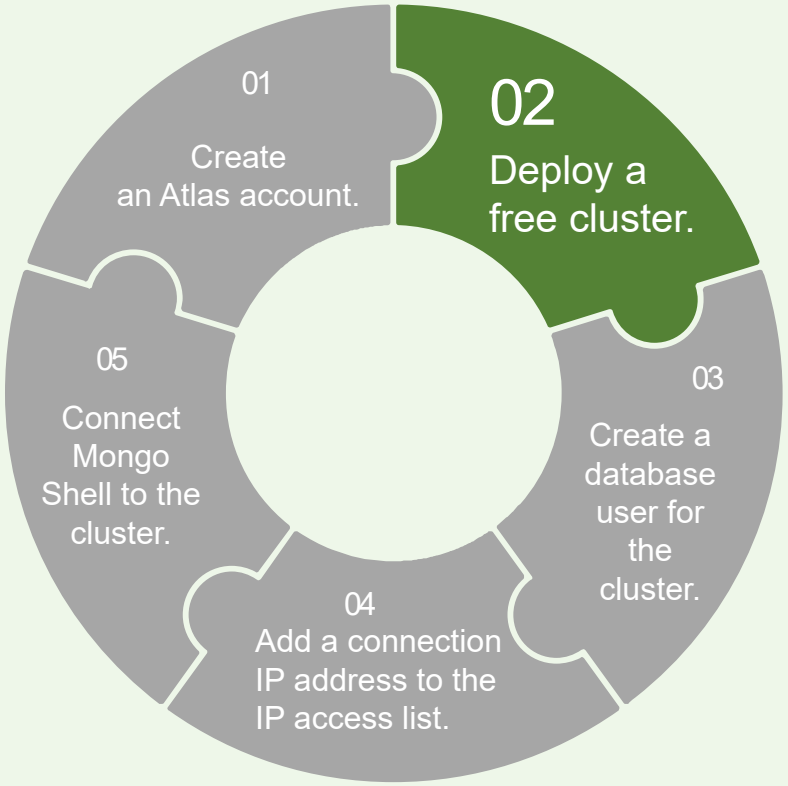
1. Navigate to https://www.mongodb.com/cloud/atlas/register
2. Click **Sign up with Google**, enter your Gmail ID and the password.
3. Review and accept the privacy policy and terms of service by selecting the **I accept the Privacy Policy and Terms of Service** check box.Click **Submit**.



Get Started with MongoDB Atlas 3-6

Deploy a free cluster.

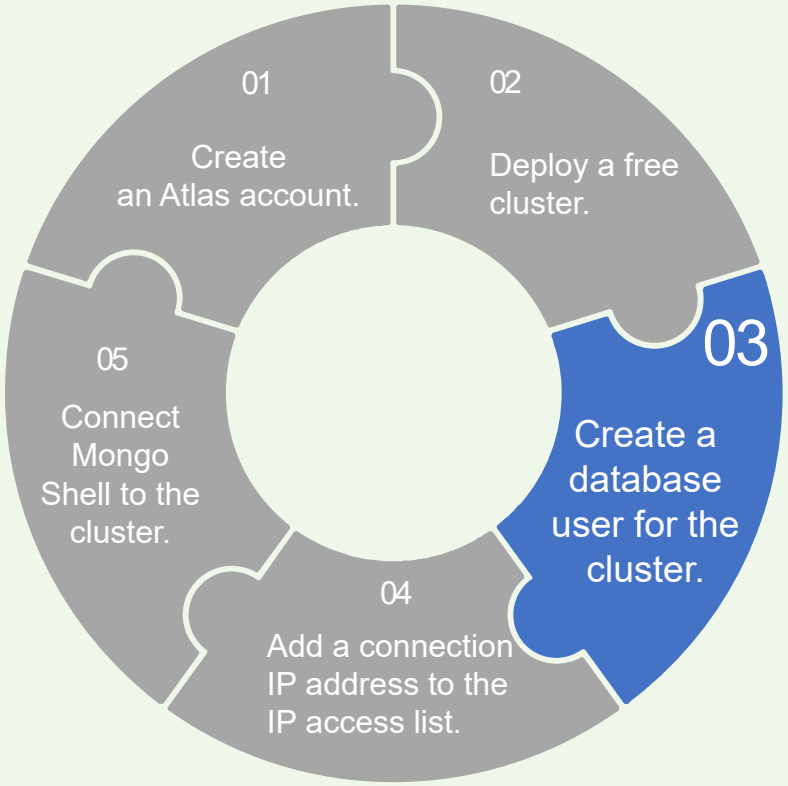
1. In the **Deploy your database** page, select the **M0** option.
2. Click **Build a Database**.
3. Select the **M0** option, choose the preferred:
   - Cloud provider from the available options (AWS, GCP, or Azure)
   - Region.
4. Specify a name in the **Name** box (using American Standard Code for Information Interchange (ASCII) letters, numbers, and hyphens).
5. Click **Create**.



