**Установка ссылок в БД**

**Последнее обновление: 30.10.2015**

В реляционных базах данных можно устанавливать внешние ключи, когда поля из одной таблицы ссылаются на поля в другой таблице. В MongoDB также можно устанавливать ссылки.

**Ручная установка ссылок**

Ручная установка ссылок сводится к присвоению значения поля \_id одного документа полю другого документа. Допустим, у нас могут быть коллекции, представляющие компании и работников, работающих в этих компаниях. Итак, сначала добавим в коллекцию companies документ представляющий компанию:

|  |  |
| --- | --- |
| 1 | > db.companies.insert({"\_id" : "microsoft", year: 1974}) |

Теперь добавим в коллекцию persons документ, представляющий работника. В этом документе будет поле company, представляющее компанию, где работает работник. И очень важно, что в качестве значения для этого поля мы устанавливаем не объект company, а значение ключа \_id добавленного выше документа:

|  |  |
| --- | --- |
| 1 | > db.users.insert({name: "Tom", age: 28, company: "microsoft"}) |

Теперь получим документ из коллекции users:

|  |  |
| --- | --- |
| 1 | > user = db.users.findOne() |

В данном случае имеется в виду, что выше добавленный элемент будет единственным в коллекции.

После этого консоль выводит полученный документ. И теперь найдем ссылку на его компанию в коллекции companies:

|  |  |
| --- | --- |
| 1 | > db.companies.findOne({\_id: user.company}) |

И если документ с таким идентификатором обнаружен, он отображается на консоли:



**Автоматическое связывание**

Используя функциональность **DBRef**, мы можем установить автоматическое связывание между документами. Посмотрим на примере применение данной функциональности. Вначале добавим новый документ в коллекцию companies:

|  |  |
| --- | --- |
| 1  2 | > apple=({"name" : "apple", "year": 1976})  > db.companies.save(apple) |

Обратите внимание, что в данном случае сохранение идет с помощью метода save, не insert. Метод save при добавлении нового документа генерирует \_id. И после сохранения мы можем вывести документ на консоль: > apple

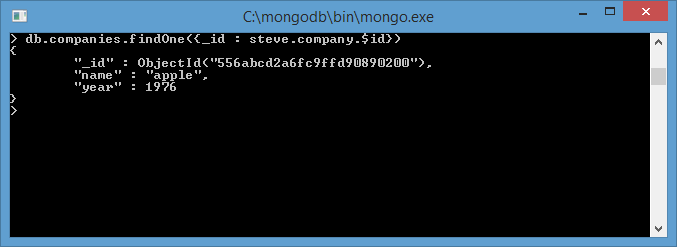
Теперь создадим новый документ для коллекции person, у которого ключ company свяжем с только что добавленным документом apple:

|  |  |
| --- | --- |
| 1  2 | > steve = ({"name": "Steve", "age": 25, company: new DBRef('companies',  apple.\_id)})  > db.users.save(steve) |



И мы можем протестировать:

|  |  |
| --- | --- |
| 1 | > db.companies.findOne({\_id: steve.company.$id}) |



Посмотрев на примере, теперь разберем организацию ссылок между документами. Для связывания с документом apple использовалось следующее выражение company: new DBRef('companies', apple.\_id)}). Формальный синтаксис DBRef следующий:

|  |  |
| --- | --- |
| 1 | { "$ref" : название\_коллекции, "$id": значение [, "$db" : название\_бд ]} |

Первый параметр $ref указывает на коллекцию, где хранится связанный документ. Второй параметр указывает на значение, которое и будет представлять что-то типа внешнего ключа. Третий необязательный параметр указывает на базу данных.

При тестировании в качестве запроса на выборку указывается выражение \_id: steve.company.$id. Так как person.company представляет теперь объект new DBRef('companies', apple.\_id)}), то нам надо конкретизировать параметр steve.company.**$id**

**Работа с индексами**

**Последнее обновление: 08.07.2016**

При поиске документов в небольших коллекциях мы не испытаем особых проблем. Однако когда коллекции содержат миллионы документов, а нам надо сделать выборку по определенному полю, то поиск нужных данных может занять некоторое время, которое может оказаться критичным для нашей задачи. В этом случае нам могут помочь индексы.

Индексы позволяют упорядочить данные по определенному полю, что впоследствии ускорит поиск. Например, если мы в своем приложении или задаче, как правило, выполняем поиск по полю name, то мы можем индексировать коллекцию по этому полю:

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

Таким образом с помощью метода createIndex устанавливается индекс по полю name. MongoDB позволяет установить до 64 индексов на одну коллекцию.

**Настройка индексов**

Если мы просто определим индекс для коллекции, например, db.users.createIndex({"name" : 1}), то мы все еще сможем добавлять в коллекцию документы с одинаковым значением ключа name. Однако, если нам потребуется, чтобы в коллекцию можно было добавлять документ с одним и тем же значением ключа только один раз, мы можем установить флаг unique:

|  |  |
| --- | --- |
| 1 | > db.users.createIndex({"name" : 1}, {"unique" : true}) |

Теперь, если мы попытаемся добавить в коллекцию два документа с одним и тем же значением name, то мы получим ошибку.

