ЛАБОРАТОРНАЯ РАБОТА 5.2

**crud-операции в СУБД MONGODB. Вставка данных. выборка данных**

**Цель:** овладеть практическими навыками работы с CRUD-операциями в СУБД MongoDB.

**Оборудование**: компьютерный класс.

**Программное обеспечение**: СУБД MongoDB**.**

1. База данных документов

База данных состоит из коллекций, состоящих из документов. Имя коллекции – произвольный идентификатор, состоящий из не более чем 128 различных алфавитно-цифровых символов и знака подчеркивания. В то же время имя коллекции не должно начинаться с префикса system., так как он зарезервирован для внутренних коллекций (например, коллекция system.users содержит всех пользователей базы данных). Имя не должно содержать знака доллара $.

Максимальный размер BSON документа составляет 16 мегабайт.

Максимальный размер документа помогает гарантировать, что один документ не может использовать чрезмерное количество памяти или, во время передачи, чрезмерное количество трафика. Для хранения документов больше, чем максимальный размер, MongoDB обеспечивает GridFS API.

Запись в MongoDB является документом, который представляет собой структуру данных, состоящий из пар «ключ: значение»:

{

field1: value1,

field2: value2,

field3: value3,

...

fieldN: valueN

}

Значения полей могут включать в себя другие документы, массивы и массивы документов (рисунки 1-3).



Рисунок 1 – Пример структуры простого документа с использованием массива

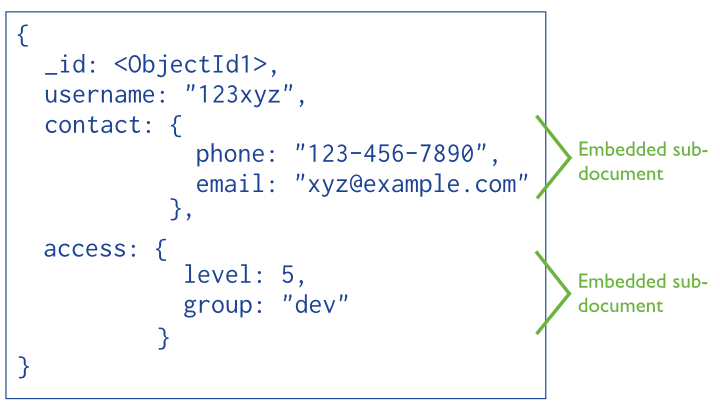


Рисунок 2 – Пример структуры документа с вложенными документами (денормализованная модель)

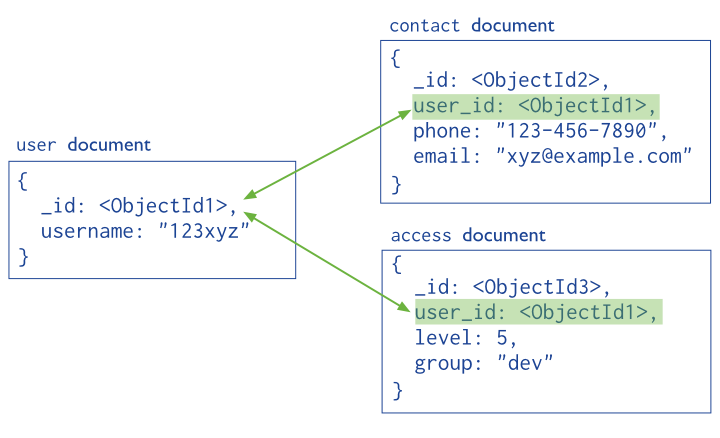


Рисунок 3 – Пример нормализованной модели с использованием ссылок между документами

В примере 3 приведена нормализованная модель БД, но это не означает наличие логических связей между коллекциями БД.

Ниже приведен пример документа, содержащий значения различных типов.

