Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

«Работа с БД в СУБД MongoDB»

по дисциплине «Проектирование и реализация баз данных»

Обучающийся Тутубалин Кирилл Сергеевич
Факультет прикладной информатики
Группа K3241
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

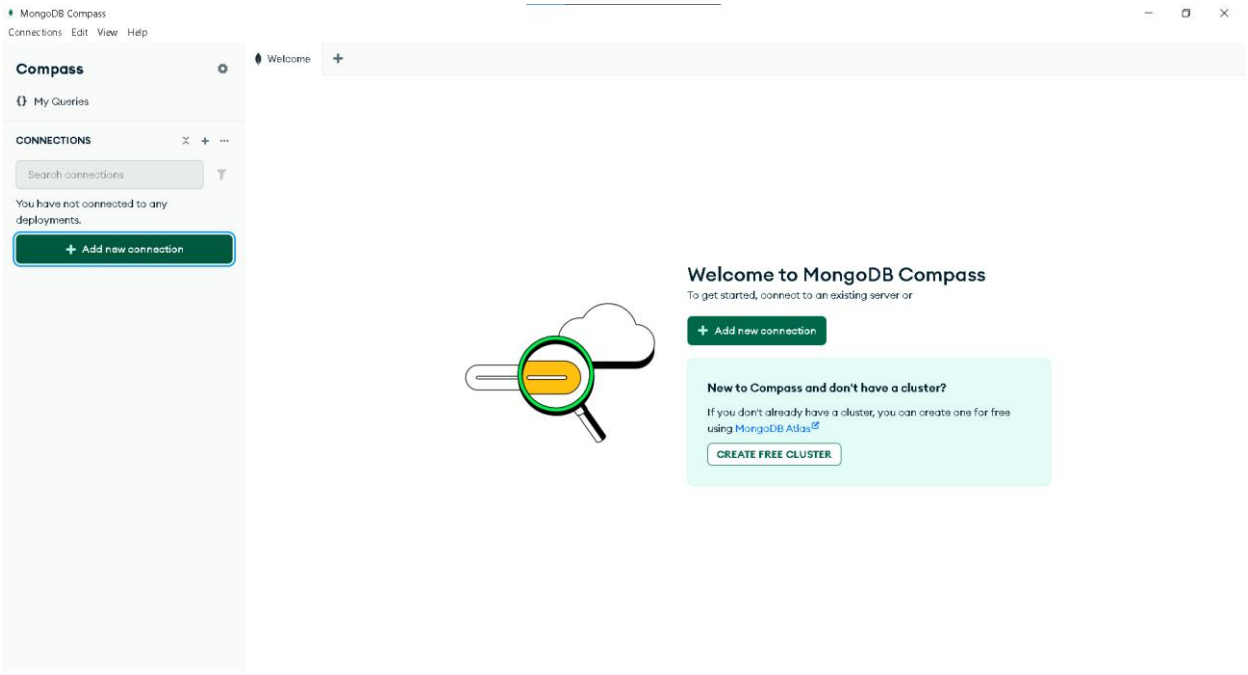
Санкт-Петербург
2024/2025

6.1 ВВЕДЕНИЕ В СУБД MONGODB. УСТАНОВКА MONGODB. НАЧАЛО РАБОТЫ С БД

Цель: овладеть практическими навыками установки СУБД MongoDB.

Ход работы:

1. Была установлена MongoDB Community Server с официального сайта. В процессе установки был также установлен MongoDB Shell (`mongosh`) и MongoDB Compass (графический интерфейс).



2. Запускаем клиент mongosh

```
C:\Users\Кирилл>mongosh
Current Mongosh Log ID: 683854f8942d7200056c4bcf
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.1
Using MongoDB:          8.0.9
Using Mongosh:          2.5.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
```

3. Проверяем работоспособность. (Версия).