Get Started with MongoDB Atlas 4-6

Create a database user for the cluster.

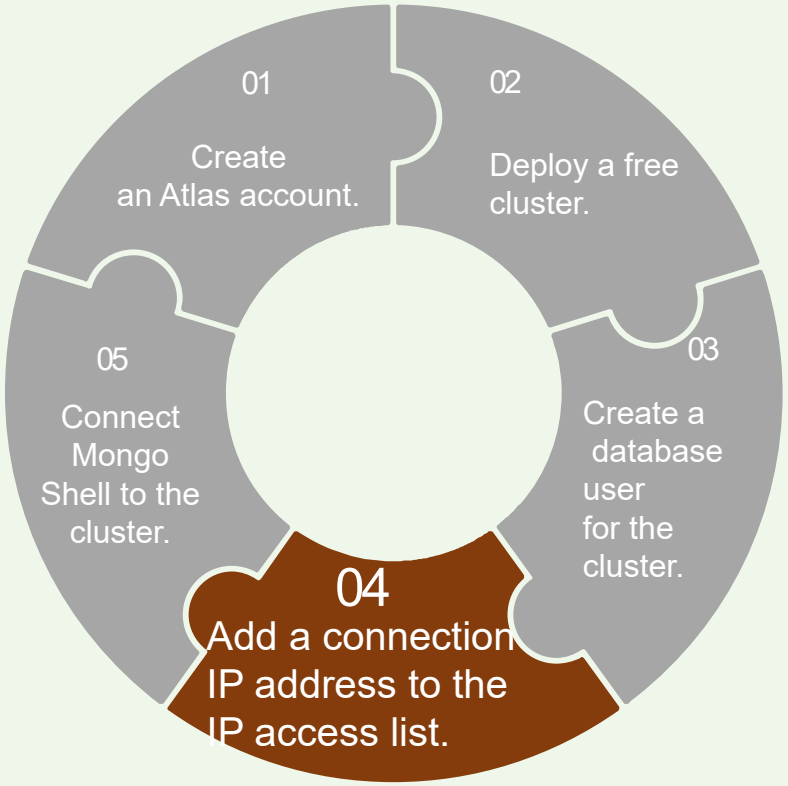
1. In the **Security Quickstart** page, enter the new username and credentials.
2. To use a password auto-generated by Atlas, click **Autogenerate Secure Password**.
3. Click **Create User**.



Get Started with MongoDB Atlas 5-6

Add a connection IP address to the IP access list.

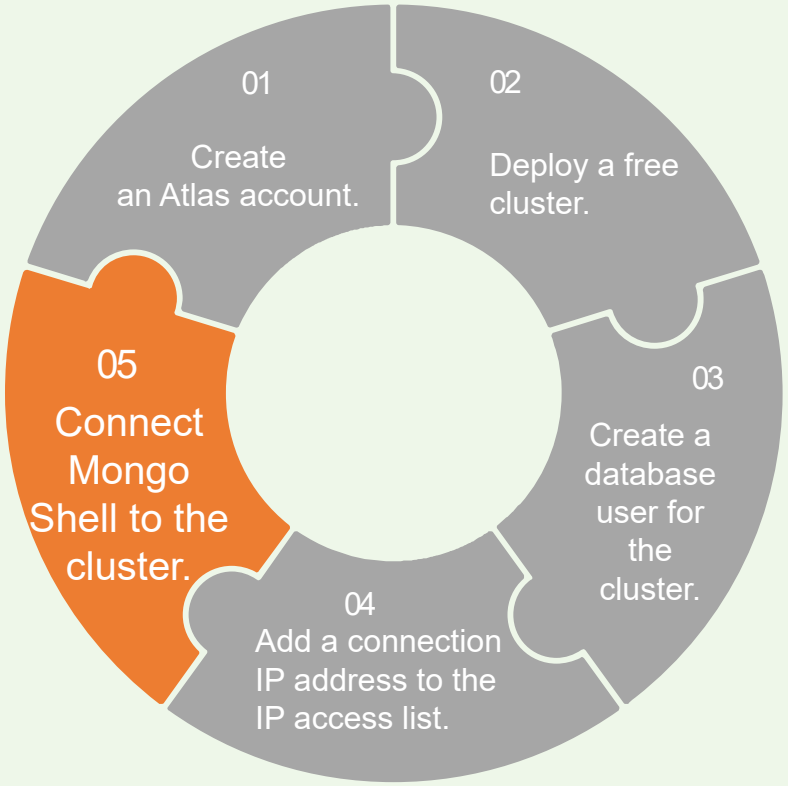
1. To add the current IP address, click **Add My Current IP Address**.
2. Click **Finish and Close**.