**var** mydoc = {

\_id: ObjectId("5099803df3f4948bd2f98391"),

name: { first: "Alan", last: "Turing" },

birth: **new** Date('Jun 23, 1912'),

death: **new** Date('Jun 07, 1954'),

contribs: [ "Turing machine", "Turing test", "Turingery" ],

views : NumberLong(1250000)

}

1. Вставка документов в коллекцию

Для вставки в коллекцию используется функция (метод) insert. Простой пример ее использования:

|  |  |
| --- | --- |
| 1  2 | > db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})  > db.users.find() |

В данном случае в коллекцию users добавляется простой объект.

Формат JSON допускает создание как единичных объектов, так и массивов.

> db.users.insert([{"name": "Tom", "age": 28, languages: ["english", "spanish"]}, {"name": "Bill", "age": 32, languages: ["english", "french"]})

Наиболее простой способом получения содержимого БД представляет использование метода find. Действие этой функции во многом аналогично обычному запросу SELECT \* FROM Table, который извлекает все строки.

Второй способ добавления в бд документа включает два этапа: определение документа (document = ( { ... } )) и собственно его добавление:

|  |  |
| --- | --- |
| 1  2 | > document=({"name": "Bill", "age": 32, languages: ["english", "french"]})  > db.users.insert(document) |

**Практическое задание 1:**

1. *Создайте базу данных learn. (use learn)*
2. *Заполните коллекцию единорогов unicorns:*

db.unicorns.insert({name: 'Horny', dob: new Date(1992,2,13,7,47), loves: ['carrot','papaya'], weight: 600, gender: 'm', vampires: 63});

db.unicorns.insert({name: 'Aurora', dob: new Date(1991, 0, 24, 13, 0), loves: ['carrot', 'grape'], weight: 450, gender: 'f', vampires: 43});

db.unicorns.insert({name: 'Unicrom', dob: new Date(1973, 1, 9, 22, 10), loves: ['energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});

db.unicorns.insert({name: 'Roooooodles', dob: new Date(1979, 7, 18, 18, 44), loves: ['apple'], weight: 575, gender: 'm', vampires: 99});

db.unicorns.insert({name: 'Solnara', dob: new Date(1985, 6, 4, 2, 1), loves:['apple', 'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});

db.unicorns.insert({name:'Ayna', dob: new Date(1998, 2, 7, 8, 30), loves: ['strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});

db.unicorns.insert({name:'Kenny', dob: new Date(1997, 6, 1, 10, 42), loves: ['grape', 'lemon'], weight: 690, gender: 'm', vampires: 39});

db.unicorns.insert({name: 'Raleigh', dob: new Date(2005, 4, 3, 0, 57), loves: ['apple', 'sugar'], weight: 421, gender: 'm', vampires: 2});

db.unicorns.insert({name: 'Leia', dob: new Date(2001, 9, 8, 14, 53), loves: ['apple', 'watermelon'], weight: 601, gender: 'f', vampires: 33});

db.unicorns.insert({name: 'Pilot', dob: new Date(1997, 2, 1, 5, 3), loves: ['apple', 'watermelon'], weight: 650, gender: 'm', vampires: 54});

db.unicorns.insert({name: 'Nimue', dob: new Date(1999, 11, 20, 16, 15), loves: ['grape', 'carrot'], weight: 540, gender: 'f'});

1. *Используя второй способ, вставьте в коллекцию единорогов документ:*

{name: 'Dunx', dob: new Date(1976, 6, 18, 18, 18), loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165}

document = ({name: 'Dunx', dob: new Date(1976, 6, 18, 18, 18), loves: ['grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165})

db. unicorns.insert(document)

1. *Проверьте содержимое коллекции с помощью метода find. (db.unicorns.find())*
2. Выборка данных из БД

Как было показано выше, наиболее простой способ получения содержимого БД представляет использование метода find. В большинстве запросов возникает необходимость извлечения только тех документов, которые удовлетворяют заданным критериям.