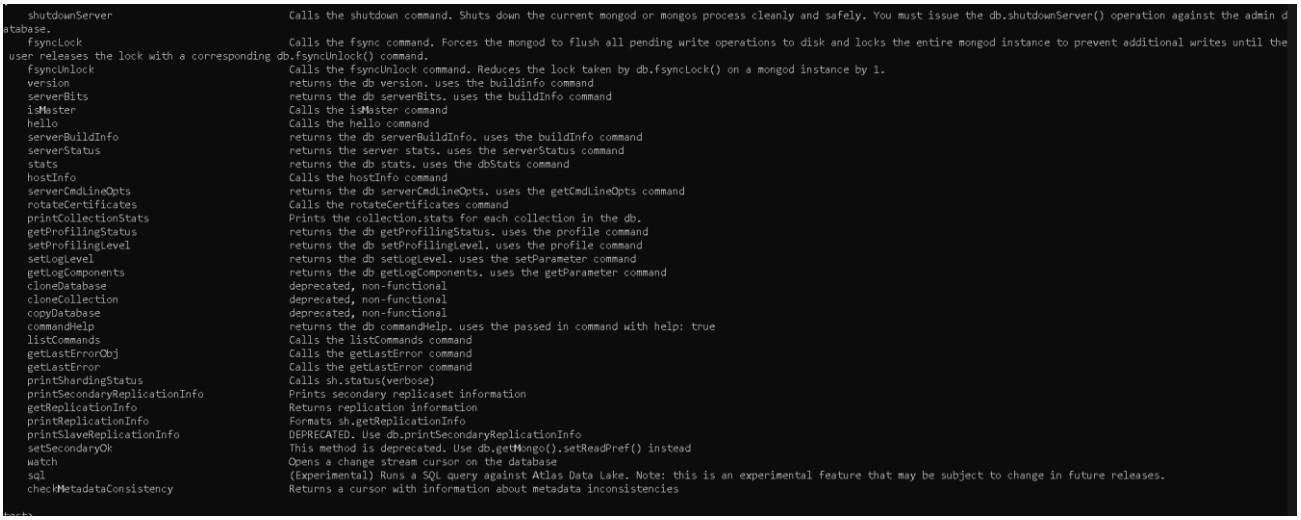
```
test> db.version()
8.0.9
```

4. db.help(), db.help, db.stats()

```
test> db.help()

  Database Class:

    getMongo                                Returns the current database connection
    getName                                 Returns the name of the DB
    getCollectionNames                      Returns an array containing the names of all collections in the current database.
    getCollectionInfos                      Returns an array of documents with collection information, i.e. collection name and options, for the current database.
    runCommand                              Runs an arbitrary command on the database.
    adminCommand                            Runs an arbitrary command against the admin database.
    aggregate                               Runs a specified admin/diagnostic pipeline which does not require an underlying collection.
    getSiblingDB                            Returns another database without modifying the db variable in the shell environment.
    getCollection                           Returns a collection or a view object that is functionally equivalent to using the db.<collectionName>.
    dropDatabase                            Removes the current database, deleting the associated data files.
    createUser                              Creates a new user for the database on which the method is run. db.createUser() returns a duplicate user error if the user already exists on the database.
    updateUser                              Updates the user's profile on the database on which you run the method. An update to a field completely replaces the previous field's values. This includes updates
to the user's roles array.
    changeUserPassword                      Updates a user's password. Run the method in the database where the user is defined, i.e. the database you created the user.
    logout                                  Ends the current authentication session. This function has no effect if the current session is not authenticated.
    dropUser                                Removes the user from the current database.
    dropAllUsers                            Removes all users from the current database.
    auth                                    Allows a user to authenticate to the database from within the shell.
    grantRolesToUser                        Grants additional roles to a user.
    revokeRolesFromUser                     Removes a one or more roles from a user on the current database.
    getUser                                 Returns user information for a specified user. Run this method on the user's database. The user must exist on the database on which the method runs.
    getUsers                                Returns information for all the users in the database.
    createCollection                        Create new collection
    createEncryptedCollection               Creates a new collection with a list of encrypted fields each with unique and auto-created data encryption keys (DEKs). This is a utility function that internally u
tilises ClientEnryption.createEncryptedCollection.
    createView                              Create new view
    createRole                              Creates a new role.
    updateRole                              Updates the role's profile on the database on which you run the method. An update to a field completely replaces the previous field's values.
    dropRole                                Removes the role from the current database.
    dropAllRoles                            Removes all roles from the current database.
    grantRolesToRole                        Grants additional roles to a role.
test>
    grantPrivilegesToRole                   Grants additional privileges to a role.
    revokePrivilegesFromRole                Removes a one or more privileges from a role on the current database.
    getRole                                 Returns role information for a specified role. Run this method on the role's database. The role must exist on the database on which the method runs.
    getRoles                                Returns information for all the roles in the database.
    currentOp                               Runs an aggregation using $currentOp operator. Returns a document that contains information on in-progress operations for the database instance. For further informa
tion, see $currentOp.
    killOp                                  Calls the killOp command. Terminates an operation as specified by the operation ID. To find operations and their corresponding IDs, see $currentOp or db.currentOp()
```

```
    shutdownServer                          Calls the shutdown command. Shuts down the current mongod or mongos process cleanly and safely. You must issue the db.shutdownServer() operation against the admin d
atabase.
    fsyncLock                               Calls the fsync command. Forces the mongod to flush all pending write operations to disk and locks the entire mongod instance to prevent additional writes until the
 user releases the lock with a corresponding db.fsyncUnlock() command.
    fsyncUnlock                             Calls the fsyncUnlock command. Reduces the lock taken by db.fsyncLock() on a mongod instance by 1.
    version                                 returns the db version. uses the buildinfo command
    serverBits                              returns the db serverBits. uses the buildInfo command
    isMaster                                Calls the isMaster command
    hello                                   Calls the hello command
    serverBuildInfo                         returns the db serverBuildInfo. uses the buildInfo command
    serverStatus                            returns the server stats. uses the serverStatus command
    stats                                   returns the db stats. uses the dbStats command
    hostInfo                                Calls the hostInfo command
    serverCmdLineOpts                       returns the db serverCmdLineOpts. uses the getCmdLineOpts command
    rotateCertificates                      Calls the rotateCertificates command
    printCollectionStats                    Prints the collection.stats for each collection in the db.
    getProfilingStatus                      returns the db getProfilingStatus. uses the profile command
    setProfilingLevel                       returns the db setProfilingLevel. uses the profile command
    setLogLevel                             returns the db setLogLevel. uses the setParameter command
    getLogComponents                        returns the db getLogComponents. uses the getParameter command
    cloneDatabase                           deprecated, non-functional
    cloneCollection                         deprecated, non-functional
    copyDatabase                            deprecated, non-functional
    commandHelp                             returns the db commandHelp. uses the passed in command with help: true
    listCommands                            Calls the listCommands command
    getLastErrorObj                         Calls the getLastError command
    getLastError                            Calls the getLastError command
    printShardingStatus                     Calls sh.status(verbose)
    printSecondaryReplicationInfo           Prints secondary replicaset information
    getReplicationInfo                      Returns replication information
    printReplicationInfo                    Formats sh.getReplicationInfo
    printSlaveReplicationInfo               DEPRECATED. Use db.printSecondaryReplicationInfo
    setSecondaryOk                          This method is deprecated. Use db.getMongo().setReadPref() instead
    watch                                   Opens a change stream cursor on the database
    sql                                     (Experimental) Runs a SQL query against Atlas Data Lake. Note: this is an experimental feature that may be subject to change in future releases.
    checkMetadataConsistency                Returns a cursor with information about metadata inconsistencies
```

```
test> db.help

  Database Class:

    getMongo                                Returns the current database connection
    getName                                 Returns the name of the DB
    getCollectionNames                      Returns an array containing the names of all collections in the current database.
    getCollectionInfos                      Returns an array of documents with collection information, i.e. collection name and options, for the current database.
    runCommand                              Runs an arbitrary command on the database.
    adminCommand                            Runs an arbitrary command against the admin database.
    aggregate                               Runs a specified admin/diagnostic pipeline which does not require an underlying collection.
    getSiblingDB                            Returns another database without modifying the db variable in the shell environment.
    getCollection                           Returns a collection or a view object that is functionally equivalent to using the db.<collectionName>.
    dropDatabase                            Removes the current database, deleting the associated data files.
    createUser                              Creates a new user for the database on which the method is run. db.createUser() returns a duplicate user error if the user already exists on the database.
    updateUser                              Updates the user's profile on the database on which you run the method. An update to a field completely replaces the previous field's values. This includes updates
to the user's roles array.
    changeUserPassword                      Updates a user's password. Run the method in the database where the user is defined, i.e. the database you created the user.
    logout                                  Ends the current authentication session. This function has no effect if the current session is not authenticated.
    dropUser                                Removes the user from the current database.
    dropAllUsers                            Removes all users from the current database.
    auth                                    Allows a user to authenticate to the database from within the shell.
    grantRolesToUser                        Grants additional roles to a user.
    revokeRolesFromUser                     Removes a one or more roles from a user on the current database.
    getUser                                 Returns user information for a specified user. Run this method on the user's database. The user must exist on the database on which the method runs.
    getUsers                                Returns information for all the users in the database.
    createCollection                        Create new collection
    createEncryptedCollection               Creates a new collection with a list of encrypted fields each with unique and auto-created data encryption keys (DEKs). This is a utility function that internally u
tilises ClientEnryption.createEncryptedCollection.
    createView                              Create new view
    createRole                              Creates a new role.
    updateRole                              Updates the role's profile on the database on which you run the method. An update to a field completely replaces the previous field's values.
    dropRole                                Removes the role from the current database.
    dropAllRoles                            Removes all roles from the current database.
    grantRolesToRole                        Grants additional roles to a role.
    revokeRolesFromRole                     Removes a one or more roles from a role on the current database.
    grantPrivilegesToRole                   Grants additional privileges to a role.
    revokePrivilegesFromRole                Removes a one or more privileges from a role on the current database.
    getRole                                 Returns role information for a specified role. Run this method on the role's database. The role must exist on the database on which the method runs.
    getRoles                                Returns information for all the roles in the database.
    currentOp                               Runs an aggregation using $currentOp operator. Returns a document that contains information on in-progress operations for the database instance. For further informa
tion, see $currentOp.
    killOp                                  Calls the killOp command. Terminates an operation as specified by the operation ID. To find operations and their corresponding IDs, see $currentOp or db.currentOp()
```



- db.help () – показывает список доступных методов
- db.help – выводит определение функции без выполнения (особенность JavaScript-оболочки)

```
test>
{
  db: 'test',
  collections: Long('0'),
  views: Long('0'),
  objects: Long('0'),
  avgObjSize: 0,
  dataSize: 0,
  storageSize: 0,
  indexes: Long('0'),
  indexSize: 0,
  totalSize: 0,
  scaleFactor: Long('1'),
  fsUsedSize: 0,
  fsTotalSize: 0,
  ok: 1
}
```

5. Создаем БД learn

```
test> use learn
switched to db learn
```

6. Смотрим список доступных БД

```
learn> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
```

БД learn еще не отображается, нужно вставить хотя бы один документ

7. Создаем коллекцию unicorns, вставляем в нее документ

```
learn> db.unicorns.insertOne({ name: 'Aurora', gender: 'f', weight: 450})
{
  acknowledged: true,
  insertedId: ObjectId('68385ffb563bab637f6c4bd0')
}
```

8. Смотрим список текущих коллекций

```
learn> show collections
unicorns
```

Теперь видим БД learn

```
learn> show dbs
admin    40.00 KiB
config  108.00 KiB
learn    40.00 KiB
local    40.00 KiB
```

9. Переименовываем коллекцию unicorns

```
learn> db.unicorns.renameCollection("horse")
{ ok: 1 }
```

```
learn> show collections
horse
```

10. Посмотреть статистику коллекции

```
    session: { 'object compaction': 0 },
    transaction: {
      'a reader raced with a prepared transaction commit and skipped an update or updates': 0,
      'number of times overflow removed value is read': 0,
      'race to read prepared update retry': 0,
      'rollback to stable history store keys that would have been swept in non-dryrun mode': 0,
      'rollback to stable history store records with stop timestamps older than newer records': 0,
      'rollback to stable inconsistent checkpoint': 0,
      'rollback to stable keys removed': 0,
      'rollback to stable keys restored': 0,
      'rollback to stable keys that would have been removed in non-dryrun mode': 0,
      'rollback to stable keys that would have been restored in non-dryrun mode': 0,
      'rollback to stable restored tombstones from history store': 0,
      'rollback to stable restored updates from history store': 0,
      'rollback to stable skipping delete rle': 0,
      'rollback to stable skipping stable rle': 0,
      'rollback to stable sweeping history store keys': 0,
      'rollback to stable tombstones from history store that would have been restored in non-dryrun mode': 0,
      'rollback to stable updates from history store that would have been restored in non-dryrun mode': 0,
      'rollback to stable updates removed from history store': 0,
      'rollback to stable updates that would have been removed from history store in non-dryrun mode': 0,
      'update conflicts': 0
    }
  },
  sharded: false,
  size: 65,
  count: 1,
  numOrphanDocs: 0,
  storageSize: 20480,
  totalIndexSize: 20480,
  totalSize: 40960,
  indexSizes: { _id_: 20480 },
  avgObjSize: 65,
  ns: 'learn.horse',
  nindexes: 1,
  scaleFactor: 1
}
```


11. Удалите коллекцию

```
learn> db.horse.drop()
true
```

```
learn> show collections
```

12. Удалите БД learn

```
learn> db.dropDatabase()
{ ok: 1, dropped: 'learn' }
```

```
learn> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
```

6.2 РАБОТА С БД В СУБД MONGODB

Цель: овладеть практическими навыками работы с CRUD-операциями, с вложенными объектами в коллекции базы данных MongoDB, агрегации и изменения данных, со ссылками и индексами в базе данных MongoDB.

Ход работы:

Практическое задание 2.1.1:

1. *Создайте базу данных learn.*
2. *Заполните коллекцию единорогов unicorns*
3. *Используя второй способ, вставьте в коллекцию единорогов документ*
4. *Проверьте содержимое коллекции с помощью метода find.*

Создаем базу данных

```
test> use learn
switched to db learn
```

Заполняем данными

```
learn> db.unicorns.insert({name: 'Horny', loves: ['carrot','papaya'], weight: 600, gender: 'm', vampires: 63});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a52107c703176c4bd0') }
}
learn> db.unicorns.insert({name: 'Aurora', loves: ['carrot', 'grape'], weight: 450, gender: 'f', vampires: 43});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a52107c703176c4bd1') }
}
learn> db.unicorns.insert({name: 'Unicrom', loves: ['energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a52107c703176c4bd2') }
}
learn> db.unicorns.insert({name: 'Roooooodles', loves: ['apple'], weight: 575, gender: 'm', vampires: 99});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a52107c703176c4bd3') }
}
learn> db.unicorns.insert({name: 'Solnara', loves:['apple', 'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a52107c703176c4bd4') }
}
learn> db.unicorns.insert({name:'Ayna', loves: ['strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a52107c703176c4bd5') }
}
learn> db.unicorns.insert({name:'Kenny', loves: ['grape', 'lemon'], weight: 690,  gender: 'm', vampires: 39});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a62107c703176c4bd6') }
}
learn> db.unicorns.insert({name: 'Raleigh', loves: ['apple', 'sugar'], weight: 421, gender: 'm', vampires: 2});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a62107c703176c4bd7') }
}
learn> db.unicorns.insert({name: 'Leia', loves: ['apple', 'watermelon'], weight: 601, gender: 'f', vampires: 33});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a62107c703176c4bd8') }
}
learn> db.unicorns.insert({name: 'Pilot', loves: ['apple', 'watermelon'], weight: 650, gender: 'm', vampires: 54});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a62107c703176c4bd9') }
}
learn> db.unicorns.insert({name: 'Nimue', loves: ['grape', 'carrot'], weight: 540, gender: 'f'});
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c4a62107c703176c4bda') }
```

Используем второй способ вставки

```
learn> document=({name: 'Dunx', loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165})
{
  name: 'Dunx',
  loves: [ 'grape', 'watermelon' ],
  weight: 704,
  gender: 'm',
  vampires: 165
}
learn> db.unicorns.insert(document)
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6838c5d12107c703176c4bdb') }
}
```

Проверяем содержимое коллекции

```
learn> db.unicorns.find()
[
  {
    _id: ObjectId('6838c4a52107c703176c4bd0'),
    name: 'Horny',
    loves: [ 'carrot', 'papaya' ],
    weight: 600,
    gender: 'm',
    vampires: 63
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd1'),
    name: 'Aurora',
    loves: [ 'carrot', 'grape' ],
    weight: 450,
    gender: 'f',
    vampires: 43
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd2'),
    name: 'Unicrom',
    loves: [ 'energon', 'redbull' ],
    weight: 984,
    gender: 'm',
    vampires: 182
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd3'),
    name: 'Roooooodles',
    loves: [ 'apple' ],
    weight: 575,
    gender: 'm',
    vampires: 99
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd4'),
    name: 'Solnara',
    loves: [ 'apple', 'carrot', 'chocolate' ],
    weight: 550,
    gender: 'f',
    vampires: 80
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd5'),
    name: 'Ayna',
    loves: [ 'strawberry', 'lemon' ],
    weight: 733,
    gender: 'f',
```

```
  _id: ObjectId('6838c4a62107c703176c4bd6'),
  name: 'Kenny',
  loves: [ 'grape', 'lemon' ],
  weight: 690,
  gender: 'm',
  vampires: 39
},
{
  _id: ObjectId('6838c4a62107c703176c4bd7'),
  name: 'Raleigh',
  loves: [ 'apple', 'sugar' ],
  weight: 421,
  gender: 'm',
  vampires: 2
},
{
  _id: ObjectId('6838c4a62107c703176c4bd8'),
  name: 'Leia',
  loves: [ 'apple', 'watermelon' ],
  weight: 601,
  gender: 'f',
  vampires: 33
},
{
  _id: ObjectId('6838c4a62107c703176c4bd9'),
  name: 'Pilot',
  loves: [ 'apple', 'watermelon' ],
  weight: 650,
  gender: 'm',
  vampires: 54
},
{
  _id: ObjectId('6838c4a62107c703176c4bda'),
  name: 'Nimue',
  loves: [ 'grape', 'carrot' ],
  weight: 540,
  gender: 'f'
},
{
  _id: ObjectId('6838c5d12107c703176c4bdb'),
  name: 'Dunx',
  loves: [ 'grape', 'watermelon' ],
  weight: 704,
  gender: 'm',
  vampires: 165
}
```

Практическое задание 2.2.1:

1) *Сформируйте запросы для вывода списков самцов и самок единорогов. Ограничьте список самок первыми тремя особями. Отсортируйте списки по имени.*

Список самцов отсортированный по именам

```
learn> db.unicorns.find({gender: 'm'}).sort({name: 1})
[
  {
    _id: ObjectId('6838c5d12107c703176c4bdb'),
    name: 'Dunx',
    loves: [ 'grape', 'watermelon' ],
    weight: 704,
    gender: 'm',
    vampires: 165
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd0'),
    name: 'Horny',
    loves: [ 'carrot', 'papaya' ],
    weight: 600,
    gender: 'm',
    vampires: 63
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd6'),
    name: 'Kenny',
    loves: [ 'grape', 'lemon' ],
    weight: 690,
    gender: 'm',
    vampires: 39
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd9'),
    name: 'Pilot',
    loves: [ 'apple', 'watermelon' ],
    weight: 650,
    gender: 'm',
    vampires: 54
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd7'),
    name: 'Raleigh',
    loves: [ 'apple', 'sugar' ],
    weight: 421,
    gender: 'm',
    vampires: 2
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd3'),
    name: 'Roooooodles',
    loves: [ 'apple' ],
    weight: 575,
    gender: 'm',
    vampires: 99
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd2'),
    name: 'Unicrom',
    loves: [ 'energon', 'redbull' ],
    weight: 984,
    gender: 'm',
    vampires: 182
  }
]
learn>
```

Список самок отсортированный по имени и ограниченный первыми 3 особями

```
learn> db.unicorns.find({gender: 'f'}).limit(3).sort({name: 1})
[
  {
    _id: ObjectId('6838c4a52107c703176c4bd1'),
    name: 'Aurora',
    loves: [ 'carrot', 'grape' ],
    weight: 450,
    gender: 'f',
    vampires: 43
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd5'),
    name: 'Ayna',
    loves: [ 'strawberry', 'lemon' ],
    weight: 733,
    gender: 'f',
    vampires: 40
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd8'),
    name: 'Leia',
    loves: [ 'apple', 'watermelon' ],
    weight: 601,
    gender: 'f',
    vampires: 33
  }
]
```

2) *Найдите всех самок, которые любят carrot. Ограничьте этот список первой особью с помощью функций findOne и limit.*

```
learn> db.unicorns.findOne({loves: 'carrot'})
{
  _id: ObjectId('6838c4a52107c703176c4bd0'),
  name: 'Horny',
  loves: [ 'carrot', 'papaya' ],
  weight: 600,
  gender: 'm',
  vampires: 63
}
```

Практическое задание 2.2.2:

1) *Модифицируйте запрос для вывода списков самцов единорогов, исключив из результата информацию о предпочтениях и пол.*

```
learn> db.unicorns.find({gender: 'm'}, {loves: 0, gender: 0}).sort({name: 1})
[
  {
    _id: ObjectId('6838c5d12107c703176c4bdb'),
    name: 'Dunx',
    weight: 704,
    vampires: 165
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd0'),
    name: 'Horny',
    weight: 600,
    vampires: 63
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd6'),
    name: 'Kenny',
    weight: 690,
    vampires: 39
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd9'),
    name: 'Pilot',
    weight: 650,
    vampires: 54
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd7'),
    name: 'Raleigh',
    weight: 421,
    vampires: 2
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd3'),
    name: 'Roooooodles',
    weight: 575,
    vampires: 99
  },
  {
    _id: ObjectId('6838c4a52107c703176c4bd2'),
    name: 'Unicrom',
    weight: 984,
    vampires: 182
  }
```

Практическое задание 2.2.3:

1) Вывести список единорогов в обратном порядке добавления.

```
learn> db.unicorns.find().sort({$natural: -1})
[
  {
    _id: ObjectId('6838c5d12107c703176c4bdb'),
    name: 'Dunx',
    loves: [ 'grape', 'watermelon' ],
    weight: 704,
    gender: 'm',
    vampires: 165
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bda'),
    name: 'Nimue',
    loves: [ 'grape', 'carrot' ],
    weight: 540,
    gender: 'f'
  },
  {
    _id: ObjectId('6838c4a62107c703176c4bd9'),
    name: 'Pilot',
    loves: [ 'apple', 'watermelon' ],
    weight: 650,
    gender: 'm',
    vampires: 54
  },
```

Практическое задание 2.1.4:

1) *Вывести список единорогов с названием первого любимого предпочтения, исключив идентификатор.*

```
learn> db.unicorns.find({}, {_id: 0, loves: {$slice: 1}})
[
  {
    name: 'Horny',
    loves: [ 'carrot' ],
    weight: 600,
    gender: 'm',
    vampires: 63
  },
  {
    name: 'Aurora',
    loves: [ 'carrot' ],
    weight: 450,
    gender: 'f',
    vampires: 43
  },
  {
    name: 'Unicrom',
    loves: [ 'energon' ],
    weight: 984,
    gender: 'm',
    vampires: 182
  },
```

Практическое задание 2.3.1:

1) *Вывести список самок единорогов весом от полутонны до 700 кг, исключив вывод идентификатора.*

```
learn> db.unicorns.find({gender: 'f', weight: {$gte: 500, $lt: 700}}, {_id: 0})
[
  {
    name: 'Solnara',
    loves: [ 'apple', 'carrot', 'chocolate' ],
    weight: 550,
    gender: 'f',
    vampires: 80
  },
  {
    name: 'Leia',
    loves: [ 'apple', 'watermelon' ],
    weight: 601,
    gender: 'f',
    vampires: 33
  },
  {
    name: 'Nimue',
    loves: [ 'grape', 'carrot' ],
    weight: 540,
    gender: 'f'
  }
```

Практическое задание 2.3.2:

1) *Вывести список самцов единорогов весом от полутонны и предпочитающих “grape” и” lemon”, исключив вывод идентификатора.*

У нас такой один

```
learn> db.unicorns.find({gender: 'm', weight: {$gte: 500}, loves: {$all: ['grape', 'lemon']}}, {_id: 0})
[
  {
    name: 'Kenny',
    loves: [ 'grape', 'lemon' ],
    weight: 690,
    gender: 'm',
    vampires: 39
  }
]
learn>
```

Практическое задание 2.3.3:

1) Найти всех единорогов, не имеющих ключ vampires.

```
learn> db.unicorns.find({vampires: {$exists: false}})
[
  {
    _id: ObjectId('6838c4a62107c703176c4bda'),
    name: 'Nimue',
    loves: [ 'grape', 'carrot' ],
    weight: 540,
    gender: 'f'
  }
```

Практическое задание 2.3.4:

1) Вывести список упорядоченный список имен самцов единорогов с информацией об их первом предпочтении.

```
learn> db.unicorns.find({gender: 'm'}, {name: 1, loves: {$slice: 1}, _id:0}).sort({name: 1})
[
  { name: 'Dunx', loves: [ 'grape' ] },
  { name: 'Horny', loves: [ 'carrot' ] },
  { name: 'Kenny', loves: [ 'grape' ] },
  { name: 'Pilot', loves: [ 'apple' ] },
  { name: 'Raleigh', loves: [ 'apple' ] },
  { name: 'Roooooodles', loves: [ 'apple' ] },
  { name: 'Unicrom', loves: [ 'energon' ] }
```

Практическое задание 3.1.1:

1) *Создайте коллекцию towns*

```
learn> db.towns.insertMany([
...   {
...     name: "Punxsutawney",
...     populatiuon: 6200,
...     last_sensus: ISODate("2008-01-31"),
...     famous_for: [""],
...     mayor: {
...       name: "Jim Wehrle"
...     }
...   },
...   {
...     name: "New York",
...     populatiuon: 22200000,
...     last_sensus: ISODate("2009-07-31"),
...     famous_for: ["status of liberty", "food"],
...     mayor: {
...       name: "Michael Bloomberg",
...       party: "I"
...     }
...   },
...   {
...     name: "Portland",
...     populatiuon: 528000,
...     last_sensus: ISODate("2009-07-20"),
...     famous_for: ["beer", "food"],
...     mayor: {
...       name: "Sam Adams",
...       party: "D"
...     }
...   }
```

*2) Сформировать запрос, который возвращает список городов с независимыми мэрами (*party="I"). *Вывести только название города и информацию о мэре.*

```
learn> db.towns.find({"mayor.party": "I"}, {name: 1, mayor: 1, _id: 0})
[
  {
    name: 'New York',
    mayor: { name: 'Michael Bloomberg', party: 'I' }
  }
```

*3) Сформировать запрос, который возвращает список беспартийных мэров (*party *отсутствует*). *Вывести только название города и информацию о мэре.*

```
learn> db.towns.find({"mayor.party": {$exists: false}}, {name: 1, mayor: 1, _id: 0})
[ { name: 'Punxsutawney', mayor: { name: 'Jim Wehrle' } } ]
```

Практическое задание 3.1.2:

1. *Сформировать функцию для вывода списка самцов единорогов.*

```
learn> function printMaleUnicorns() {
...   var cursor = db.unicorns.find({ gender: 'm' });
...   cursor.forEach(function(unicorn) {
...     print(unicorn.name);
...   });
... }
[Function: printMaleUnicorns]
learn> printMaleUnicorns();
Horny
Unicrom
Roooooodles
Kenny
Raleigh
Pilot
Dunx
```

2. *Создать курсор для этого списка из первых двух особей с сортировкой в лексикографическом порядке.*

```
learn> var cursor = db.unicorns.find({ gender: 'm' }).sort({ name: 1 }).limit(2);
```

3. *Вывести результат, используя forEach.*

```
learn> cursor.forEach(function(unicorn) {
...   print(unicorn.name);
... });
Dunx
Horny
```

Практическое задание 3.2.1:

Вывести количество самок единорогов весом от полутонны до 600 кг.

```
learn> db.unicorns.find({gender: "f", weight: {$gte: 500, $lte: 600}}).count()
2
```

Практическое задание 3.2.2:

Вывести список предпочтений.

```
learn> db.unicorns.distinct("loves")
[
  'apple',      'carrot',
  'chocolate',  'energon',
  'grape',      'lemon',
  'papaya',     'redbull',
  'strawberry', 'sugar',
  'watermelon'
]
```

Практическое задание 3.2.3:

Посчитать количество особей единорогов обоих полов.

```
learn> db.unicorns.aggregate({"$group":{_id:"$gender",count:{$sum:1}}})
[ { _id: 'm', count: 7 }, { _id: 'f', count: 5 } ]
```

Практическое задание 3.3.1:

Выполнить команду save

```
learn> db.unicorns.updateOne(
...     {name: "Barny"},
...     {$set: {loves: ["grape"], weight: 340, gender: "m"}},
...     {upsert: true}
...     )
{
  acknowledged: true,
  insertedId: ObjectId('6857e159adcd4347a8ec34db'),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
```

```
learn> db.unicorns.find({name: "Barny"});
[
  {
    _id: ObjectId('6857e159adcd4347a8ec34db'),
    name: 'Barny',
    gender: 'm',
    loves: [ 'grape' ],
    weight: 340
  }
]
```

Практическое задание 3.3.2:

1. *Для самки единорога* Ayna *внести изменения в БД: теперь ее вес 800, она убила 51 вампира.*

```
learn> db.unicorns.update({name: "Ayna"},
...     {$set: {weight: 800, vampires: 51}}
...     )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
learn> db.unicorns.find({name: "Ayna"});
[
  {
    _id: ObjectId('6838c4a52107c703176c4bd5'),
    name: 'Ayna',
    loves: [ 'strawberry', 'lemon' ],
    weight: 800,
    gender: 'f',
    vampires: 51
  }
]
```

Практическое задание 3.3.3:

1. Для самца единорога Raleigh *внести изменения в БД: теперь он любит рэдбул.*

2.Проверить содержимое коллекции unicorns.

```
learn> db.unicorns.update({ name: "Raleigh" }, { $addToSet: { loves: "redbull" } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
learn> db.unicorns.find({name: "Raleigh"})
[
  {
    _id: ObjectId('6838c4a62107c703176c4bd7'),
    name: 'Raleigh',
    loves: [ 'apple', 'sugar', 'redbull' ],
    weight: 421,
    gender: 'm',
    vampires: 2
  }
]
```

Практическое задание 3.3.4:

1) *Всем самцам единорогов увеличить количество убитых вампиров на 5.*

```
learn> db.unicorns.updateMany(
...     {gender: "m"},
...     {$inc: {vampires: 5}}
...     )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 8,
  modifiedCount: 8,
  upsertedCount: 0
}
```

Практическое задание 3.3.5:

1) *Изменить информацию о городе Портланд: мэр этого города теперь беспартийный.*

```
learn> db.towns.update(
...     {name: "Portland"},
...     {$unset: {"mayor.party": 1}}
...     )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
learn> db.towns.find({name: "Portland"})
[
  {
    _id: ObjectId('68513a1569323a54496c4bd2'),
    name: 'Portland',
    populatiuon: 528000,
    last_sensus: ISODate('2009-07-20T00:00:00.000Z'),
    famous_for: [ 'beer', 'food' ],
    mayor: { name: 'Sam Adams' }
  }
```

Практическое задание 3.3.6:

1) *Изменить информацию о самце единорога* Pilot: *теперь он любит и шоколад.*

```
learn> db.unicorns.update(
...     {name: "Pilot"},
...     {$addToSet: {loves: "chocolate"}}
...     )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
learn> db.unicorns.find({name: "Pilot"})
[
  {
    _id: ObjectId('6838c4a62107c703176c4bd9'),
    name: 'Pilot',
    loves: [ 'apple', 'watermelon', 'chocolate' ],
    weight: 650,
    gender: 'm',
    vampires: 59
  }
```