Get Started with MongoDB Atlas 6-6

Connect Mongo Shell to the cluster.

1. Navigate to **Database Deployments** page.
2. Click **Connect**.
3. Click **Choose a connection method**.
4. Under Access your data through tools section, click **Shell**.Click **I have the MongoDB Shell installed** and **s**elect the `mongosh` version.
5. Copy and run the unique connection string in a command prompt. Enter your database user's password when prompted.



Access MongoDB Atlas Cluster

To load sample data

Navigate to the **Database Deployments** page.

Click **Load Sample Dataset**.

To view the loaded sample data

In Atlas UI, click **Browse Collections**.

In Mongo Shell, open a command prompt and execute `show dbs`.

Data Import and Export

```
mongoimport --uri
mongodb+srv://<USERNAME>:<PASSWORD>@cluster0.qbq5rtw.mongodb.net/<DATABASE>
--collection <COLLECTION> --type <FILETYPE> --file <FILENAME>
```

```
mongoexport --uri
mongodb+srv://thea:<PASSWORD>@cluster0.qbq5rtw.mongodb.net/<DATABASE> --
collection <COLLECTION> --type <FILETYPE> --out <FILENAME>
```

Placeholder	Values
<PASSWORD>	Replace with the password for the user; You can include the database user for the project using the --username switch. To authenticate as a different user, replace the value of --username and specify the password for that user in --password.
<DATABASE>	Replace with the name of the database to which data is being imported or exported.
<COLLECTION>	Replace with the name of the collection to which data is being imported or exported.
<FILETYPE>	Replace with the file type of the data source from which data is being imported or exported.
<FILENAME >	Replace with the name of the data source from which data is being imported or exported.

Steps to Import and Export Data

To import data from the local system to MongoDB Atlas or export Data from Atlas Cluster to the local system, navigate to the **Database Deployments** page and choose **Command Line Tools** for the desired cluster.

Under **Data Import and Export Tools** section, copy the `mongoimport` or the `mongoexport` connection string.

Edit and run the connection strings from the command prompt and NOT the Mongo shell.

Administering MongoDB Cluster

Pause	Resume
• Depending on the cluster tier, Atlas either automatically pauses clusters or allows manual initiation of the pause. • **M0 Clusters** CANNOT be paused. • An automatically paused **M0** cluster can be resumed or terminated any time.	• To resume monitoring a paused cluster, connect using **MongoDB Driver**, `mongosh`, or **Data Explorer**. • When a **Backup Compliance Policy** is enabled, resuming a cluster in Atlas automatically enables **Cloud Backup**.

Terminate	Terminating a cluster also deletes any backup snapshots for that cluster.

Summary

- To get started with MongoDB Atlas, begin to create an Atlas account, set up a cluster, and then create a user for that cluster.
- MongoDB Atlas provides sample data to load into the Atlas database deployments.
- A dataset can be imported from the local system into the MongoDB Atlas using the `mongoimport` command.
- A dataset can be exported to the local system from the MongoDB Atlas Database using the `mongoexport` command.
- As part of administration tasks, the MongoDB clusters can be paused, resumed, and terminated.

Session: 12

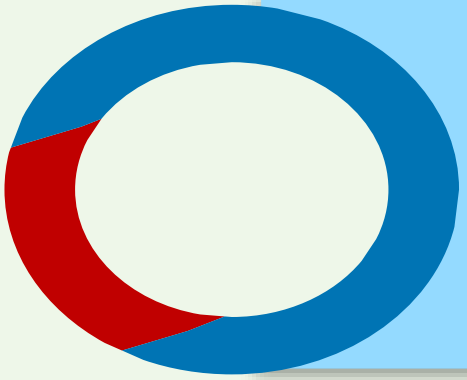
MONGODB DATABASE CONNECTIVITY WITH PYTHON

Managing Large Datasets Using MongoDB

Objectives

- Describe how to connect Python with MongoDB
- Explain how to install `PyMongo` Driver
- Describe how to create a database and collection in MongoDB using Python
- Describe the ways to query, sort, update, and delete documents in MongoDB using Python

Connecting Python with MongoDB



Create a connection between Python and MongoDB using `PyMongo`.