Пусть в БД добавлены документы:

|  |  |
| --- | --- |
| 1  2  3 | > db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]})  > db.users.insert({"name": "Bill", "age": 32, languages: ["english", "french"]})  > db.users.insert({"name": "Tom", "age": 32, languages: ["english", "german"]}) |

Выведем все документы, имеющие name=Tom:

|  |  |
| --- | --- |
| 1 | > db.users.find({name: "Tom"}) |

Такой запрос выведет два документа с именем name=Tom.

Для поиска документа по элементу в массиве используем следующий запрос: вывести все документы, у которых в массиве languages есть german.

|  |  |
| --- | --- |
| 1 | > db.users.find({languages: "german"}) |

Более сложный запрос: вывести те объекты, у которых name=Tom и одновременно age=32. На языке SQL это могло бы выглядеть так: SELECT \* FROM Table WHERE Name='Tom' AND Age=32. Данному критерию соответствует последний добавленный объект. Тогда использовать написать следующий запрос:

|  |  |
| --- | --- |
| 1 | > db.users.find({name: "Tom", age: 32})  Для того, чтобы извлечь лишь одно значение из полученной выборки можно воспользоваться методом findOne(). |

> db.users.findOne({name: "Tom"})

Однако, к методу findOne() прибегают чаще в тех ситуациях, когда ожидается, что результирующая коллекция будет содержать лишь один документ. В тех же случаях, когда следует ограничить выборку несколькими документами предпочитают использовать метод limit(), который принимает в качестве аргумента количество извлекаемых документов. Она задает максимально допустимое количество получаемых документов. Количество передается в виде числового параметра. Например, ограничим выборку тремя документами:

|  |  |
| --- | --- |
| 1 | > db.users.find().limit(3) |

В данном случае получены первые три документа (если в коллекции 3 и больше документов). Но что, если нужно произвести выборку не сначала, а пропустив какое-то количество документов? В этом используется функция **skip**. Например, пропустить первые три записи:

|  |  |
| --- | --- |
| 1 | > db.users.find().skip(3) |

MongoDB предоствляет возможности отсортировать полученный из бд набор данных с помощью функции **sort**. Передавая в эту функцию значения 1 или -1, можно указать в каком порядке сортировать: по возрастанию (1) или по убыванию (-1). Во многом эта функция по действию аналогична выражению ORDER BY в SQL. Например, сортировка по возрастанию по полю name:

|  |  |
| --- | --- |
| 1 | > db.users.find().sort({name: 1}) |

Можно совмещать все эти функции в одной цепочке:

|  |  |
| --- | --- |
| 1 | > db.users.find().sort({name: 1}).skip(3).limit(3) |

**Практическое задание 2:**

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

db.unicorns.find({gender: "m"})

db.unicorns.find({gender: "f"})

db.unicorns.find({gender: "f"}).limit(3)

db.unicorns.find({gender: "f"}).limit(3).sort({name: 1}) ИЛИ db.unicorns.find({gender: "f"}).sort({name: 1})

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

db.unicorns.find({gender: "f", loves: "carrot"}) ИЛИ db.unicorns.find({gender: "f", loves: "carrot"}).limit(1)

db.unicorns.findOne({gender: "f", loves: "carrot"})

По умолчанию выборка содержит все поля документа, однако, в том случае, если требуется выбрать лишь конкретные поля, методам find() и findOne() можно передавать второй аргумент в виде JSON-структуры, с ключами, совпадающими с названиями столбцов и значениями 1, если поле должно попадать в выборку и 0, если его необходимо исключить из выборки.

Вывести только значения полей "age" у все документов, в которых name=Tom:

|  |  |
| --- | --- |
| 1 | > db.users.find({name: "Tom"}, {age: 1}) |

Использование единицы в качестве параметра {age: 1} указывает, что запрос должен вернуть только содержание свойства age.

Обратная ситуация: нужно найти все параметры документа, кроме свойства age. В этом случае в качестве параметра указать 0:

|  |  |
| --- | --- |
| 1 | > db.persons.find({name: "Tom"}, {age: 0}) |

При этом надо учитывать, что даже если мы отметим, что мы хотим получить только поле name, поле \_id также будет включено в результирующую выборку. Поэтому, если не нужно видеть данное поле в выборке, то надо явным образом указать: {"\_id":0}

Альтернативно вместо 1 и 0 можно использовать true и false:

|  |  |
| --- | --- |
| 1 | > db.users.find({name: "Tom"}, {age: true, \_id: false}) |

**Практическое задание 3:**

*Модифицируйте запрос для вывода списков самцов единорогов, исключив из результата информацию о дате рождения и поле.*

db.unicorns.find({gender: "m"}, {dob: 0, gender: false})

Если нужно отсортировать ограниченную коллекцию, то можно воспользоваться параметром $natural. Этот параметр позволяет задать сортировку: документы передаются в том порядке, в каком они были добавлены в коллекцию, либо в обратном порядке.

Например, отобрать последние пять документов:

|  |  |
| --- | --- |
| 1 | > db.users.find().sort({ $natural: -1 }).limit(5) |

**Практическое задание 4:**

*Вывести список единорогов в обратном порядке добавления.*

db.unicorns.find().sort({ $natural: -1}).

Для работы с массивами используется оператор $slice. Он является в некотором роде комбинацией функций limit и skip.

Оператор $slice принимает два параметра. Первый параметр указывает на общее количество возвращаемых документов. Второй параметр необязательный, но если он используется, тогда первый параметр указывает на смещение относительно начала (как функция skip), а второй - на ограничение количества извлекаемых документов.

Например, в каждом документе определен массив languages для хранения языков, на которых говорит человек. Их может быть и 1, и 2, и 3 и более. И допустим, ранее мы добавили следующий объект:

|  |  |
| --- | --- |
| 1 | > db.users.insert({"name": "Tom", "age": "32", languages: ["english", "german"]}) |

Если необходимо при выводе документов сделать так, чтобы в выборку попадал только один язык из массива languages, а не весь массив, использовать запрос:

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom"}, {languages: {$slice : 1}}) |

Данный запрос при извлечении документа оставит в результате только первый язык из массива languages, то есть в данном случае english.

Обратная ситуация: нужно оставить в массиве также один элемент, но не с начала, а с конца. В этом случае необходимо передать в параметр отрицательное значение:

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom"}, {languages: {$slice : -1}}); |

Теперь в массиве окажется german, так как он первый с конца в добавленном элементе.

Использовать сразу два параметра:

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom"}, {languages: {$slice : [-1, 1]}}); |

Первый параметр говорит пропустить один элемент с конца (так как отрицательное значение), а второй параметр указывает на количество возвращаемых элементов массива. В итоге в массиве language окажется "german".

**Практическое задание 5:**

*Вывести список единорогов с названия первого любимого фрукта, исключив идентификатор.*

db.unicorns.find({}, {"loves": {$slice: 1}}, {\_id:false}).

В качестве селектора могут выступать не только строки, но и регулярные выражения, Например, найти все документы, в которых значение ключа name начинается с буквы T:

> db.users.find({name:/T\w+/i})

1. ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

В таблице ниже представлены логические операторы сравнения и логические операторы MongoDB.

|  |  |  |
| --- | --- | --- |
| Оператор SQL | MongoDB | Описание |
| < | $lt | меньше |
| <= | $lte | меньше или равно |
| > | $gt | больше |
| >= | $gte | больше или равно |
| <> | $ne | не равно |
| NOT | $not | отрицание |
| EXISTS | $exists | проверка существования поля |
| OR | $or | или |
| NOT OR | $nor | не или |
| RLIKE, REGEXP | $regex | соответствие регулярному выражению |
| LIKE | $elemMatch | соответствие всех полей вложенного документа |
| - | $size | соответствие размеру массива |
| - | $type | соответствие, если поле имеет указанный тип |
| IN | $in | входит в список |
| NOT IN | $nin | не входит в список |
| ALL | $all | одновременное совпадение набора элементов |

Операторы из таблицы выше можно использовать как по отдельности, так и в комбинации.

В MongoDB в запросах можно использовать условные конструкции с помощью операторов сравнения:

* **$gt** (больше чем)
* **$lt** (меньше чем)
* **$gte** (больше или равно)
* **$lte** (меньше или равно)

Например, найти все документы, у которых значение ключа age меньше 30:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$lt : 30}}) |

Аналогично будет использование других операторов сравнения. Например, тот же ключ, только больше 30:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$gt : 30}}) |

Внимание: сравнение здесь проводится над целочисленными типами, а не строками. Если ключ age представляет строковые значения, то надо соответственно проводить сравнение над строками: db.users.find ({age: {$gt : "30"}}), однако результат будет тем же.

Если нужно найти все объекты со значением поля age больше 30, но меньше 50, можно скомбинировать два оператора:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$gt : 30, $lt: 50}}) |

**Практическое задание 5:**

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

*db.unicorns.find({weight:{$gt :500, $lt:700}},{ \_id:false},{gender: "f"})*

Оператор $ne извлекает все документы, **не** соответствующие некоторому условию:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$ne : 22}}) |

Оператор $in определяет массив возможных выражений и ищет те ключи, значение которых имеется в массиве:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$in : [22, 32]}}) |

Противоположным образом действует оператор $nin: он определяет массив возможных выражений и ищет те ключи, значение которых отсутствует в этом массиве:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$nin : [22, 32]}}) |

Оператор $all похож на $in: он также определяет массив возможных выражений, но требует, чтобы документы имели весь определяемый набор выражений. Например, следующий запрос не вернет нам ни одного результата:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$all : [22, 32]}}) |

Так как в документе, который представляет человека, может быть только одно значение поля age (не может же быть у человека два возраста), то естественно ни в одном документе не будет найдено сразу 22 и 32. Если мы сократим до одного элемента массива, тогда можем получить результат:

|  |  |
| --- | --- |
| 1 | > db.users.find ({age: {$all : [22]}}) |

Другая ситуация: человек может знать несколько языков, которые присваиваются ключу languages в форме массива. Тогда, например, чтобы найти всех людей, говорящих одновременно и по-английски, и по-французски, можно использовать следующее выражение:

|  |  |
| --- | --- |
| 1 | > db.users.find ({languages: {$all : ["english", "french"]}}) |

Оператор $or определяет набор пар ключ-значение, которые должны иметься в документе. И если документ имеет хоть одну такую пару ключ-значение, то он соответствует данному запросу и извлекается из бд:

|  |  |
| --- | --- |
| 1 | > db.users.find ({$or : [{name: "Tom"}, {age: 22}]}) |

Это выражение вернет все документы, в которых либо name=Tom, либо age=22.

Другой пример вернет все документы, в которых name=Tom, а age равно либо 22, либо среди значений languages есть "german":

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: "Tom", $or : [{age: 22}, {languages: "german"}]}) |

**Практическое задание 6:**

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

db.unicorns.find({weight:{$gt :500},gender: "m", loves:{$all: ["grape","lemon"]}} ,{\_id:0})

Оператор $exists позволяет извлечь только те документы, в которых определенный ключ присутствует или отсутствует. Например, вернуть все документы, в который есть ключ company:

|  |  |
| --- | --- |
| 1 | > db.users.find ({company: {$exists:true}}) |

Если указать у оператора $exists в качестве параметра false, то запрос вернет только те документы, в которых не определен ключ company.

**Практическое задание 7:**

*Найти всех единорогов, не имеющих ключ vampires.*

*db.unicorns.find({vampires:{$exists: false}})*

Оператор $regex задает регулярное выражение, которому должно соответствовать значение поля. Например, пусть поле name обязательно имеет букву "d":

|  |  |
| --- | --- |
| 1 | > db.users.find ({name: {$regex:"b"}}) |

Важно понимать, что $regex принимает не просто строки, а именно регулярные выражения, например: name: {$regex:"om$"} - значение name должно оканчиваться на "om".