Практическое задание 3.3.7:

1)*Изменить информацию о самке единорога* Aurora: *теперь она любит еще и*

сахар, и лимоны.

```
learn> db.unicorns.update(
...     {name: "Aurora"},
...     {$addToSet: {loves: {$each: ["sugar", "lemon"]}}}
...     )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
learn> db.unicorns.find({name: "Aurora"})
[
  {
    _id: ObjectId('6838c4a52107c703176c4bd1'),
    name: 'Aurora',
    loves: [ 'carrot', 'grape', 'sugar', 'lemon' ],
    weight: 450,
    gender: 'f',
    vampires: 43
  }
```

Практическое задание 3.4.1:

1) Создайте коллекцию towns, включающую следующие документы:

```
learn> db.towns.find()
[
  {
    _id: ObjectId('6857ed9b819998c0756c4bd1'),
    name: 'Punxsutawney',
    population: 6200,
    last_sensus: ISODate('2008-01-31T00:00:00.000Z'),
    famous_for: [ 'phil the groundhog' ],
    mayor: { name: 'Jim Wehrle' }
  },
  {
    _id: ObjectId('6857ed9b819998c0756c4bd2'),
    name: 'New York',
    population: 22200000,
    last_sensus: ISODate('2009-07-31T00:00:00.000Z'),
    famous_for: [ 'statue of liberty', 'food' ],
    mayor: { name: 'Michael Bloomberg', party: 'I' }
  },
  {
    _id: ObjectId('6857ed9b819998c0756c4bd3'),
    name: 'Portland',
    population: 528000,
    last_sensus: ISODate('2009-07-20T00:00:00.000Z'),
    famous_for: [ 'beer', 'food' ],
    mayor: { name: 'Sam Adams', party: 'D' }
  }
```

2) Удалите документы с беспартийными мэрами.

```
learn> db.towns.remove({"mayor.party": {$exists: false}})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, delete
Many, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 1 }
```

3) Проверьте содержание коллекции.

```
learn> db.towns.find()
[ acknowledged: true, deletedCount: 1 }
  {
    _id: ObjectId('6857ed9b819998c0756c4bd2'),
    name: 'New York',
    population: 22200000,
    last_sensus: ISODate('2009-07-31T00:00:00.000Z'),
    famous_for: [ 'statue of liberty', 'food' ],
    mayor: { name: 'Michael Bloomberg', party: 'I' }
  },
  {
    _id: ObjectId('6857ed9b819998c0756c4bd3'),
    name: 'Portland',
    population: 528000,
    last_sensus: ISODate('2009-07-20T00:00:00.000Z'),
    famous_for: [ 'beer', 'food' ],
    mayor: { name: 'Sam Adams', party: 'D' }
  }
```

4) Очистите коллекцию.

```
learn> db.towns.remove({})
{ acknowledged: true, deletedCount: 2 }
learn> db.towns.find()
```

5) Просмотрите список доступных коллекций.

```
learn> show collections
towns
unicorns
learn>
```

Контрольные вопросы:

1. Как используется оператор точка?

Оператор . в MongoDB используется для доступа к вложенным полям в документе.

Например:

db.cities.find({ "mayor.name": "Jim Wehrle" });

2. Как можно использовать курсор?

Курсор – это результат запроса find(), который позволяет пошагово перебирать документы.

3. Какие возможности агрегирования данных существуют в MongoDB?

Агрегация выполняется через aggregate(), поддерживает $group, $sum, $avg, $match, $sort, $count, distinct(), count() и др.

4. Какая из функций save или update более детально позволит настроить редактирование документов коллекции?

update() (и его современные аналоги updateOne, updateMany) - лучше для детального и контролируемого обновления.

update позволяет:

- обновлять только нужные поля ($set, $unset, $inc)
- использовать операторы условий
- обновлять несколько документов

save() (устаревший) заменяет весь документ целиком, и может перезаписать важные данные, если неаккуратно использовать.

5. Как происходит удаление документов из коллекции по умолчанию?

Метод remove({условие}) по умолчанию удаляет все документы, удовлетворяющие условию. Если указать {justOne: true} - удалит один.

<u>Практическое задание 4.1.1:</u>

1. *Создайте коллекцию зон обитания единорогов, указав в качестве идентификатора кратко название зоны, далее включив полное название и описание.*
2. *Включите для нескольких единорогов в документы ссылку на зону обитания, использую второй способ автоматического связывания.*
3. *Проверьте содержание коллекции едиорогов.*

```
learn> db.habitats.insertOne({_id: "forest", name: "Enchanted Forest", description:
"Dense magical forest filled with glowing flora and rate creatures"})
{ acknowledged: true, insertedId: 'forest' }
```

```
learn> db.unicorns.update( { name: "Barny" }, { $set: { habitat: { $ref: "habitats", $id: "forest" } } } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
learn> db.unicorns.find({name: "Barny"})
[
  {
    _id: ObjectId('6857e159adcd4347a8ec34db'),
    name: 'Barny',
    gender: 'm',
    loves: [ 'grape' ],
    weight: 340,
    vampires: 5,
    habitat: DBRef('habitats', 'forest')
  }
]
```

Практическое задание 4.2.1:

Проверьте, можно ли задать для коллекции unicorns индекс для ключа name с флагом unique.

```
learn> db.unicorns.createIndex({name: 1}, {unique: true})
name_1
```

Практическое задание 4.3.1:

1) Получите информацию о всех индексах коллекции unicorns .

2) Удалите все индексы, кроме индекса для идентификатора.

3) Попытайтесь удалить индекс для идентификатора.

```
learn> db.unicorns.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1', unique: true }
]
```

db

```
learn> db.unicorns.dropIndexes()
{
  nIndexesWas: 2,
  msg: 'non-_id indexes dropped for collection',
  ok: 1
}
```

```
learn> db.unicorns.dropIndex("_id_")
MongoServerError[InvalidOptions]: cannot drop _id index
```

Практическое задание 4.4.1:

1. Создайте объемную коллекцию numbers, задействовав курсор:

```
for(i = 0; i < 100000; i++){db.numbers.insert({value: i})}
```

2. Выберите последних четыре документа.