Install PyMongo Driver

Download and install the latest version of Python from the URL:
https://www.python.org/downloads/

To install PyMongo:

01 Open the Command Prompt.

02 Navigate to the directory where Python's installation file is located.

03 Execute the command:

```
pip install pymongo
```

Create a Database and Collection Using Python

01 **Establish a connection with MongoDB.**

```
from pymongo import MongoClient
client = MongoClient()
```

02 **Create a database in MongoDB.**

```
db = client.library
```

03 **Create a collection in MongoDB.**

```
lib_collection = db.library_details
```

04 **Insert documents into the collection.**

```
lib =[<document1>, <document2>,...]
library_details.insert_many(lib)
```

Query Documents Using Python

- Find a Document
- Find All Documents
- Filter the Result
- Filter Query Results Using Comparison Operators
- Filter Query Results Using Boolean Operators
- Sort the Query Results
- Update a Document
- Delete a Document

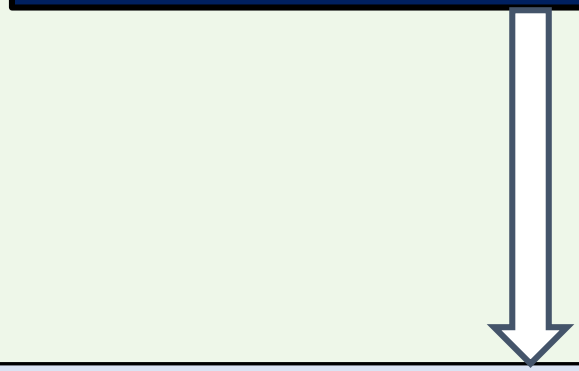
Find a Document

To fetch the first document from the collection:

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.library
library_details = db.library_details
pprint.pprint(library_details.find_one())
```

Find All Documents

To fetch all documents from the collection:



```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.library
library_details = db.library_details
for lib in library_details.find({}):
print(lib)
```

Filter the Result

To filter the fetched documents based on a field:

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.library
library_details = db.library_details
for lib in
library_details.find({"book_author":
"Wes Mckinney"}):
print(lib)
```

Filter Query Results Using Comparison Operators

To filter the fetched documents based on comparison operators:

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.library
library_details = db.library_details
for lib in
library_details.find({"volume":{"$gt":1}}):
print(lib)
```

Filter Query Results Using Boolean Operators

To filter the fetched documents using Boolean operators:

```
client = MongoClient()
db = client.library
library_details = db.library_details
for lib in library_details.find({"$and":[{"book_name":
"Python Programming"},{"volume":{"$gt":1}}]}):
 print(lib)
```

Sort the Query Results

To sort the documents that are fetched:

```
library_details.find().sort("field name", -1)
```

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.library
library_details = db.library_details
for lib in
library_details.find().sort("book_name",
-1):
 print(lib)
```

Update a Document

To update the value of a field in a document:

```
library_details.update_one(myquery, updatevalue)
```

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.library
library_details = db.library_details
myquery = { "book_name": "Python Programming" }
updatevalue = { "$set":{"volume":4} }
library_details.update_one(myquery, updatevalue)
for lib in library_details.find():
print(lib)
```

Delete a Document

To delete a document from the collection:

```
library_details.delete_one
(myquery)
```

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client. library
library_details = db.library_details
myquery = {"book_author":"David Beazley"}
library_details.delete_one(myquery)
for lib in library_details.find():
print(lib)
```

Summary

- The combination of Python, a versatile programming language, and MongoDB, an extremely scalable and reliable database, forms a robust database application.
- `PyMongo` is MongoDB's official Python driver that helps to create a connection between Python and MongoDB.
- `PyMongo` can be installed using the command `pip install PyMongo`.
- After establishing the connection between Python and MongoDB, databases and collections can be created in MongoDB using Python.
- The collections in the MongoDB database can be queried to retrieve either a document or all the documents in the collection.
- The results of a query can be filtered using the comparison or Boolean operators.
- The results of a query can be sorted using the `sort` method.
- The documents in a MongoDB collection can be updated or deleted using Python by specifying conditions.