

MongoDB II

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ЦЕЛЬ

Изучить Update и Delete. Изучить Aggregation Framework в MongoDB

ПЛАН ЗАНЯТИЯ

Изучить Update, Delete

aggregate()

Подготовка

Для подготовки к дальнейшим примерам выполним данный код.

Мы сознательно допустили ошибку указав, что акулы не являются хищниками.

```
db.animals.insertMany([
  {kind: "tiger", weight: 300, name: "Alan", predatory: true},
  {kind: "penguin", weight: 2, name: "Kovalsky", predatory:
true},
  {kind: "chicken", weight: 1, name: "Cipa", predatory: false},
  {kind: "shark", weight: 400, name: "Sharp", predatory: false},
  {kind: "shark", weight: 450, name: "John", predatory: false},
  {kind: "panda", weight: 200, name: "Po", predatory: false},
]);
```

Update

- `updateOne()` изменить один документ
- `updateMany()` изменить несколько документов
 - аргументы
 - `filter`
 - `action`

Операторы

- `$set,`
- `$inc,`
- `$rename,`
- `$push,`
- `$pull,`
- `$unset`

Изменение нескольких документов - \$set

Изменим свойство predatory у акул
- на true.

updateMany - обновит все
документы соответствующие
фильтру

Первый параметр: { kind:
"shark" } - это параметр
поиска

Второй: сами изменения

\$set оператор изменит поле
predatory у всех акул на true

```
db.animals.updateMany({ kind: "shark" }, { $set: {  
predatory: true } });
```


Изменение нескольких документов - \$inc

`$inc` увеличивает значение на указанную величину

В данном примере мы увеличиваем вес всех животных на три.

В качестве первого параметра передали пустой объект поиска, так мы совсем не ограничили поиск

```
db.animals.updateMany({}, {$inc: {weight: 3}});
```

Изменение нескольких документов - \$rename

RENAME - переименовать
имя свойства

Переименуем у всех
животных свойство
weight в kg

```
db.animals.updateMany({}, {$rename: {weight:  
"kg"}})
```

Изменение нескольких документов

Добавим хищникам новое
свойство

```
foods: ["fish",  
"squid"]
```

```
db.animals.updateMany(  
  { predatory: true },  
  { $set: { foods: ["fish", "squid"] } }  
);
```

Изменение нескольких документов - \$push

С помощью push мы можем
добавить в массив еды
foods новое значение

В данном случае мы
добавили "wolf" в
массив еды тигра

```
db.animals.updateOne(  
  {kind: "tiger"},  
  {$push: {foods: "wolf"}}  
);
```

Изменение нескольких документов - \$pull

С помощью pull мы можем
забрать из массива еды
foods указанное
значение

В данном случае мы
убрали "squid" из
массив еды тигра.

```
db.animals.updateOne(  
  {kind: "tiger"},  
  {$pull: {foods: "squid"}}  
);
```

Изменение нескольких документов - \$unset

С помощью `unset` мы можем удалить само свойство (не только значение, но и само свойство исчезает)

В данном случае мы убрали у цыпленка свойство `predatory`

```
db.animals.updateOne(  
  {kind: "chicken"},  
  {$unset: {predatory: ""}}  
);
```

Изменение одного документа - updateOne()

Изменение конкретного документа удобнее всего осуществлять по `_id`

Замените `id` на значение из вашей базы

В данном примере мы изменили `name` х

```
db.animals.updateOne(  
  { _id: ObjectId("65c5e83e0829498ae91d3ebc") },  
  { $set: { name: "Kovalsky 2" } }  
);
```

MongoDB: CRUD

Delete

- `deleteOne()` удалить один документ
- `deleteMany()` удалить несколько документов
 - аргументы
 - `filter`

Удаление одного документа - deleteOne()

Удаление конкретного документа тоже удобно осуществить именно по `_id`

```
db.animals.deleteOne({_id:  
  ObjectId("65c5e83e0829498ae91d3ebe")});
```

Удаление нескольких документов - deleteMany()

Удаление многих можно
осуществить по
параметру поиска

В данном примере мы
удалили по все у кого
свойство **kind: "t-rex"**

```
// добавим динозавра  
db.animals.insertOne({kind: "t-rex", kg: 1500});  
  
// Устроим вымирание динозаврам  
db.animals.deleteMany({kind: "t-rex"});
```

MongoDB Aggregation Framework

1. позволяет получить вычисленные данные
2. инструмент для анализа, обработки данных
3. реализуется методом `aggregate()`
4. представляет конвейер (pipeline), который содержит определенные этапы обработки (stages)

```
// вывести всех пользователей
```

```
db.users.aggregate()
```

Основные стадии (этапы, stages) обработки - операторы

- `$match` фильтрация
- `$sort` сортировка (-1 по убыванию, 1 по возрастанию)
- `$project` проекция
- `$limit` лимитирование
- `$skip` пропустить (документы)
- `$group` группировка
- `$lookup` объединение коллекций
- `$addFields` добавить поля
- `$sample` получить произвольные документы
- `$count` возвращает ко-во документов

Aggregation - \$match

Получим всех хищных

```
db.animals.aggregate([  
  {$match: {predatory: true}}  
]);
```

Aggregation - \$sort

Получим всех хищных и
отсортируем их по весу
от тяжелого к легкому

```
db.animals.aggregate([  
  {$match: {predatory: true}},  
  {$sort: {kg: -1}}  
]);
```

Aggregation - \$limit

Чтобы получить самого
тяжелого не-хищника

Мы можем отфильтровать
не хищных,
отсортировать по весу и
оставить только первого
при помощи limit.