3. Проанализируйте план выполнения запроса 2. Сколько потребовалось времени на выполнение запроса? (по значению параметра executionTimeMillis)

4. Создайте индекс для ключа value.

5. Получите информацию о всех индексах коллекции numbres.

6. Выполните запрос 2.

7. Проанализируйте план выполнения запроса с установленным индексом. Сколько потребовалось времени на выполнение запроса?

8. Сравните время выполнения запросов с индексом и без. Дайте ответ на вопрос: какой запрос более эффективен?

```
learn> for (i = 0; i < 100000; i++) {
...   db.numbers.insert({ value: i });
... }
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('685879ad32237a30f56dd26f') }
}
```

```
learn> db.numbers.find().sort({ value: -1 }).limit(4)
[
  { _id: ObjectId('685879ad32237a30f56dd26f'), value: 99999 },
  { _id: ObjectId('685879ad32237a30f56dd26e'), value: 99998 },
  { _id: ObjectId('685879ad32237a30f56dd26d'), value: 99997 },
  { _id: ObjectId('685879ad32237a30f56dd26c'), value: 99996 }
]
```

```
learn> db.numbers.find().sort({ value: -1 }).limit(4).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'learn.numbers',
    parsedQuery: {},
    indexFilterSet: false,
    queryHash: 'BA27D965',
    planCacheShapeHash: 'BA27D965',
    planCacheKey: '7A892B81',
    optimizationTimeMillis: 0,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
      stage: 'SORT',
      sortPattern: { value: -1 },
      memLimit: 104857600,
      limitAmount: 4,
      type: 'simple',
      inputStage: { stage: 'COLLSCAN', direction: 'forward' }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 4,
    executionTimeMillis: 165,
    totalKeysExamined: 0,
    totalDocsExamined: 123637,
    executionStages: {
      isCached: false,
      stage: 'SORT',
      nReturned: 4,
      executionTimeMillisEstimate: 116,
      works: 123643,
      advanced: 4,
      needTime: 123638,
      needYield: 0,
      saveState: 10,
      restoreState: 10,
      isEOF: 1,
      sortPattern: { value: -1 },
      memLimit: 104857600,
```

```
    spilledDataStorageSize: 0,
    inputStage: {
      stage: 'COLLSCAN',
      nReturned: 123637,
      executionTimeMillisEstimate: 91,
      works: 123638,
      advanced: 123637,
      needTime: 0,
      needYield: 0,
      saveState: 10,
      restoreState: 10,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 123637
    }
  }
},
queryShapeHash: 'A1C8CFCE9F916AB3A6ABCC8ABBB22506390F4D987582D3F926F0F2B36CA78396',
command: {
  find: 'numbers',
  filter: {},
  sort: { value: -1 },
  limit: 4,
  '$db': 'learn'
},
serverInfo: {
  host: 'LAPTOP-LQQJ398O',
  port: 27017,
  version: '8.0.9',
  gitVersion: 'f882ef816d531ecfbb593843e4c554fda90ca416'
},
serverParameters: {
  internalQueryFacetBufferSizeBytes: 104857600,
  internalQueryFacetMaxOutputDocSizeBytes: 104857600,
  internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
  internalDocumentSourceGroupMaxMemoryBytes: 104857600,
  internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
  internalQueryProhibitBlockingMergeOnMongoS: 0,
  internalQueryMaxAddToSetBytes: 104857600,
  internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
  internalQueryFrameworkControl: 'trySbeRestricted',
  internalQueryPlannerIgnoreIndexWithCollationForRegex: 1
},
ok: 1
```

```
learn> db.numbers.createIndex({value: 1})
value_1
```

```
learn> db.numbers.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { value: 1 }, name: 'value_1' }
]
```

```
learn> db.numbers.find().sort({ value: -1 }).limit(4).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'learn.numbers',
    parsedQuery: {},
    indexFilterSet: false,
    queryHash: 'BA27D965',
    planCacheShapeHash: 'BA27D965',
    planCacheKey: '7A892B81',
learn>
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
learn>
      limitAmount: 4,
      inputStage: {
        stage: 'FETCH',
        inputStage: {
          stage: 'IXSCAN',
          keyPattern: { value: 1 },
          indexName: 'value_1',
          isMultiKey: false,
          multiKeyPaths: { value: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'backward',
          indexBounds: { value: [ '[MaxKey, MinKey]' ] }
        }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 4,
    executionTimeMillis: 0,
    totalKeysExamined: 4,
    totalDocsExamined: 4,
    executionStages: {
      isCached: false,
      stage: 'LIMIT',
      nReturned: 4,
      executionTimeMillisEstimate: 0
```

executionTimeMillis (1): 165

executionTimeMillis (2): 0

Вывод: с индексами поиск идет намного быстрее

Контрольные вопросы:

1. Назовите способы связывания коллекций в MongoDB:

В MongoDB коллекции можно связывать следующими способами:

Автоматическое связывание через DBRef - используется специальный объект:

{ "$ref": "collection", "$id": <значение>, "$db": <имя_БД> }.
Пример: company: new DBRef("companies", apple._id).

Ручное связывание (manual referencing) - сохранение _id связанного документа напрямую как значение поля, без использования DBRef.

Оба способа позволяют связать документы из разных коллекций, а затем по значению $id получить связанные данные через findOne().

2. Сколько индексов можно установить на одну коллекцию в БД MongoDB?

MongoDB позволяет установить до 64 индексов на одну коллекцию

(включая индекс по _id).

3. Как получить информацию о всех индексах базы данных

MongoDB?

Для каждой коллекции нужно вызвать:

db.<collection>.getIndexes()

Пример:

db.numbers.getIndexes()

Выводы:
В ходе лабораторной работы была освоена базовая работа с документной базой данных MongoDB: создание и удаление коллекций, вставка, обновление и удаление документов. Особое внимание уделялось работе с индексами, что позволило значительно повысить скорость выборок и улучшить производительность запросов. Также была изучена работа с курсорами для эффективной обработки больших объемов данных. Полученные навыки позволяют уверенно работать с MongoDB для решения практических задач хранения и обработки неструктурированных данных.