В тоже время тут есть свои тонкости. Так, документ может не иметь ключа name. В этом случае для добавляемого документа автоматически создается ключ name со значением null. Поэтому при добавлении второго документа, в котором не определен ключ name, будет выброшено исключение, так как ключ name со значением null уже присутствует в коллекции.

Также можно задать уникальный индекс сразу для двух полей:

|  |  |
| --- | --- |
| 1 | > db.users.createIndex({"name" : 1, "age" : 1}, {"unique" : true}) |

Однако в этом случае все добавляемые документы должны иметь уникальные значения для обоих полей.

Кроме того, тут есть свои ограничения. Например, значение поля, по которому идет индексация, не должно быть больше 1024 байт.

**Управление индексами**

Все индексы базы данных хранятся в системной коллекции *system.indexes*. Обратившись к ней, мы можем получить все индексы и связанную с ними информацию:

|  |  |
| --- | --- |
| 1 | > db.system.indexes.find() |

Также мы можем воспользоваться методом getIndexes и вывести всю информацию об индексах для конкретной коллекции:

|  |  |
| --- | --- |
| 1 | > db.users.getIndexes() |

Данная команда вернет вывод наподобие следующего:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | [      {          "v" : 1,          "key" : {                  "\_id" : 1          },          "ns" : "test.users",          "name" : "\_id\_"      },      {          "v" : 1,          "key" : {                  "name" : 1          },          "ns" : "test.users",          "name" : "name\_1"      }  ] |

Как мы видим, здесь для коллекции users (из бд test) определено 2 индекса: id и name. Поле key используется для поиска максимального и минимального значений, для различных операций, где надо применять данный индекс. Поле name применяется в качестве идентификатора для операций администрирования, например, для удаления индекса:

|  |  |
| --- | --- |
| 1 | > db.users.dropIndex("name\_1") |

**Управление коллекцией**

**Последнее обновление: 25.03.2018**

**Явное создание коллекции**

В предыдущих темах коллекция создавалась неявно автоматически при добавлении в нее первых данных. Но мы также можем создать ее явным образом, применив метод **db.createCollection(name, options)**, где name - название коллекции, а options - необязательный объект с дополнительными настройками инициализации. Например:

|  |  |
| --- | --- |
| 1  2 | > db.createCollection("accounts")  {"ok" : 1} |

Таким образом, создается коллекция accounts.

**Переименование коллекции**

В процессе работы, возможно, потребуется изменить название коллекции. Например, если при первом добавлении данных в ее названии была опечатка. И чтобы не удалять и затем пересоздавать коллекцию, следует использовать функцию renameCollection:

|  |  |
| --- | --- |
| 1 | > db.users.renameCollection("новое\_название") |

И если переименование пройдет удачно, то консоль отобразит:

{"ok" : 1}

**Ограниченные коллекции**

Когда мы отправляем запрос к бд на выборку, то MongoDB возвращает нам документы в том порядке, как правило, в котором они были добавлены. Однако такой порядок не всегда гарантируется, так как данные могут быть удалены, перемещены, изменены. Поэтому в MongoDB существует понятие **ограниченной коллекции** (capped collection). Подобная коллекция гарантирует, что документы будут располагаться в том же порядке, в котором они добавлялись в коллекцию. Ограниченные коллекции имеют фиксированный размер. И когда в коллекции уже нет места, наиболее старые документы удаляются, и в конец добавляются новые данные.

В отличие от обычных коллекций ограниченные мы можем задать явным образом. Например, создадим ограниченную коллекцию с названием profile и зададим для нее размер в 9500 байт:

|  |  |
| --- | --- |
| 1 | > db.createCollection("profile", {capped:true, size:9500}) |

И после удачного создания коллекции консоль выведет:

{"ok":1}

Также можно ограничить количество документов в коллекции, указав его в параметре max:

|  |  |
| --- | --- |
| 1 | > db.createCollection("profile", {capped:true, size:9500, max: 150}) |

Однако при таком способе создания коллекции следует учитывать, что если все место под коллекцию заполнено (например, выделенные нами 9500 байтов), а количество документов еще не достигло максимума, в данном случае 150, то в этом случае при добавлении нового документа самый старый документ будет удаляться, а на его место будет вставляться новый документ.

При обновлении документов в таких коллекциях надо помнить, что документы не должны расти в размерах, иначе обновление не удастся произвести.

Также нельзя удалять документы из подобных коллекций, можно только удалить всю коллекцию.

**Подколлекции**

Для упрощения организации данных в коллекциях мы можем использовать подколлекции. Например, данные по коллекции users надо разграничить на профили и учетные данные. И мы можем использовать создать коллекции db.users.profiles и db.users.accounts. При этом они не будут никак связаны с коллекцией users. То есть в итоге будут три разные коллекции, однако в плане логической организации хранения данных, возможно, для кого-то так будет проще.