Например, limit(2) -
ограничить результат
первыми двумя
документами

```
db.animals.aggregate([  
  {$match: {predatory: false}},  
  {$sort: {kg: -1}},  
  {$limit: 1}  
]);
```

Aggregation - \$skip

В данном примере мы получаем третьего по тяжести хищника.

Для этого отфильтровали хищников, отсортировали по весу, пропустили первых двух, и из всех оставшихся – вывели первого

```
db.animals.aggregate([  
  { $match: { predatory: true } },  
  { $sort: { kg: -1 } },  
  { $skip: 2 },  
  { $limit: 1 }  
]);
```


Aggregation - \$project

С помощью проекции
можно указать, какие
поля нам интересно
получить, а какие нет.

Логика такая же как
была с find: 1 - хотим
отобразить, 0 - нет

```
db.animals.aggregate([  
  {$match: {kg: 3}},  
  {$project: {name: 1, foods: 1, _id: 0}}  
]);
```

Aggregation - \$count

С помощью count можно
посчитать, сколько
получено документов.

В данном примере

'number_of_planteaters'

это псевдоним под

которым выведется

результат

```
db.animals.aggregate([  
  {$match: {predatory: false}},  
  {$count: 'number_of_planteaters'}  
]  
);
```

Aggregation - \$sample

С помощью `sample` можно
получить случайный
документ

В данном примере мы
получим случайное
животное

```
db.animals.aggregate([  
  { $sample: { size: 1, }, },  
]);
```

Aggregation - \$sample

В данном примере мы
получим случайное
животное с весом больше
100 кг

```
db.animals.aggregate([  
  { $match: { kg: { $gt: 100 } } },  
  { $sample: { size: 1 } },  
]);
```

Aggregation - \$lookup

С помощью lookup можно “подсмотреть” в другую коллекцию для получения дополнительной информации

Чтобы рассмотреть lookup нам придется немного подготовиться, давайте добавим пост и комментарии к нему

```
// добавили Пост (для примера захардкодили id)  
db.posts.insertOne({  
  _id: ObjectId("65c6096e0451b42a2273e13c"),  
  likes: 10,  
  text: "Hi, I am glad to be on linkedin",  
});
```

Aggregation - \$lookup

Добавим комментарии к
посту

```
// добавили комментарий к этому посту  
db.comments.insertMany([  
  {  
    email: "fish@mail.com",  
    message: "Oh, you are here! Wonderful!",  
    post_id: ObjectId("65c6096e0451b42a2273e13c"),  
  },  
  {  
    email: "eidelman@mail.com",  
    message: "Hey, man!",  
    post_id: ObjectId("65c6096e0451b42a2273e13c"),  
  },  
  {  
    email: "eidelman@mail.com",  
    message: "P.S. love you so!",  
    post_id: ObjectId("65c6096e0451b42a2273e13c"),  
  },  
]);
```

Aggregation - \$lookup

Получим информацию из
одной коллекцию,
дополненную информацией
из другой

```
db.comments.aggregate([
  {$lookup: {
    from: "posts", // куда будем подсматривать -
                   коллекция
    localField: "post_id", // как называется в
                           колл. комментарии
    foreignField: "_id",   // как называется в
                           колл. посте
    as: 'post_info'
  }}
]);
```

Группировка, оператор \$group

- получает на входе документы
- объединяет их в группы по заданному **полю (или полям)** группировки
- на выходе - один документ равен одному уникальному значению **поля группировки**

Базовые операторы группировки (аккумуляторы)

- \$sum - сумма
- \$avg - среднее
- \$min - минимальное значение
- \$max - максимальное значение
- \$count - количество

Aggregation - \$group

Заполним базу данных информацией, чтобы мы могли совершить группировку

```
db.kids.insertMany([  
  { name: "John", age: 5, gender: "boy" },  
  { name: "Anna", age: 6, gender: "girl" },  
  { name: "Leyla", age: 4, gender: "girl" },  
  { name: "Frieda", age: 3, gender: "boy" },  
  { name: "Bob", age: 5, gender: "boy" },  
]) ;
```

Aggregation - \$group + \$avg

Заполним базу данных информацией, чтобы мы могли совершить группировку

В результате мы получаем средний возраст по полу

```
db.kids.aggregate([
  { $group: { _id: "$gender", averageAge: { $avg:
"$age" } } },
]);
```

```
1  [
2    {
3      "_id": "girl",
4      "averageAge": 5
5    },
6    {
7      "_id": "boy",
8      "averageAge": 4.333333333333333
9    }
10 ]
```

Aggregation - \$group + \$min

С помощью \$min мы можем
получить минимальный
возраст

```
db.kids.aggregate([  
  {$group: {_id: "$gender", minAge: {$min: "$age"}}}  
]);
```

В результате мы
получаем минимальный
возраст по полу

```
[  
  {  
    "_id": "girl",  
    "minAge": 4  
  },  
  {  
    "_id": "boy",  
    "minAge": 3  
  }  
]
```

Aggregation - \$group + \$max

С помощью \$max мы можем
получить минимальный
возраст

В результате мы
получаем максимальный
возраст по группам:
мальчики, девочки

```
db.kids.aggregate([  
  $group: { _id: "$gender", maxAge: { $max:  
    "$age" } }  
]);
```

```
1  [  
2    {  
3      "_id": "boy",  
4      "maxAge": 5  
5    },  
6    {  
7      "_id": "girl",  
8      "maxAge": 6  
9    }  
10 ]
```



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH