ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

Кафедра «Математика и информатика»

Лабораторные работы

по дисциплине

«Анализ и поиск в больших базах данных»

Автор

Галабурдин А.В.

Ростов-на-Дону, 2026

Аннотация

Методические указания к лабораторным работам предназначены для студентов направления 09.04.02 Информационные системы и технологии, профиль: Искусственный интеллект, математическое моделирование и суперкомпьютерные технологии в разработке информационных систем.

Автор

доцент, кандидат физико-математических наук, доцент кафедры «Математика и информатика»
Галабурдин А.В.

Оглавление

ВВЕДЕНИЕ

Целью освоения дисциплины «Анализ и поиск в больших базах данных» является теоретическая и практическая подготовка студентов к работе с большими данными. Знания, полученные в результате освоения дисциплины, помогут при сборе и анализе огромных объемов структурированной или неструктурированной информации, при разработке моделей данных и получении новых знаний. Все это необходимо выпускнику, освоившему программу бакалавриата, для решения различных задач практической и научно-исследовательской деятельности.

Предлагаемые лабораторные работы позволят студентам получить навыки обработки и представления данных, а также приобрети умения применять на практике методы статистического и математического анализа. Самостоятельная работа студентов заключается в изучении литературы необходимой для усвоения и углубления знаний полученных при выполнении лабораторных работ.

Лабораторная работа №1
Знакомство с Mongo

Для работы с БД в Mongo Shell необходимо запустить файл mongo.exe, находящийся в папке <Домашний каталог>\bin, например D:\mongodb\mongo.exe.

Происходит подключение к серверу MongoDB и устанавливается умалчиваемая база test, к которой адресуются последующие запросы. Для работы с БД может быть запущено несколько таких приложений. Далее в ответ на приглашение «>» в командной строке вводятся команды и обращения к методам управления данными, а также дополнительные команды для получения справок и выполнения функций операционной системы.

Можно сказать упрощенно, что MongoDB состоит из ``баз данных", которые состоят из ``коллекций". ``Коллекции" состоят из ``документов". Каждый ``документ" состоит из ``полей". ``Коллекции" могут быть проиндексированы, что улучшает производительность выборки и сортировки. И наконец, получение данных из MongoDB сводится к получению "курсора", который отдает эти данные по мере надобности.

Основной единицей хранения, доступа и обработки в базе MongoDB является документ. Документ имеет набор свойств — атрибутов. Каждый атрибут в документе задается парой <Имя атрибута> : <Значение атрибута>. Наборы документов хранятся в коллекциях. Хотя в одну коллекцию обычно помещаются документы, представляющие однотипные объекты, документы в одной коллекции не обязаны иметь одинаковые наборы атрибутов. Также не требуется совпадения типов значений одноименных атрибутов в разных документах коллекции. Для идентификации каждый документ имеет атрибут с именем _id, содержащий уникальный ключ и набор заданных пользователем произвольных атрибутов. Если при включении документа ключ не задан, то он генерируется автоматически. Автоматически созданный ключ соответствует 24-разрядному шестнадцатеричному числу. В целом для задания документа используется формат JSON (JavaScript Object Notation).

Документ — неупорядоченный набор пар <Имя (ключ) атрибута> : <Значение атрибута>. Документ начинается с

открывающей фигурной скобки и заканчивается закрывающей фигурной скобкой. Имя и значение атрибута разделяет двоеточие. Пары <Имя атрибута> : <Значение атрибута> в документе разделяются запятой. Значение атрибута может иметь простой предопределенный в системе тип. Простые (атомарные) типы данных представлены в табл.

Кроме простого типа значение атрибута может быть представлено массивом значений или другим документом. Таким образом, атрибуты в документах могут образовывать вложенные структуры. Массив — упорядоченный набор значений. Массив начинается с открывающей квадратной скобки и заканчивается закрывающей квадратной скобкой. Элементы массива разделены запятой и индексируются целыми числами, начиная с нуля. Вложенные массивы не допускаются. Однако если элемент массива является документом, то его атрибуты также могут быть заданы массивами.

Справочная команда help выводит сведения о назначении других команд и обращения к справочникам по методам отдельных объектов MongoDB. Например, информацию о методах базы данных, имеющей псевдоним db, можно получить обращением к методу db.help (), а выполнение метода db.<Имя коллекции>.help () предоставит информацию о методах коллекции. Команды, необходимые для начала работы с MongoDB, представлены в таблице

Команда Назначение
Команды и методы для работы с БД в MongoDB Shell
help
Вывод списка команд MongoDB Shell и справочных методов для БД
db.help () Справочник по методам базы данных
db. <Имя коллекции>help () Справочник по методам коллекции
show dbs Вывод имен, имеющихся на сервере БД
use <Имя БД> Установка текущей БД, с которой далее ведется работа. Установить можно и несуществующую базу.

Создание первой коллекции приведет к созданию базы. Обращение к установленной базе выполняется под псевдонимом >db. Например, >db. getCollectionNames () выполняет метод БД, возвращающий имена коллекций в установленной базе

db.getName () Вывод имени текущей базы

show collections Вывод имен коллекций в установленной базе (команда приложения mongo shell)

show logs Вывод имен используемых журналов. По умолчанию имя журнала global

db.dropDatabase () Удаление базы

exit Выход из mongo shell

Команды создания и обработки коллекции в БД

db.createCollection (<Имя коллекции>. [,{size:… [, capped: …, max: …}]]) Создание новой коллекции

db.getCollectionNames () Вывод списка имен коллекций в виде массива.

db.<Имя коллекции>.find () Вывод документов из коллекции

db.<Имя существ. коллекции>. renameCollection ("<Новое имя коллекции>") Переименование коллекции

db.<Имя коллекции>.drop () Удаление коллекции

db.<Имя коллекции>.find () Вывод всех документов из заданной коллекции

В документе между любыми лексемами можно задавать произвольное число пробелов. Пример документа, содержащего сведения об авторе Джеймсе Олдридже и заданных в массиве Books двух его книгах:

{Au_id: "000–11–0001", Au_lname: "Олдридж, Джеймс", Years: [1918, 2015], Contract: false,

Books: [{Title_id: "BB1111", Title: "Морской орел"}, {Title_id: "BB2222", Title: "Последний дюйм"}]}.

При добавлении документа в базу указывается коллекция, в которую включается и где хранится документ. Коллекции в БД должны иметь уникальные имена. Управление документами выполняется функциями (методами) коллекции. Для поиска, включения, удаления или изменения документа необходимо вызвать

соответствующий метод коллекции. Коллекции являются объектами БД и имеют полный набор методов для доступа и обработки документов.

Создание и удаление коллекций

Далее следует выполнять команды, выделенные жирным шрифтом.

Создание новой коллекции выполняет метод createCollection для БД >db.createCollection (<Имя коллекции> [, {size:<Размер базы в байтах> [, capped: true|false, max:<Число документов>}]])

Например, создание коллекции authors

>db.createCollection ("authors");

приводит к созданию определения коллекции authors. Фактическое создание коллекции происходит при включении в нее первого документа.

Выполнение метода

>db.createCollection (authors, {size: 3000000})

создает коллекцию authors, для которой предварительно резервируется ~3 Мб в каталоге для баз данных, заданным параметром dbpath в конфигурационном файле mongod.cfg

Количественные характеристики коллекции выводит ее метод stats (): >db.<Имя коллекции>.stats (); Например, метод

db.authors.stats ();

выводит статистику коллекции authors в базе Authors (размеры букв в именах имеют значения):

"ns": "Authors. authors",
"count": 0,
"size": 0,
"storageSize": 3002368,
"numExtents": 1,
"nindexes": 1,
"lastExtentSize": 3002368,
"paddingFactor": 1,
"systemFlags": 1,
"userFlags": 0,
"totalIndexSize": 8176,
"indexSizes": { "_id_": 8176 },

"ok": 1

В строке "storageSize": 3002368 указан размер зарезервированной памяти

Необязательный параметр capped: true метода db.createCollection используется при создании ограниченных коллекций, в которых общее число документов не может превышать значение, заданное параметром max:<Число документов>. При добавлении в ограниченную коллекцию лишних документов происходит удаление наиболее старых документов. Ограниченные коллекции требуют обязательного указания размера резервируемой памяти (size) и не допускают добавления в существующие документы новых атрибутов.

Вывод списка имеющихся в базе коллекций выполняет метод db.getCollectionNames ().

Для полного удаления коллекции со всеми имеющимися в ней документами предназначен метод самой коллекции drop. Например, >db.authors.drop () удалит коллекцию authors из текущей БД.

Переименование коллекции выполняется методом renameCollection: db.<Имя существующей коллекции>.renameCollection ("<Новое имя коллекции>").

Создадим коллекцию OldCollection

> db.createCollection("OldCollection");

Убедимся в том, что она создана

>db.getCollectionNames()

Переименуем ее , дав ей имя NewCollection

> db.OldCollection.renameCollection("NewCollection")

Убедимся в том, что коллекция получила новое имя

> db.getCollectionNames()

Удалим коллекцию NewCollection

> db.NewCollection.drop()

Убедимся в том, что коллекция удалена

> db.getCollectionNames()

Включение документов в коллекцию

Добавление нового документа в коллекцию выполняет метод Insert (): db.<Имя коллекции>.insert (<Документ в формате JSON>)

или db.<Имя коллекции>.insert ({<Имя атрибута>:<Значение> [,..});

Например, включение в коллекцию авторов Николая Лескова может выполнить метод:

> db.authors.insert ({"Au_lname":"Лесков", "Au_fname": "Николай", "birthday": ISODate ("1831-02-16")});

При этом атрибут "birthday" (день рождения) имеет тип Date и поэтому задается строкой в аргументе функции ISODate, которая преобразует строку в значение типа дата-время в формате GMT. В примере время в течение суток не задано, поэтому будет доопределено нулем (началом суток).

Вывод документов из заданной коллекции

> db.authors.find()

Возвратит введенный документ

{ "_id" : ObjectId("58695c396b9421d84b9efb93"), "Au_lname" : "Лесков", "Au_fname" : "Николай", "birthday" : ISODate("1831-02-16T00:00:00Z") }

В момент включения документа сервер MongoDB автоматически создал для него ключевой атрибут _id, имеющий специальный тип ObjectId с уникальным значением ObjectId ("58695c396b9421d84b9efb93"). Ключ документа может использоваться в условии запроса для получения конкретного документа.

Запрос > db.authors.find ({_id: ObjectId("58695c396b9421d84b9efb93")})

возвратит тот же документ

Пользователь при добавлении документа методом insert может задать свое уникальное в коллекции значение атрибута _id. Например,

db.authors.insert ({_id: "000–11–1111", "Au_ lname": "Пелевин", " Au_fname ": "Ви´ктор", "year_of_birth": 1962, Contract: true});

По заданному пользователем значению _id также возможен поиск документа: db.authors.find ({_id: "000–11– 1111"});

Метод insert позволяет добавлять в коллекцию документы из переменной java script:

var doc={"Au_lname":"Н. В. Гоголь", "year_of_ birth":1809};

db.authors.insert (doc);

Добавим в коллекцию также документ

> var dsc={"_id":"01","Фамилия":"Калинин","Имя":"Юрий", "Книги":["Букварь","Чтение"]};

db.authors.insert (dsc);

и документ

> db.authors.insert ({_id: "02", "Автор": "Сумкин Н", "Книги ": ["Азбука"]})

Задание для самостоятельной работы

Создать базу данных с коллекцией, содержащей результаты последней экзаменационной сессии, в которой была бы информация о дисциплинах изученных Вами, преподавателях, которым Вы сдавали зачеты и экзамены, о форме отчетности (зачет или экзамен) и полученных оценках. Для ввода документов используйте разные способы. Выведете документы, содержащие информацию о сданных экзаменах

Вопросы для защиты работы

1.Перечислите типы данных и укажите способы их записи

2.Перечислите команды и методы для работы с БД в MongoDB

3. Перечислите команды создания и обработки коллекции в БД

4. Как задается атрибут в документе?

5. Что такое массив и как он задается?

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения задания самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты

Лабораторная работа №2
Средства поиска данных в базе MongoDB

Целью запроса к базе MongoDB являются документы из определенной коллекции. Для поиска (отбора) документов предназначены методы коллекции find и findOne. Вызовы методов find и findOne имеют одинаковую структуру и содержат критерий отбора документов (селектор), имена выводимых атрибутов и правило сортировки выбранных документов. Различие методов в том, что метод find возвращает все документы, соответствующие критерию отбора, а метод findOne — только один (первый) документ, из удовлетворяющих селектору.

Следует выполнять команды, выделенные **жирным шрифтом**

Основной синтаксис вызова метода find ()имеет вид:

db.<Коллекция>.find ([{<Селектор>}

[, {<Список атрибутов>}]]);

В простейшем виде запрос db.<Коллекция>.find (); возвращает все документы заданной коллекции. Отсутствие селектора и списка атрибутов приводит к выводу всех документов с полными наборами их атрибутов. Параметр <список атрибутов> содержит имена выводимых или, напротив, исключаемых из вывода атрибутов найденных документов. Кроме параметров find и findOne имеют присоединенные методы, выполняющие сортировку, ограничение количества и способ вывода найденных документов. С присоединенными методами запрос имеет вид:

db.<Коллекция>.find ([{<Селектор>}

[, {<Список атрибутов>}]])

[.sort ({<Атрибут>: 1 |-1, …})]

[.limit (<Число выводимых документов>)]

[.skip (<Количество пропускаемых документов>)]

[.pretty ()]; — выводит каждый атрибут в отдельной строке.

Количество документов, удовлетворяющих запросу, возвращает метод коллекции count:

db.<Коллекция>.count ({<Селектор>}).

Вывод различных значений (удаление дубликатов) определенного атрибута в коллекции выполняет метод db.<Коллекция>.distinct (<Имя атрибута>). Например, запрос

db.authors.distinct ("Au_lname")

вернет массив из фамилий авторов, выбранных из атрибута Au_lname в документах коллекции authors

["Лесков", "Н. В. Гоголь"].

Селектор содержит критерий для выбора документов и записывается в формат JSON-объекта: {<Условия на значения атрибутов>}. Пустой селектор, который обеспечивает выборку всех документов из коллекции, задается отсутствием условия {} или Null. Отдельные условия и логические операции в селекторе также записываются в формате JSON.

Виды условий в селекторе

1.<Атрибут>:<Значение> соответствует условию <Атрибут>=<Значение>.

Например, поиск документа по фамилии автора:

db.authors.find ({"Au_lname": "Лесков"});

выдаст

{ "_id" : ObjectId("58695c396b9421d84b9efb93"), "Au_lname" : "Лесков", "Au_fname" : "Николай", "birthday" : ISODate("1831-02-16T00:00:00Z") }

2. Модификаторы сравнений в условии $lt (less than, <), $lte (less than or equal, <=), $gt (greater than, >), $gte (greater than or equal, >=), $ne (not equal, #) задают условие выбора документов в форме неравенства. Для сохранения формата условия модификатор и значение для сравнения записывается в виде JSON-объекта:

{<Модификатор>: <Значение для сравнения>}.

Для демонстрации данного способа создадим коллекцию number

> db.createCollection ("number")

Для включения в коллекцию документов используем метод save()

```
for(i=0; i<25; i++) {
   db.number.save({num: i});
   }
```

В результате получим коллекцию

> db.number.find()

{ "_id" : ObjectId("586d240aa1e9f68581a87062"), "num" : 0 }

{ "_id" : ObjectId("586d240aa1e9f68581a87063"), "num" : 1 }
{ "_id" : ObjectId("586d240aa1e9f68581a87064"), "num" : 2 }
{ "_id" : ObjectId("586d240aa1e9f68581a87065"), "num" : 3 }
{ "_id" : ObjectId("586d240aa1e9f68581a87066"), "num" : 4 }
{ "_id" : ObjectId("586d240aa1e9f68581a87067"), "num" : 5 }
{ "_id" : ObjectId("586d240aa1e9f68581a87068"), "num" : 6 }
{ "_id" : ObjectId("586d240aa1e9f68581a87069"), "num" : 7 }
{ "_id" : ObjectId("586d240aa1e9f68581a8706a"), "num" : 8 }
{ "_id" : ObjectId("586d240aa1e9f68581a8706b"), "num" : 9 }
{ "_id" : ObjectId("586d240aa1e9f68581a8706c"), "num" : 10 }
{ "_id" : ObjectId("586d240aa1e9f68581a8706d"), "num" : 11 }
{ "_id" : ObjectId("586d240aa1e9f68581a8706e"), "num" : 12 }
{ "_id" : ObjectId("586d240aa1e9f68581a8706f"), "num" : 13 }
{ "_id" : ObjectId("586d240aa1e9f68581a87070"), "num" : 14 }
{ "_id" : ObjectId("586d240aa1e9f68581a87071"), "num" : 15 }
{ "_id" : ObjectId("586d240aa1e9f68581a87072"), "num" : 16 }

{ "_id" : ObjectId("586d240aa1e9f68581a87073"), "num" : 17 }

{ "_id" : ObjectId("586d240aa1e9f68581a87074"), "num" : 18 }

{ "_id" : ObjectId("586d240aa1e9f68581a87075"), "num" : 19 }

Выводится первые 20 результатов. Чтобы вывести следующую порцию, надо выполнить команду

> **it**

{ "_id" : ObjectId("586d240aa1e9f68581a87076"), "num" : 20 }

{ "_id" : ObjectId("586d240aa1e9f68581a87077"), "num" : 21 }

{ "_id" : ObjectId("586d240aa1e9f68581a87078"), "num" : 22 }

{ "_id" : ObjectId("586d240aa1e9f68581a87079"), "num" : 23 }

{ "_id" : ObjectId("586d240aa1e9f68581a8707a"), "num" : 24 }

Чтобы запросить документы, для которых значение num не равно 5, следует ввести

> **db.number.find({num: {"$ne":5}})**

{ "_id" : ObjectId("586d240aa1e9f68581a87062"), "num" : 0 }

{ "_id" : ObjectId("586d240aa1e9f68581a87063"), "num" : 1 }

{ "_id" : ObjectId("586d240aa1e9f68581a87064"), "num" : 2 }

{ "_id" : ObjectId("586d240aa1e9f68581a87065"), "num" : 3 }

{ "_id" : ObjectId("586d240aa1e9f68581a87066"), "num" : 4 }

{ "_id" : ObjectId("586d240aa1e9f68581a87068"), "num" : 6 }

{ "_id" : ObjectId("586d240aa1e9f68581a87069"), "num" : 7 }

{ "_id" : ObjectId("586d240aa1e9f68581a8706a"), "num" : 8 }

{ "_id" : ObjectId("586d240aa1e9f68581a8706b"), "num" : 9 }

{ "_id" : ObjectId("586d240aa1e9f68581a8706c"), "num" : 10 }

{ "_id" : ObjectId("586d240aa1e9f68581a8706d"), "num" : 11 }

{ "_id" : ObjectId("586d240aa1e9f68581a8706e"), "num" : 12 }

{ "_id" : ObjectId("586d240aa1e9f68581a8706f"), "num" : 13 }

{ "_id" : ObjectId("586d240aa1e9f68581a87070"), "num" : 14 }

{ "_id" : ObjectId("586d240aa1e9f68581a87071"), "num" : 15 }

{ "_id" : ObjectId("586d240aa1e9f68581a87072"), "num" : 16 }

{ "_id" : ObjectId("586d240aa1e9f68581a87073"), "num" : 17 }

{ "_id" : ObjectId("586d240aa1e9f68581a87074"), "num" : 18 }

{ "_id" : ObjectId("586d240aa1e9f68581a87075"), "num" : 19 }

{ "_id" : ObjectId("586d240aa1e9f68581a87076"), "num" : 20 }

Type "it" for more

Чтобы запросить документы, для которых значение num меньше равно 5, следует ввести

> **db.number.find({num: {"$lte":5}})**

{ "_id" : ObjectId("586d240aa1e9f68581a87062"), "num" : 0 }

{ "_id" : ObjectId("586d240aa1e9f68581a87063"), "num" : 1 }

{ "_id" : ObjectId("586d240aa1e9f68581a87064"), "num" : 2 }

{ "_id" : ObjectId("586d240aa1e9f68581a87065"), "num" : 3 }

{ "_id" : ObjectId("586d240aa1e9f68581a87066"), "num" : 4 }

{ "_id" : ObjectId("586d240aa1e9f68581a87067"), "num" : 5 }

3. Инвертирование (отрицание) условия $not. <Атрибут>:{$not: {<Сравнение>}}}. Модификатор $not применим только с модификаторами сравнения $lt, $lte и т. д. Условию с модификатором $not также удовлетворяют документы, не имеющие проверяемого атрибута.

Например,

<db.number.find({num: {$not:{"$gte":5}}})

{ "_id" : ObjectId("586d240aa1e9f68581a87062"), "num" : 0 }

{ "_id" : ObjectId("586d240aa1e9f68581a87063"), "num" : 1 }

{ "_id" : ObjectId("586d240aa1e9f68581a87064"), "num" : 2 }

{ "_id" : ObjectId("586d240aa1e9f68581a87065"), "num" : 3 }

{ "_id" : ObjectId("586d240aa1e9f68581a87066"), "num" : 4 }

4. $regex — регулярные выражения в селекторе (поиск по строковым атрибутам).

Создадим еще одну коллекцию-список

> db.createCollection ("список")

Добавим в коллекцию документы

> db.список.insert({"Фамилия и инициалы":"Котов А В","Цех":"Механический","Табельный номер":"01"})

WriteResult({ "nInserted" : 1 })

> db.список.insert({"Фамилия и инициалы":"Котаев П В","Цех":"Сборочный","Табельный номер":"02"})

WriteResult({ "nInserted" : 1 })

> db.список.insert({"Фамилия и инициалы":"Светин П С","Цех":"Сборочный","Табельный номер":"03"})

WriteResult({ "nInserted" : 1 })

> db.список.insert({"Фамилия и инициалы":"Светов В В","Цех":"Литейный","Табельный номер":"03"})

WriteResult({ "nInserted" : 1 })

> db.список.insert({"Фамилия и инициалы":"Ветров П В","Цех":"Сборочный","Табельный номер":"05"})

WriteResult({ "nInserted" : 1 })

Краткая запись условия с регулярным выражением: <Строковый атрибут>:/<Шаблон>/<Опции i и m>,

где шаблон — искомый контекст.

Запрос

> db.список.find ({"Фамилия и инициалы":/Кот/});

выдаст результат

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический", "Табельный номер" : "01" }

{ "_id" : ObjectId("586d58bc62ed48682494547f"), "Фамилия и инициалы" : "Котаев П В", "Цех" : "Сборочный", "Табельный номер" : "02" }

Опции: i — сравнение выполняется без учета регистра символов,

m — сравнение применяется к многострочным текстам.

Например,

запрос

> db.список.find ({"Фамилия и инициалы":/ве/i});

находит

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d597762ed486824945481"), "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d59bf62ed486824945482"), "Фамилия и инициалы" : "Ветров П В", "Цех" : "Сборочный", "Табельный номер" : "05" }

5. $in — проверка совпадения значения атрибута с одним из элементов в массиве: {<Атрибут>: {$in: [<Список значений>]}.

Например:

<db.список.find({"Цех":{$in:["Механический","Литейный"]}})

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический", "Табельный номер" : "01" }

{ "_id" : ObjectId("586d597762ed486824945481"), "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный", "Табельный номер" : "03" }

6. $nin — проверка отсутствия определенных значений атрибута или самого атрибута (альтернатива для $in).

<Атрибут>: {$nin: [<Список значений>]} находит документы, в которых заданный атрибут не совпадает ни с

одним из элементов в списке значений или документ не имеет проверяемого атрибута.

Например:

> db.список.find({"Цех":{$in:["Механический","Столярный"]}})

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический", "Табельный номер" : "01" }

7. Выбор документов, одновременно удовлетворяющих нескольким условиям. В краткой форме отдельные условия задаются в общем списке:

<Атрибут1>:<Значение1>, <Атрибут2> : <Значение2>,... Заданные списком условия связываются по «И».

Например, запрос

> db.список.find ({"Цех":"Сборочный","Табельный номер":"03"})

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный", "Табельный номер" : "03" }.

Полная (в JSON) форма связывания по «И» использует модификатор $and с массивом условий: $and: [{<Условие1>}, …]

Например:

> db.список.find ({$and: [{"Цех": "Сборочный"}, {"Табельный номер": "03"}]})

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный", "Табельный номер" : "03" }

8. Связывание условий по «ИЛИ» $or: [<Список условий>].

Например,

> db.список.find ({$or: [{"Цех":"Механический"}, {"Табельный номер":"05"}]})

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический", "Табельный номер" : "01" }

{ "_id" : ObjectId("586d59bf62ed486824945482"), "Фамилия и инициалы" : "Ветров П В", "Цех" : "Сборочный", "Табельный номер" : "05" }

9. Связывание условий по «инвертированному ИЛИ»: $nor:

[<Список условий >] — выбирает документы, в которых не выполняются все условия или отсутствует атрибут, участвующий в условии.

Например, запрос:

> db.список.find ({$nor: [{"Цех":"Механический"}, {"Табельный номер":"05"}]})

дает результат

{ "_id" : ObjectId("586d58bc62ed48682494547f"), "Фамилия и инициалы" : "Котаев П В", "Цех" : "Сборочный", "Табельный номер" : "02" }

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d597762ed486824945481"), "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный", "Табельный номер" : "03" }

10. Поиск документов по наличию/отсутствию атрибута:

<Атрибут>: {$exists: false|true}.

По условию <Атрибут>: {$exists: false} находятся те документы, в которых проверяемый атрибут отсутствует или его значение не задано (null). Например, запрос

> db.список.find({"Год_рожд":{$exists: false}})

выдает все документы, так как у них отсутствует атрибут **Год_рожд**

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический", "Табельный номер" : "01" }

{ "_id" : ObjectId("586d58bc62ed48682494547f"), "Фамилия и инициалы" : "Котаев П В", "Цех" : "Сборочный", "Табельный номер" : "02" }

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d597762ed486824945481"), "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d59bf62ed486824945482"), "Фамилия и инициалы" : "Ветров П В", "Цех" : "Сборочный", "Табельный номер" : "05" }

db.authors.find ({"Год_рожд":{$exists: false}}) находит авторов, у которых не задан атрибут "Год_рожд".

11. Проверка типа для значения атрибута: {<атрибут>: {$type: <код BSON-типа>}}. Выбирает документы, у которых указанный атрибут имеет заданный тип.

Основные коды BSON-типов:

Double — 1 (числовой, задает целые числа и числа с фиксированной точкой), String — 2,

Boolean — 8,

Date — 9,

null — 10.

Например,

> db.список.find ({"Цех": {$type: 2}})

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический", "Табельный номер" : "01" }

{ "_id" : ObjectId("586d58bc62ed48682494547f"), "Фамилия и инициалы" : "Котаев П В", "Цех" : "Сборочный", "Табельный номер" : "02" }

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d597762ed486824945481"), "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный", "Табельный номер" : "03" }

{ "_id" : ObjectId("586d59bf62ed486824945482"), "Фамилия и инициалы" : "Ветров П В", "Цех" : "Сборочный", "Табельный номер" : "05" }

выберет все документы, так как у них атрибут "Год_рожд" задан строкой текста.

Управление выводом атрибутов в запросе

Список выводимых атрибутов задается вторым параметром в методах find и findOne. В параметре атрибут, который необходимо вывести, указывается со значением 1, а не требуемый для вывода — 0.

db.<коллекция>.find ([{<селектор>} [, {<имя атрибута>: 1 |0, …}]]);

Например, запрос

> db.список.find ({},{"Фамилия и инициалы": 1, "Цех":1})

{ "_id" : ObjectId("586d583062ed48682494547e"), "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический" }

{ "_id" : ObjectId("586d58bc62ed48682494547f"), "Фамилия и инициалы" : "Котаев П В", "Цех" : "Сборочный" }

{ "_id" : ObjectId("586d591162ed486824945480"), "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный" }

{ "_id" : ObjectId("586d597762ed486824945481"), "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный" }

{ "_id" : ObjectId("586d59bf62ed486824945482"), "Фамилия и инициалы" : "Ветров П В", "Цех" : "Сборочный" }

выведет из всех документов атрибуты **Фамилия и инициалы** , **Цех** и ключ _id, который выводится по умолчанию, если его вывод не отключен явно.

Если в списке перечислены только атрибуты, не требующие вывода, то незаданные атрибуты выводятся по умолчанию.

Например, запрос

> db.список.find ({}, {_id: 0, "Цех": 0})

{ "Фамилия и инициалы" : "Котов А В", "Табельный номер" : "01" }

{ "Фамилия и инициалы" : "Котаев П В", "Табельный номер" : "02" }

{ "Фамилия и инициалы" : "Светин П С", "Табельный номер" : "03" }

{ "Фамилия и инициалы" : "Светов В В", "Табельный номер" : "03" }

{ "Фамилия и инициалы" : "Ветров П В", "Табельный номер" : "05" }

выведет все атрибуты, кроме _id и **Цех**, из всех документов.

В одном запросе не допускается смешивать выводимые и невыводимые атрибуты документа. Так, запрос db.authors. find ({}, {"Au_lname": 1, "Au_fname": 0}) является ошибочным. Исключение составляет атрибут _id — идентификатор документов.

Сортировка документов в запросе

Для сортировки результата запроса find () используется присоединенный метод db.<Коллекция>.find ().sort ({<Атрибут>: 1 |-1,..}). Указание атрибута со значением 1 приводит к сортировке по возрастанию, —1 — по убыванию значений атрибута.

Например, запрос

> db.список.find ({}, {_id:0,"Фамилия и инициалы": 1, "Цех": 1}).sort ({"Фамилия и инициалы": 1})

{ "Фамилия и инициалы" : "Ветров П В", "Цех" : "Сборочный" }

{ "Фамилия и инициалы" : "Котаев П В", "Цех" : "Сборочный" }

{ "Фамилия и инициалы" : "Котов А В", "Цех" : "Механический" }

{ "Фамилия и инициалы" : "Светин П С", "Цех" : "Сборочный" }

{ "Фамилия и инициалы" : "Светов В В", "Цех" : "Литейный" }

выведет фамилии , отсортированные по возрастанию фамилии.

Ограничение множества выводимых документов

Для ограничения количества выводимых документов используются присоединенные методы:

· limit (<Количество выводимых документов>),

· skip (<Количество пропускаемых в начале документов>).

Например, запрос

> db.список.find (null, {_id:0, "Фамилия и инициалы":1}).limit (3).skip (2)

{ "Фамилия и инициалы" : "Светин П С" }

{ "Фамилия и инициалы" : "Светов В В" }

{ "Фамилия и инициалы" : "Ветров П В" }

выведет из коллекции пять первых фамилий , пропустив две первых фамилии.

Задание для самостоятельной работы

Создать базу данных с коллекцией, содержащей результаты реализации канцелярских товаров различными магазинами. Каждый документ должен содержать следующие атрибуты: название магазина, наименование товара, количество поступивших на реализацию единиц товара, количество проданных единиц товара, цена товара. Составить следующие запросы:

1 запрос содержащий результаты реализации указанного (по выбору студента) товара

2. запрос содержащий результаты реализации указанного (по выбору студента) товара по цене ниже средней (среднюю цену товара вычислить самостоятельно)

3. запрос содержащий результаты реализации указанного (по выбору студента) товара в указанных (по выбору студента) магазинах

4. запрос, содержащий только название магазина, наименование товара и цену товара

5. запрос, содержащий все атрибуты, в котором название магазинов расположены в алфавитном порядке

Коллекция должна содержать не менее пятнадцати документов. Название базы данных должно содержать фамилию студента.

Вопросы для защиты работы

1.Основной синтаксис вызова метода find ()

2. Виды условий в селекторе

3. Управление выводом атрибутов в запросе

4. Сортировка документов в запросе

5. Методы, используемые для ограничения количества выводимых документов

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной

работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения заданий самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты

Лабораторная работа №3

Операции с документами

Удаление документов

Следует выполнять команды, выделенные **жирным шрифтом**

Для дальнейшего изучения возможностей Mongo создадим новую базу данных

> use База1,

а в ней коллекцию Авторы

> db.createCollection(Авторы)

В созданную коллекцию введем документы

> db.Авторы.insert ({"Au_lname":"Лесков", "Au_fname": "Николай", "birthday": ISODate ("1831-02-16")})

> db.Авторы.insert ({_id: "000–11–1111", "Au_ lname": "Пелевин", " Au_fname ": "Ви´ктор", "year_of_birth": 1962, Contract: true})

> db.Авторы.insert({"Au_lname":"Н. В. Гоголь", "year_of_ birth":1809})

Для удаления отдельных документов из коллекции предназначен перегружаемый метод remove. При вызове метода с одним параметром db.<Коллекция >.remove ({<Селектор>}) происходит удаление всех документов, удовлетворяющих селектору. А в отсутствие селектора из коллекции удаляются все документы. При вызове с двумя параметрами db.<Коллекция>.remove ({<Селектор>}, true) удаляется только один (первый) документ из соответствующих селектору.

Например, команда

> db.Авторы.remove ({"year_of_birth":{$gt:1907}})

удалит документы с годом рождения авторов, большим, чем 1907.

Убедимся в этом

> db.Авторы.find()

{ "_id" : ObjectId("586f5cc203ef629951278f6d"), "Au_lname" : "Лесков", "Au_fname" : "Николай", "birthday" : ISODate("1831-02-16T00:00:00Z") }

{ "_id" : ObjectId("586f5f4303ef629951278f6e"), "Au_lname" : "Н. В. Гоголь", "year_of_ birth" : 1809 }

Изменение документов

Изменения в документы коллекции вносятся методом update.

Метод может изменить определенные атрибуты одного или нескольких существующих документов или полностью заменить

существующий документ новым. Вид выполняемого действия определяется используемыми параметрами и их модификаторами. Базовый вызов метода update имеет вид:

db. <Коллекция>.update ({<Селектор>}, {<Новый документ или модификатор с изменяемыми атрибутами>}, {<Опции обновления>});

Полная замена документа, удовлетворяющего селектору, выполняется, если новые значения атрибутов заданы без использования модификаторов

Пусть в коллекции Авторы имеется документ:

> db.Авторы.insert({Au_id: "0007", lname: "Гоголь", Years: [1809, 1852]})

Применение метода

db.authors.update ({Au_id:"0007"}, {Au_ lname:"Гоголь", Years: [1809, 1852], Books: ["Вий", "Тарас Бульба"]})

приведет к замене выбранного селектором {Au_id:"0007"} документа новым.

Заносим в переменную doc новый документ:

> var doc = {Au_id: "0007", Au_lname: "Н. В. Гоголь", Years: [1809, 1852]}

Выполняем замену документа:

> db.Авторы.update ({Au_id: "0007"}, doc)

Убедимся в том, что операция выполнена

> db.Авторы.find()

{ "_id" : ObjectId("586f69584652a7f8a3964bd7"), "Au_lname" : "Лесков", "Au_fname" : "Николай", "birthday" : ISODate("1831-02-16T00:00:00Z") }

{ "_id" : ObjectId("586f69a44652a7f8a3964bd8"), "Au_lname" : "Н. В. Гоголь", "year_of_ birth" : 1809 }

{ "_id" : ObjectId("586f6ce34652a7f8a3964bd9"), "Au_id" : "0007", "Au_lname" : "Н. В. Гоголь", "Years" : [1809, 1852] }

Третий параметр «Опции обновления» влияет на «поведение» метода update. Опция upsert (типа boolean) со значением true при отсутствии документа, удовлетворяющие условию селектора, добавляет документ с заданными атрибутами. При значении false новый документ не создается. По умолчанию действует false. Опция multi (типа boolean): по значению true обновляются все документы, удовлетворяющего условию селектора. В значении false обновляет один документ. Значение по умолчанию false. Добавление, удаление или изменение атрибутов в документе задается модификаторами, устанавливаемыми перед их значениями. Изменяемые атрибуты записываются в виде:

<Модификатор>:{<Атрибут>:<Значение>},..

Ниже приводится сводная таблица модификаторов, используемых в методе update ().

Модификатор	Описание
$set	Обновляет, а при отсутствии — создает атрибут
$unset	Удаляет атрибут
$inc	Увеличивает значение атрибута на заданное число
$pop	При значении 1 удаляет последний, при –1 — первый элемент массива
$push	Добавляет в массив новый элемент
$pushAll	Помещает несколько новых элементов в массив
$addToSet	Добавляет новый элемент в массив (исключаются дубликаты)
$pull	Удаляет из массива значение (при его наличии)
$pullAll	Удаляет из массива все подходящие значения

Рассмотрим использование модификаторов внесения изменений.

Пусть в коллекции Авторы находится следующий документ:

{"Au_id": "000–11–0009", "LName": "Толстой", "Lang": "Русский", "Years": [1882, 1945], "Product": ["Аэлита", "Князь Серебряный"]}

> db.Авторы.insert({"Au_id": "000–11–0009", "LName": "Толстой", "Lang": "Русский", "Years": [1882, 1945], "Product": ["Аэлита", "Князь Серебряный"]})

> db.Авторы.insert({"Au_id": "000–11–0009", "Au_lname": "Толстой", "Years": [1882, 1945], "Books": ["Аэлита", "Князь Серебряный"]})

1. Замена всего документа

<db.authors.update ({Au_id: "000–11–0009"}, {Au_ id: "000–11–0009", Au_lname: "Толстой", Years: [1882, 1945], Books: ["Аэлита", "Князь Серебряный"]})

2. Добавление нового атрибута. Модификатор $set добавляет атрибут, если его не было в документе:

> db.authors.update ({Au_id: "000–11–0009"}, {$set:{"Страна": "СССР"}})

Запрос

> db.Авторы.find({Au_id: "000–11–0009"})

{ "_id" : ObjectId("586f7ddf4a7bd4ba7abfc2dc"), "Au_id" : "000–11–0009", "Au_lname" : "Толстой", "Years" : [1882, 1945], "Books" : ["Аэлита", "Князь Серебряный"], "Страна" : "СССР" }

подтверждает внесенные изменения

3. Изменение значения атрибута. Модификатор $set изменяет значение атрибута, если такой атрибут существует в документе.

Уточняем автора:

> db.Авторы.update ({Au_id: "000–11–0009"}, {$set:{Au_lname: "А. Н. Толстой"}})

Проверяем соответствующим запросом

> db.Авторы.find({Au_id: "000–11–0009"})

{ "_id" : ObjectId("586f7ddf4a7bd4ba7abfc2dc"), "Au_id" : "000–11–0009", "Au_lname" : "А. Н. Толстой", "Years" : [1882, 1945], "Books" : ["Аэлита", "Князь Серебряный"], "Страна" : "СССР" }

4. Удаление ошибочного элемента "Князь Серебряный" из атрибута-массива "Books":

> db.Авторы.update ({Au_id: "000–11–0009"}, {$pull:{"Books":"Князь Серебряный"}})

Проверяем соответствующим запросом

<db.Авторы.find({Au_id: "000–11–0009"})

{ "_id" : ObjectId("586f7ddf4a7bd4ba7abfc2dc"), "Au_id" : "000–11–0009", "Au_lname" : "А. Н. Толстой", "Years" : [1882, 1945], "Books" : ["Аэлита"], "Страна" : "СССР" }

5. Добавление нового элемента "Петр Первый" в массив "Books"

db.Авторы.update ({Au_id: "000–11–0009"}, {$push:{"Books": "Петр Первый"}})

Проверяем соответствующим запросом

> db.Авторы.find({Au_id: "000–11–0009"})

{ "_id" : ObjectId("586f7ddf4a7bd4ba7abfc2dc"), "Au_id" : "000–11–0009", "Au_lname" : "А. Н. Толстой", "Years" : [1882, 1945], "Books" : ["Аэлита", "Петр Первый"], "Страна" : "СССР" }

Добавление или замена документа в коллекции — метод save

db.<Коллекция>.save (<Документ>)

Если указанный в параметре документ не содержит атрибут _id, то выполняется добавление документа с созданием уникального идентификатора _id.

Если атрибут _id в документе задан и он совпадает с _id документа в коллекции, то выполняется замена существующего документа, иначе добавляется новый документ с заданным ключом _id.

По аналогии с методами insert и update параметром метода save может быть содержащая документ переменная.

Например,

> var doc = {Au_id: "0008", Au_lname: "А. С. Пушкин", Years: [1799, 1837]};

> db.Авторы.save (doc)

> db.Авторы.find()

{ "_id" : ObjectId("586f69584652a7f8a3964bd7"), "Au_lname" : "Лесков", "Au_fname" : "Николай", "birthday" : ISODate("1831-02-16T00:00:00Z") }

{ "_id" : ObjectId("586f69a44652a7f8a3964bd8"), "Au_lname" : "Н. В. Гоголь", "year_of_ birth" : 1809 }

{ "_id" : ObjectId("586f6ce34652a7f8a3964bd9"), "Au_id" : "0007", "Au_lname" : "Н. В. Гоголь", "Years" : [1809, 1852] }

{ "_id" : ObjectId("586f7ddf4a7bd4ba7abfc2dc"), "Au_id" : "000–11–0009", "Au_lname" : "А. Н. Толстой", "Years" : [1882, 1945], "Books" : ["Аэлита", "Петр Первый"], "Страна" : "СССР" }

{ "_id" : ObjectId("586f87fa4a7bd4ba7abfc2dd"), "Au_id" : "0008", "Au_lname" : "А. С. Пушкин", "Years" : [1799, 1837] }

Задание для самостоятельной работы

Создать базу данных с коллекцией, содержащей информацию о Ваших друзьях, родственниках и знакомых. Каждый документ должен содержать следующие атрибуты: Фамилия и инициалы, номер телефона, год рождения, город проживания, адрес (улица, номер дома, номер квартиры). Имя коллекции должно совпадать с Вашей фамилией. Атрибут-массив "номер телефона" в некоторых документов должен содержать более двух значений. Коллекция должна содержать не менее 15 документов. Для добавления документов в коллекцию следует использовать разные методы (insert, save,…). Выполните следующие действия

1.Добавьте в документы еще один атрибут – страна проживания.

2.Удалите один из телефонов из атрибут-массивов некоторых документах

3.Добавте номер телефона в атрибут-массивы некоторых документах

4. Изменение значения атрибута "город проживания" в некоторых документах

5. Удалите документы, у которых атрибут "год рождения" превосходит Ваш год рождения

Вопросы для защиты работы

1.Синтаксис метода отдельных документов

2. Какой имеет вид базовый вызов метода update

3. Какие модификаторы, используются в методе update

4. В каком виде записываются изменяемые атрибуты в методе update

5. Синтаксис метода save

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения заданий самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты.

Лабораторная работа №4

Конвейерная обработка документов коллекции

Следует выполнять команды, выделенные **жирным шрифтом**

В MongoDB реализован инструмент для выполнения цепочки операторов обработки документов коллекции — фреймворк агрегирования данных. На вход агрегирования поступают исходные документы коллекции, для которых последовательно выполняется цепочка операций. Документы на выходе одной операции поступают на вход следующей. Результат последней операции является результатом всей обработки. Конвейерную обработку выполняет метод aggregate: db.<Коллекция>.aggregate ([<Операция агрегации>,..]);¶

<Операция агрегации> содержит оператор — выполняемое действие и аргументы — атрибуты обрабатываемых входных документов, записанные в форме JSON-объекта:

{<Оператор>: [<Аргумент1>, < Аргумент2> …]}

или для одиночного аргумента операция агрегации имеет вид:

{<Оператор>: <Аргумент>}

Метод aggregate возвращает единственный документ с атрибутом "result", содержащий массив документов, элементы которого являются результатом агрегатной обработки исходных документов.

В цепочке могут использоваться следующие операции агрегации.

1. {$match: {<Селектор>}} Оператор $match фильтрует входные документы, отбирая соответствующие заданному селектору. В

селекторе используются те же условия отбора, что и в методе find.

2. {$project: {<Спецификация атрибута>,...}} управляет набором атрибутов в выходных документах, добавляя новые, удаляя и переименовывая атрибуты входных документов. <Спецификация атрибута> имеет следующие формы:

· <Атрибут входного документа>:1 включает атрибут входного документа в выходной документ;

· _id: 0 исключает вывод идентификатора документа (по умолчанию идентификаторы выводятся);

· <Новый атрибут >: <Выражение для значения атрибута> используется для создания новых атрибутов, значения которых вычисляются из атрибутов входного документа.

<Выражение для значения атрибута> записывается в форме JSON-объекта, содержащего массив:

{<Операция>: [<Операнд1>, <Операнд2>…]}

или {<Операция: <Единственный операнд>}.

Тип операции зависит от типа операндов.

Поддерживаются следующие типы операций:

· булевские ($and, $or, $not),

· строковые ($concat, $substr, $toUpper и т. д.),

· арифметические ($add, $multiply и т. д.),

· операции сравнения ($eg, $gt, $gte и т. д.) и другие типы.

Например, в агрегации db.authors.aggregate
({$match:{"Au_lname":/Лес/}},

{$project:{_id: 0,

"Автор": {$concat: ["$Au_lname"," ","$Au_ fname"]},

"birthday":1}});

задан конвейер из двух операций:

· $match выбирает из коллекции authors документы об авторах, в фамилии которых присутствует слог «Лес», и передает следующей операции;

· $project для выбранных авторов отменяет вывод ключа _id, создает новый атрибут "Автор", в котором соединяются ($concat) в одной строке фамилия и имя автора, затем выводится атрибут "birthday". Операция $concat работает начиная с версии 3.0.4.

3. Операция {$unwind: <Имя атрибута-массива>} «разворачивает» массив, создавая для каждого элемента указанного массива новый документ, в котором присутствуют все атрибуты исходного документа с одним из элементов разворачиваемого массива.

4. Операция группировки входных документов

{$group: {_id: <Группирующее выражение>,

<Агрегирующий атрибут1>:

{<Оператор агрегации1>: <Выражение агрегирования1>},
.....}}

$group действует по аналогии с group by в SQL-Select: выполняет группировку путем создания нового документа из нескольких входных документов с одинаковым значением группирующего выражения и вычисления агрегирующих атрибутов по значениям атрибутов документов в группе. Способ вычисления агрегирующих атрибутов задается оператором агрегации.

Операторы агрегации:

· {$sum: <Выражение>} возвращает сумму числовых значений заданного выражения, вычисленного на документах в группе; нечисловые значения пропускаются;

· {$avg: <Выражение>} вычисляет арифметическое среднее числовых значений заданного выражения на документах в группе;

· {$first: <Выражение>} ({$last: < выражение >}) возвращает значение выражения, вычисленного по первому (последнему) документу в группе; используется, если определен порядок документов в группе;

· {$max: <Выражение>} ({$min: < выражение >}) возвращает максимальное (минимальное) значение выражения по документам в группе;

· {$push: <Выражение>} создает массив из результатов вычисления выражения для каждого документа в группе; порядок элементов в массиве не определен; · {$addToSet: <Выражение>} создает массив из уникальных значений результатов вычисления выражения для каждого документа в группе.

5. Операция сортировки документов {$sort: {<Атрибут1>: <Порядок>,, …}} упорядочивает на выходе поток входных документов, выполняя ступенчатую сортировку по перечисленным

атрибутам. Порядок сортировки задается значениями: 1 — сортировка по возрастанию атрибута, 0 — по убыванию.

6. Операция {$skip: <Число документов>} задает количество первых пропускаемых документов. Передает оставшиеся документы следующей операции в конвейере.

7. Операция {$limit: <Число документов>} ограничивает число документов, передаваемых следующей операции в конвейере.

8. Операция {$out: "<Выходная коллекция>"} записывает итоговые документы конвейера агрегации в заданную коллекцию. Оператор $out должен быть последним этапом конвейера.

Рассмотрим пример конвейерной обработки коллекции документов. Пусть коллекция Writer содержит сведения о четырех писателях:

{Автор: "Кант И.", Страна: "Германия", Название: ["Критика чистого разума"], Тип:"Философия"},

{Автор: "Пелевин В. О.", Страна: "Россия", Название: ["Чапаев и Пустота"], Тип:"Художественная литература"},

{Автор: "Пелевин В. О.", Страна: "Россия", Название: ["Бетман Аполло"], Тип:"Фантастика"},

{Автор: "Лукьяненко С. В.", Страна: "Россия", Название: ["Ночной дозор", "Дневной дозор"], Тип:"Фантастика"}.

Построим конвейер для подсчета числа книг каждого типа для русских писателей. Процесс обработки требует фильтрации документов по селектору {Страна: "Россия"} и последующей группировки по атрибуту "Тип". В результате должны получиться документы следующего вида:

{"Тип": "Фантастика", "Количество": 3}.

Создадим новую базу данных

> use База4

и в ней новую коллекцию

Например, пусть в коллекции writer находятся 4 документа, содержащих данные о писателях и их книгах:

> db.createCollection("writer")

Добавьте указанные выше документы в коллекцию самостоятельно

Создание конвейера обработки по шагам

1. Выбор российских авторов.

> db.Writer.aggregate ({$match:{Страна:/Рос/}});

Получим массив из трех отфильтрованных документов:

{ "_id" : ObjectId("5872a038bcae9444dc182648"), "Автор" : "Пелевин В. О.", "Страна" : "Россия", "Название" : ["Чапаев и Пустота"], "Тип" : "Художественная литература" }

{ "_id" : ObjectId("5872a067bcae9444dc182649"), "Автор" : "Пелевин В. О.", "Страна" : "Россия", "Название" : ["Бетман Аполло"], "Тип" : "Фантастика" }

{ "_id" : ObjectId("5872a08bbcae9444dc18264a"), "Автор" : "Лукьяненко С. В.", "Страна" : "Россия", "Название" : ["Ночной дозор", "Дневной дозор"], "Тип" : "Фантастика" }

2. В коллекции, полученной фильтрацией, разворачиваем масив названий книг.

> db.Writer.aggregate ({$match:{Страна:/Рос/}}, {$unwind: "$Название"});

В результате для каждой книги формируется отдельный документ. Последний документ для автора Лукьяненко С. В., имеющего две книги, разворачивается в два документа этого автора:

{ "_id" : ObjectId("5872a038bcae9444dc182648"), "Автор" : "Пелевин В. О.", "Страна" : "Россия", "Название" : "Чапаев и Пустота", "Тип" : "Художественная литература" }

{ "_id" : ObjectId("5872a067bcae9444dc182649"), "Автор" : "Пелевин В. О.", "Страна" : "Россия", "Название" : "Бетман Аполло", "Тип" : "Фантастика" }

{ "_id" : ObjectId("5872a08bbcae9444dc18264a"), "Автор" : "Лукьяненко С. В.", "Страна" : "Россия", "Название" : "Ночной дозор", "Тип" : "Фантастика" }

{ "_id" : ObjectId("5872a08bbcae9444dc18264a"), "Автор" : "Лукьяненко С. В.", "Страна" : "Россия", "Название" : "Дневной дозор", "Тип" : "Фантастика" }

Документы, содержащие несколько массивов, могут быть развернуты отдельно по каждому массиву или по нескольким любым массивам.

Например, пусть в коллекцию добавляется документ с двумя массивами, "Название" и "Формы":

<db.Writer.insert ({Автор:"Пушкин А. С.", Страна: "Россия", Название: ["Руслан и Людмила", "Капитанская дочка"], Формы: ["Стихи", "Проза"]});

Тогда его последовательное разворачивание по обоим массивам

> db.Writer.aggregate ({$match:{Страна:/Рос/}}, {$unwind: "$Название"}, {$unwind: "$Формы"});

приводит к созданию документов, в которых представлены все пары элементов из этих массивов:

{ "_id" : ObjectId("5872a2febcae9444dc18264b"), "Автор" : "Пушкин А. С.", "Страна" : "Россия", "Название" : "Руслан и Людмила", "Формы" : "Стихи" }

{ "_id" : ObjectId("5872a2febcae9444dc18264b"), "Автор" : "Пушкин А. С.", "Страна" : "Россия", "Название" : "Руслан и Людмила", "Формы" : "Проза" }

{ "_id" : ObjectId("5872a2febcae9444dc18264b"), "Автор" : "Пушкин А. С.", "Страна" : "Россия", "Название" : "Капитанская дочка", "Формы" : "Стихи" }

{ "_id" : ObjectId("5872a2febcae9444dc18264b"), "Автор" : "Пушкин А. С.", "Страна" : "Россия", "Название" : "Капитанская дочка", "Формы" : "Проза" }

Обратите внимание, в результат попадают только те документы, которые содержат разворачиваемые массивы. В приведенном примере разворачивание продемонстрировало формирование документов с семантическим несоответствием названия и формы произведения ("Название": "Капитанская дочка",

"Формы": "Стихи"). Причина несоответствия в отрыве связанных по смыслу данных при размещении названия и формы произведения в разных массивах. Для исключения подобных ошибок связанные данные должны размещаться в одном массиве. Например, можно представить сведения о названии и форме произведения в виде вложенного массива:

<db.Writer.insert ({Автор:"Пушкин А. С.", Страна:"Россия", Произведение: [["Руслан и Людмила", "Стихи"], ["Капитанская дочка", "Проза"]]});

Теперь разворачивание внешнего массива "Произведение" дает верный результат**:**

db.Writer.find()

{ "_id" : ObjectId("58729ffebcae9444dc182647"), "Автор" : "Кант И.", "Страна" : "Германия", "Название" : ["Критика чистого разума"], "Тип" : "Философия" }

{ "_id" : ObjectId("5872a038bcae9444dc182648"), "Автор" : "Пелевин В. О.", "Страна" : "Россия", "Название" : ["Чапаев и Пустота"], "Тип" : "Художественная литература" }

{ "_id" : ObjectId("5872a067bcae9444dc182649"), "Автор" : "Пелевин В. О.", "Страна" : "Россия", "Название" : ["Бетман Аполло"], "Тип" : "Фантастика" }

{ "_id" : ObjectId("5872a08bbcae9444dc18264a"), "Автор" : "Лукьяненко С. В.", "Страна" : "Россия", "Название" : ["Ночной дозор", "Дневной дозор"], "Тип" : "Фантастика" }

{ "_id" : ObjectId("5872a2febcae9444dc18264b"), "Автор" : "Пушкин А. С.", "Страна" : "Россия", "Название" : ["Руслан и Людмила", "Капитанская дочка"], "Формы" : ["Стихи", "Проза"] }

{ "_id" : ObjectId("5872a4b7bcae9444dc18264c"), "Автор" : "Пушкин А. С.", "Страна" : "Россия", "Произведение" : [["Руслан и Людмила", "Стихи"], ["Капитанская дочка", "Проза"]] }

Так как новый документ о Пушкине А. С. не содержит массив "Название", он не повлияет на результаты следующих шагов.

3. Для подсчета количества книг каждого типа необходима группировка документов по типу книги. Для этого на третьем шаге в каждом документе, полученном разворачиванием массива "Название", оставляем необходимый для группировки атрибут "Тип" и добавляем новый атрибут "Количество" равным 1 во всех документах. На следующем шаге при группировке атрибут "Количество" будет суммироваться отдельно для каждого типа книг.

> db.Writer.aggregate ({$match:{Страна:/Рос/}}, {$unwind: "$Название"},{$project:{_id:0, "Тип":1, "Количество":{$add: [1]}}});

Получим

{ "Тип" : "Художественная литература", "Количество" : 1 }
{ "Тип" : "Фантастика", "Количество" : 1 }
{ "Тип" : "Фантастика", "Количество" : 1 }
{ "Тип" : "Фантастика", "Количество" : 1 }
{ "Количество" : 1 }
{ "Количество" : 1 }

4. Следующий шаг — группировка документов по значениям атрибута "Тип" с использованием оператора $sum для суммирования атрибута "Количество" для документов в группе:

> db.Writer.aggregate ({$match:{Страна:/Рос/}}, {$unwind: "$Название"}, {$project: {_id:0, "Тип":1, "Количество": {$add: [1]}}}, {$group: {_id: "$Тип", "Число книг": {$sum: "$Количество"}}});

{ "_id" : null, "Число книг" : 2 }
{ "_id" : "Фантастика", "Число книг" : 3 }
{ "_id" : "Художественная литература", "Число книг" : 1 }

Задание для самостоятельной работы

Используя приведенную ниже таблицу создать базу данных с коллекцией, содержащей данные таблицы. Документы коллекции должны иметь атрибуты, соответствующие заголовкам столбцов таблицы

Фамилия и инициалы	Страна	Год рождения	Картины	Техника рисования
Перов В Г	Россия	1834	Рыболов	Холст, масло
Саврасов А К	Россия	1830	Грачи прилетели, Поселок	Холст, масло
Шишкин И И	Россия	1832	Рожь, Осенний лес	Холст, масло
Репин И Е	Россия	1844	Диоген и мальчик,	Картон, масло
Поленов В Д	Россия	1844	Деревня Оку-лова гора	Картон, масло
Васнецов В М	Россия	1848	Жница, Автопортрет	Холст, масло
Суриков В И	Россия	1848	Боярыня Морозова	Холст, масло
Ван Гог	Голландия	1853	Звездная ночь	Холст, масло
Мане Эдуард	Франция	1832	Флейтист	Холст, масло
Сезан Поль	Франция	1839	Мальчик в красном жилете	Холст, масло

Создать конвейер обработки документов коллекции, выполнив следующие операции

1. Произвести выбор (операцию фильтрации) российских авторов.
2. В коллекции, полученной фильтрацией, разворачиваем массив названий книг.
3. Произвести группировку документов по технике рисования
4. Произвести группировку документов по значениям атрибута "Техника рисования" с использованием оператора $sum для суммирования атрибута "Количество" для документов в группе:

Вопросы для защиты работы

1. Синтаксис метода aggregate
2. Какие операции могут использоваться в цепочке в процессе агрегации данных?
3. Какие поддерживаются типы операций в процессе агрегации данных?
4. Перечислите операторы агрегации
5. Какие формы имеет<Спецификация атрибута> ?

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной

работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения заданий самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты

Лабораторная работа №5

МОДИФИКАТОРЫ МАССИВОВ. ПОЗИЦИОННЫЕ МОДИФИКАТОРЫ МАССИВОВ

Модификаторы массивов в запросах

Создадим новую базу данных и коллекцию

> use Basa

>db.createCollection ("Массивы");

>db.Массивы.insert({номер:"1","Цвет":["красный","белый","желтый"],Вектор:[1,2,3,4,5]});

>db.Массивы.insert({номер:"2","Цвет":["красный","зеленый","желтый"],Вектор:[5,2,3,4,0]});

>db.Массивы.insert({номер:"3","Цвет":["синий","белый","красный"],Вектор:[1,2,0,4,1]});

>db.Массивы.insert({номер:"3","Цвет":["розовый","синий","красный"],Вектор:[3,5,3,4,1]});

>db.Массивы.insert({номер:"3","Цвет":["зеленый","синий","красный"],Вектор:[3,5,3,5,2]});

Для добавления элемента в массив используется модификатор «$push», который используется как параметр метода «update». Синтаксис модификатора:

{ $push: { <массив>: <значение> },

где:

<массив> – массив, в который происходит добавление элемента;

<значение> – значение добавляемого элемента.

Пример использования модификатора:

> db.Массивы.update({номер: "1"},{$push: {Вектор: 9}});

Данный запрос добавит элемент «9» в массив «Вектор» первого документа.

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "белый", "желтый"], "Вектор" : [1, 2, 3, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4, 0] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [1, 2, 0, 4, 1] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Добавление двух чисел в первый документ с номером 3 выполняется командой

> db.Массивы.update({номер: "3"},{$push: {Вектор: {$each:[3,5]}}});

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "белый", "желтый"], "Вектор" : [1, 2, 3, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 0, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Для удаления элемента из массива используется модификатор «$pop».

Синтаксис модификатора**:** { $pop: { <поле>: <опц. удаления> } },

где:

<поле> – имя массива;

<опц. удаления> – 1 – если требуется удалить последний элемент в массиве, -1 – если требуется удалить первый элемент в массиве.

Удаление последнего элемента в массиве выглядит следующим образом:

> db.Массивы.update({номер: "2"},{$pop: {Вектор: 1}});

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "белый", "желтый"], "Вектор" : [1, 2, 3, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [1, 2, 0, 4, 1] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Удаление первого элемент в документе с номером 3 выполняется командой

> db.Массивы.update({номер: "3"},{$pop: {Вектор: -1}});

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "белый", "желтый"], "Вектор" : [1, 2, 3, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 0, 4, 1] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Модификатор «$push» добавляет элементы в массив, не проверяя их на уникальность. Для того чтобы добавить в массив только уникальные элементы используется модификатор «$addToSet». Синтаксис модификатора аналогичен синтаксису «$push».

Для удаления элемента массива по определенному критерию используется модификатор «$pull». Синтаксис модификатора:

{ $pull: { <массив>: <запрос> } },

где:

<массив> – имя массива;

<запрос> – запрос для отыскания требуемого элемента.

Например, для удаления элемента «3» из массива «Вектор» первого документа используется запрос:

> db.Массивы.update({ номер: "1" },{ $pull: { Вектор: 3 } });

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "белый", "желтый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 0, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Добавление зеленого цвета в массив "Цвет" первого документа выполняется следующим образом

>db.Массивы.update({номер: "1"},{$push: {Цвет: "зеленый"}});

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "белый", "желтый", "зеленый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 0, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Удаление "белый" в массиве "Цвет" документа с номером 1 можно выполнить следующим образом

> db.Массивы.update({ номер: "1" },{ $pull: { Цвет: "белый" } });

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый", "зеленый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 4] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 0, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Позиционные модификаторы массивов

Для доступа к конкретному элементу массива используется два способа: по конкретной позиции, и с помощью использования позиционного оператора (символ ‘$’).

Для доступа к элементу массива по номеру, после запроса на выборку документа в качестве имени изменяемого поля используется: «<массив>.<№ элемента>»

Например, для увеличения четвертого элемента массива "Вектор" на 10 во втором документе используется следующий запрос:

> db.Массивы.update({ номер: "2" }, {$inc: {"Вектор.3": 10}});

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый", "зеленый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 14] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 0, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Чтобы увеличить элемент массива "Вектор" равный "0" на семь следует выполнить команду

> db.Массивы.update({ "Вектор": 0 }, {$inc: {"Вектор.$": 7}});

> db.Массивы.find();

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый", "зеленый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый", "желтый"], "Вектор" : [5, 2, 3, 14] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 7, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Запросы в массивах

Иногда требуется составить запрос, в котором происходит поиск по массиву значений. В таком случае применяются запросы в массивах. Выше было показано, как можно отыскать элемент в массиве. Для того чтобы найти элемент массива Вектор равный «4», используется следующий запрос:

> db.Массивы.find({ "Вектор": 4 });

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый", "зеленый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 7, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий", "красный"], "Вектор" : [3, 5, 3, 4, 1] }

Для того чтобы выбрать документы больше, чем по одному элементу массива, можно использовать оператор «$all».

Синтаксис оператора:

{ <поле>: { $all: [<значение1>, <значение 2>, …]}},

где:

<поле> – поле, по которому проводится поиск;

<значение 1, 2, …> – значения массива, которые нужно найти.

Для получения документов, в которых массив содержит элементы «1», «2» и «5» используется запрос:

> db.Массивы.find({Вектор: {$all: [1, 2, 5]}});

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый", "зеленый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый", "красный"], "Вектор" : [2, 7, 4, 1, 3, 5] }

Для получения документов по полному совпадению элементов в массиве необходимо просто перечислить искомые элементы также как и при вставке:

> db.Массивы.find({Цвет: {$all: ["зеленый", "синий", "красный"]}});

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий", "красный"], "Вектор" : [3, 5, 3, 5, 2] }

Для управления количеством возвращаемых элементов в массиве используется оператор «$slice». Синтаксис оператора:

{$slice: <количество элементов>}.

Для возвращения элементов массива, начиная с его конца, оператору передается отрицательное значение, с начала – положительное.

Оператор $slice может действовать подобно «skip» и «limit», только в отношении массивов. В этом случае оператор имеет следующий синтаксис: {$slice: [<skip>, <limit>]},

где: <skip> – пропускаемое количество элементов;

<limit> – возвращаемое количество элементов.

В качестве примера рассмотрим запрос для получения первых двух элементов массива «Цвет», который выглядит следующим образом

> db.Массивы.find({}, {Цвет : {$slice: 2}});

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый"], "Вектор" : [5, 2, 3, 14] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый"], "Вектор" : [2, 7, 4, 1, 3, 5] }

{ "_id" : ObjectId("61deb983231b1bfb9aa06ef8"), "номер" : "3", "Цвет" : ["розовый", "синий"], "Вектор" : [3, 5, 3, 4, 1] }

{ "_id" : ObjectId("61deba30231b1bfb9aa06ef9"), "номер" : "3", "Цвет" : ["зеленый", "синий"], "Вектор" : [3, 5, 3, 5, 2] }

Вывод только первых 3 документов, удовлетворяющих данному требованию выполняется командой

> db.Массивы.find({}, {Цвет : {$slice: 2}}).limit(3);

{ "_id" : ObjectId("61deb88b231b1bfb9aa06ef5"), "номер" : "1", "Цвет" : ["красный", "желтый"], "Вектор" : [1, 2, 4, 5, 9] }

{ "_id" : ObjectId("61deb8bf231b1bfb9aa06ef6"), "номер" : "2", "Цвет" : ["красный", "зеленый"], "Вектор" : [5, 2, 3, 14] }

{ "_id" : ObjectId("61deb90b231b1bfb9aa06ef7"), "номер" : "3", "Цвет" : ["синий", "белый"], "Вектор" : [2, 7, 4, 1, 3, 5] }

Задание для самостоятельной работы

1.Создайте самостоятельно коллекцию, аналогичную рассмотренной выше и содержащей не менее двух массивов

2. Создайте 3 различных запроса для вставки данных в массив

3. Создайте 3 различных запроса, производящих обновление данных в массиве: как по позиции элемента в массиве, так и по его значению.

4. Создайте запросы, удаляющие элементы из массива: по позиции элемента в массиве и по его значению.

Вопросы для защиты работы

1. Приведите синтаксис «$push» и «$pop».

2. Каким образом можно вставить в массив несколько элементов? Приведите пример запроса.

3. Для чего используются модификаторы массивов в запросах?

4. Какие способы обновления данных в массиве вы знаете? Приведите примеры.

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной

работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения заданий самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты

Лабораторная работа №6

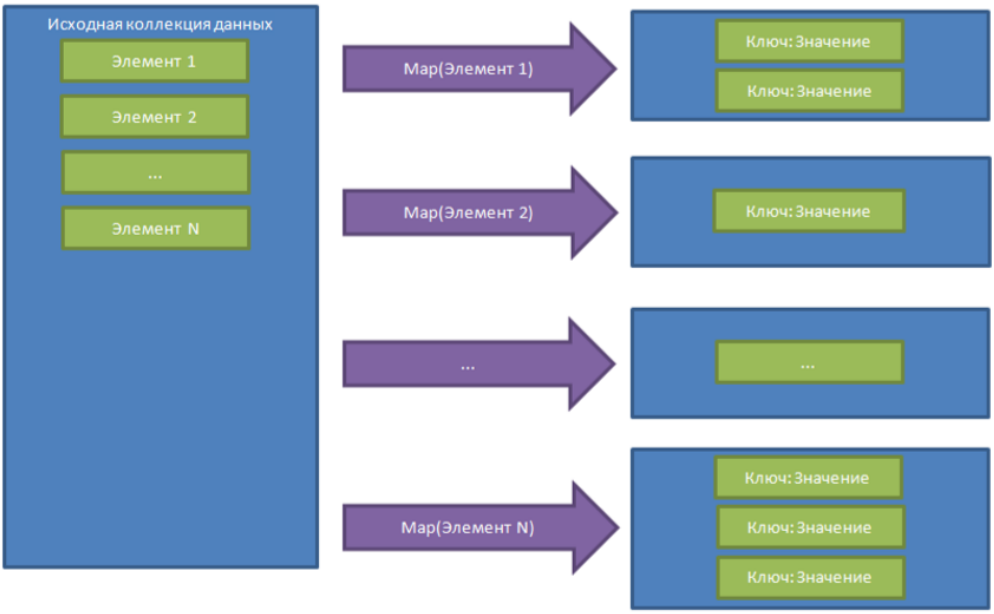
Распределенные вычисления

Цель лабораторной работы: изучить модель распределенных вычислений MapReduce. Задачи лабораторной работы: научиться выполнять распределенные операции над документами.

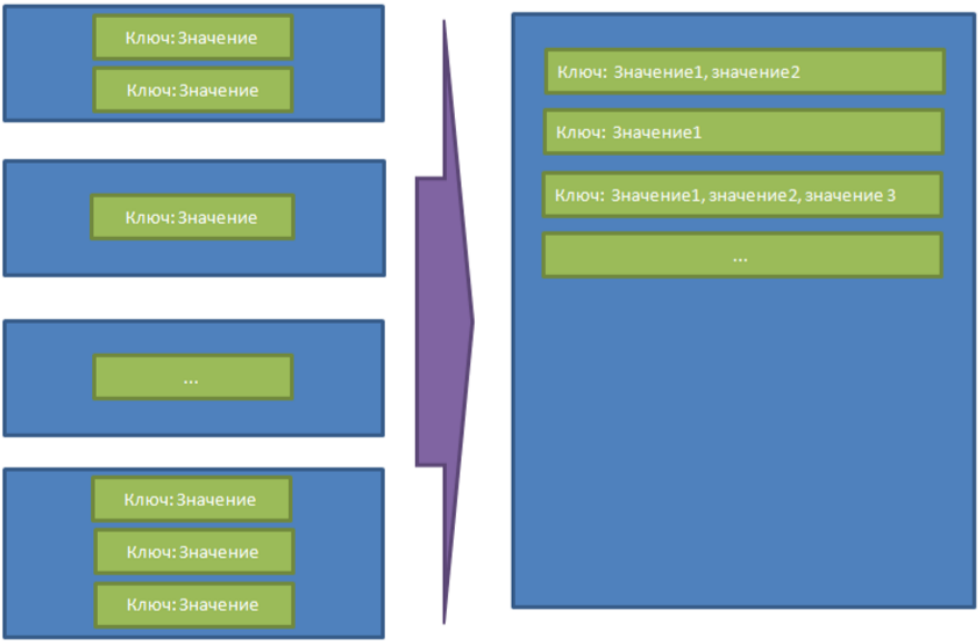
Теоретическое обоснование

MapReduce – это модель распределённых вычислений, параллельной обработки очень больших объемов данных (измеряемыми петабайтами), в компьютерных кластерах. Компьютеры, входящие в вычислительный кластер называются «узлами». Существуем два типа узлов: главный узел и рабочий. Работа MapReduce состоит из двух шагов: Map и Reduce (отображение и свертка). Мар-шаг производит предварительную обработку

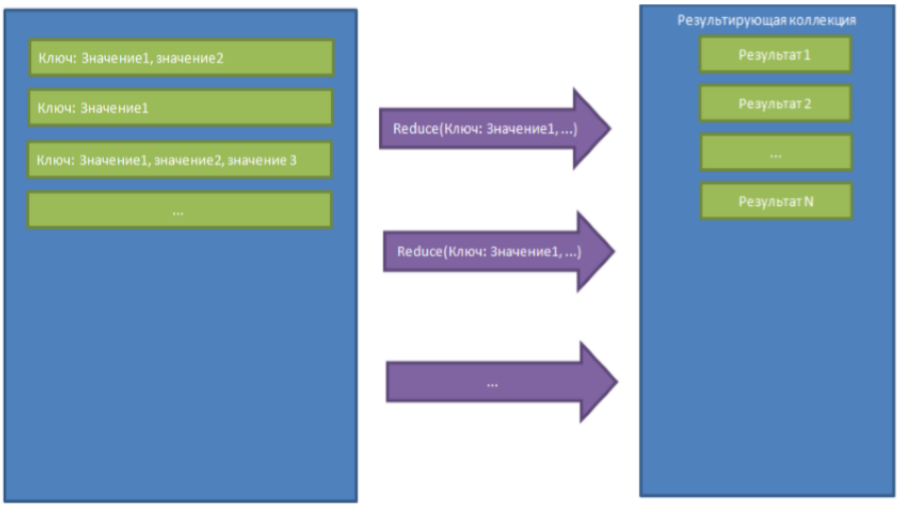
входных данных. Для этого главный узел получает входные данные задачи, разделяет их на части и передает рабочим узлам для предварительной обработки. На этом шаге входные данные преобразуются в пары ключ/значение (и ключ, и значение могут быть составными). Для этого функция Map применяется к каждому элементу исходной коллекции. Графически работа функции показана на рисунке



После первого шага, алгоритм сортирует все пары ключ/значение, и группирует все значения по ключу. Данный шаг продемонстрирован на рисунке ниже



На втором шаге (Reduce-шаг) происходит свертка предварительно обработанных данных. При свертке на входе получаются ключ и массив значений, порожденный для этого ключа, а на выходе – финальный результат. Функция Reduce выполнится для каждого сгруппированного экземпляра пары ключ/значение. Графически результат работы Reduce-шага представлен на рисунке



После завершения Reduce-шага главный узел получает ответы от рабочих узлов и на их основе формирует результат – решение задачи. И Map и Reduce шаги выполняются распределено на компьютерах кластера. Все операции независимы друг от друга и могут производиться параллельно. Основным преимуществом MapReduce по сравнению с традиционными решениями по обработке данных является производительность. Производительность достигается за счет распараллеливания вычислений и их независимости друг от друга. Кроме того, функции Map и Reduce описываются на языке программирования высокого уровня, что, несомненно, предоставляет широкие возможности по обработке данных.

Работа с MapReduce в MongoDB

Для обработки коллекций данных с помощью MapReduce необходимо в консоли mongo выполнить команду «mapReduce» применительно к обрабатываемой коллекции. Синтаксис команды:

db.<коллекция>.mapReduce(<map>, <reduce>,

{

<out> (коллекция),

<query> (Документ),

<sort> (Документ),

<Limit> (Число),

<finalize> (Функция),

<scope> (Документ),

<jsMode> (Логическое),

<verbose> (Логическое)

},

где:

<map> – функция Map написанная на JavaScript;

<reduce> – функция Reduce написана на JavaScript;

<out> – указывает, куда выводить результат выполнения MapReduce;

<query> – определяет критерии отбора с использованием операторов запроса для входных документов Map функции;

<sort> – сортирует входные документы;

<limit> – задает максимальное число документов для возврата из коллекции; <finalize> – JavaScript функция, которая вызывается после исполнения Reduce-шага;

<scope> – определяет глобальные переменные, которые доступны в функциях Map, Reduce и Finalize;

<jsMode> – указывает, требуется ли преобразовывать промежуточные данные в BSON формат;

<verbose> – указывает, следует ли включать информацию о времени выполнения в результирующую информацию.

Первые три параметра функции являются обязательными, остальные – опциональные. При передаче в качестве параметра out имени коллекции все данные из неё будут удалены. Для того чтобы добавить данные в существующую коллекцию необходимо использовать опцию «merge»: {out: {merge: <коллекция>}}

Порядок выполнения лабораторной работы

1. Создайте коллекцию для хранения исходных документов для MapReduce.

```
> db.selling.find()
```

2. Наполните коллекцию данными.

```
db.selling.insert({name: "Nexus One", date: new Date(2013, 0, 20, 12, 00)})
db.selling.insert({name: "Nexus One", date: new Date(2013, 0, 20, 13, 00)})
db.selling.insert({name: "iPhone 4", date: new Date(2013, 0, 20, 13, 30)})
db.selling.insert({name: "Nexus One", date: new Date(2013, 0, 20, 14, 00)})
db.selling.insert({name: "iPhone 4", date: new Date(2013, 0, 21, 15, 30)})
db.selling.insert({name: "iPhone 4", date: new Date(2013, 0, 21, 15, 40)})
db.selling.insert({name: "Nexus One", date: new Date(2013, 0, 21, 16, 20)})
db.selling.insert({name: "iPhone 4", date: new Date(2013, 0, 21, 17, 00)})
db.selling.insert({name: "Nexus One", date: new Date(2013, 0, 21, 18, 00)})
db.selling.insert({name: "Nexus One", date: new Date(2013, 0, 22, 19, 00)})
```

3. Создайте map-функцию в консоли (консоль позволяет вводить многострочные конструкции).

```
var map = function() {
        var key = {
            name: this.name,
            year: this.date.getFullYear(),
            month: this.date.getMonth(),
            day: this.date.getDate()
        };
        emit(key, {count: 1});
```

}

4. Создайте Reduce функцию в консоли.

```
var reduce = function(key, values) {
        var sum = 0;
        values.forEach(function(value) {
                sum += value['count'];
        });
        return {count: sum};
};
```

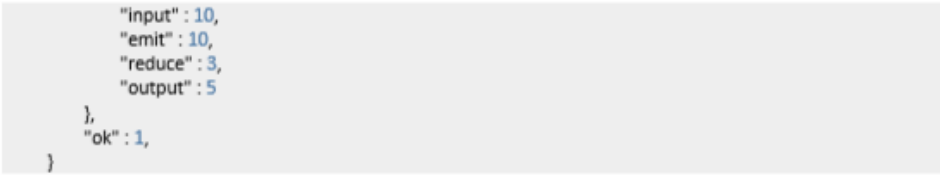
5. Выполните команду mapReduce над коллекцией «selling». Укажите параметр «inline: 1» для вывода результата в консоль.

>db.selling.mapReduce(map, reduce, {out: {inline: 1}})

6. Ознакомьтесь с результатом выполнения mapReduce.

> db.selling.mapReduce(map, reduce, {out: {inline: 1}})

```
"results" : [
    {
        "_id" : {
            "name" : "Nexus One",
            "year" : 2013,
            "month" : 0,
            "day" : 20
        },
        "value" : {
            "count" : 3
        }
    },
    {
        "_id" : {
            "name" : "Nexus One",
            "year" : 2013,
            "month" : 0,
            "day" : 21
        },
        "value" : {
            "count" : 2
        }
    },
    {
        "_id" : {
            "name" : "Nexus One",
            "year" : 2013,
            "month" : 0,
            "day" : 22
        },
        "value" : {
            "count" : 1
        }
    },
    {
```



7. Выполните команду mapReduce над коллекцией «selling». Для вывода результата в коллекцию параметр «out» принимает имя результирующей коллекции.

>db.selling.mapReduce(map, reduce, {out: "sellingResult"})

8. Удостоверьтесь в том, что выполнение функции произошло без ошибок и результат соответствует ожидаемому. Запросите список документов коллекции «sellingResult»:

Сравните список полученных документов с ожидаемым результатом:

```
{ "_id" : { "name" : "Nexus One", "year" : 2013, "month" : 0, "day" : 20 }, "val
ue" : { "count" : 3 } }
{ "_id" : { "name" : "Nexus One", "year" : 2013, "month" : 0, "day" : 21 }, "val
ue" : { "count" : 2 } }
{ "_id" : { "name" : "Nexus One", "year" : 2013, "month" : 0, "day" : 22 }, "val
ue" : { "count" : 1 } }
{ "_id" : { "name" : "iPhone 4", "year" : 2013, "month" : 0, "day" : 20 }, "valu
e" : { "count" : 1 } }
{ "_id" : { "name" : "iPhone 4", "year" : 2013, "month" : 0, "day" : 21 }, "valu
e" : { "count" : 3 } }
```

Рассмотрим следующий пример и создадим новую коллекцию Writer

> db.createCollection("Writer")

Довавим в коллекцию Writer пять документов:

{"Автор": "Пелевин В. О.", "Название": ["Чапаев и Пустота"], "Тип": "Роман"}

{"Автор": "Пелевин В. О.", "Название": ["Бетман Аполло"], "Тип":"Фантастика"}

{"Автор": "Лукьяненко С. В.", "Название": ["Ночной дозор", "Дневной дозор"], "Тип":"Фантастика"}

{"Автор": "Кант И.", "Название": ["Критика чистого разума", "Критика практического разума"], "Тип":"Философия"}

{"Автор": "Пелевин В. О.", "Название": "Generation «П»", "Тип": "Роман"}

Используем MapReduce для вычисления количества произведений каждого типа. Для этого выбираем ключом атрибут

"Тип" и создаем функцию map, которая для каждого входного документа генерирует пару (key, value), где ключ (key) получает значение атрибута «Тип», а value — количество произведений — элементов в массиве "Название". При создании map-функции необходимо учесть, что в последнем документе значение атрибута "Название": "Generation «П» " не является массивом и попытка определить его длину приводит к ошибке. Поэтому в функции предусмотрена проверка типа атрибута: typeof (this.Название) == 'object'. Функция map имеет вид:

```
> var mapFunc1 = function () {
... var key = this.Тип;
... if (typeof (this.Название) == 'object')
...         value= this.Название.length;
... else value = 1;
... emit (key, value);}
```

Задачей функции reduce является суммирование элементов (количество произведений) в массиве значений для каждого значения ключа (тип произведения):

```
> var reduceFunc1 = function (keyType, valuesNumb) {
... return Array.sum (valuesNumb);}
```

Последовательное выполнение обеих функций реализует метод обрабатываемой коллекции db.Writer.mapReduce с сохранением результата в коллекции "TypeNumb":

```
> db.Writer.mapReduce ( mapFunc1, reduceFunc1, {out:
"TypeNumb"})
```

По окончании обработки коллекции Writer метод возвратит статистику решения задачи в формате документа

Запрос db.TypeNumb.find ({}) вернет коллекцию с результатом обработки:

> db.TypeNumb.find()

{ "_id" : "Роман", "value" : 2 }

{ "_id" : "Фантастика", "value" : 3 }

{ "_id" : "Философия", "value" : 2 }

В следующем примере рассмотрим обратную задачу подсчета числа авторов для каждой книги в коллекции Writer. Ключом служит название книги, являющееся элементом массива "Название", а значение — число документов (авторов), содержащих в атрибуте-массиве это название.

Дополним коллекцию писателей Writer двумя книгами, имеющими несколько авторов:

({"Автор": "Илья Ильф", "Книги": [{"Название":"Золотой теленок", "Год":1931}, {"Название":"12 стульев", "Год": 1928}]})

({"Автор": "Евгений Петров", "Книги": [{"Название":"Золотой теленок", "Год":1931}, {"Название":"12 стульев", "Год": 1928}]})

Структура новых документов отличается от введенных ранее. Теперь "Название" является атрибутом документа, вложенного в массив "Книги". Наличие атрибута "Книги" позволит выполнить тестирование новых map- и reduce-функций только для этих двух документов.

Создаем функцию для этапа map, которая проверяет в документе наличие атрибута "Книги" и для такого документа генерирует набор пар вида <Ключ, Значение>, где ключом является название книги, а значением 1.

> var mapFunc2 = function () {if (this.Книги!= null) {//проверка наличия атрибута "Книги"

... for (var i = 0; i < this.Книги.length; i++) { emit (this.Книги [i].Название, 1)} } };

Функция Reduce суммирует значения 1 для каждого ключа (название книги) и записывает в виде документа {<Ключ>, <Сумма единиц>} в выходную коллекцию:

> var reduceFunc2 = function (keyНазв, valuesКол) { return Array.sum (valuesКол);}

Для выполнения функций map и Reduce с сохранением результата в коллекции "Книга_ЧислоАвторов" вызываем метод MapReduce:

> db.Writer.mapReduce (mapFunc2, reduceFunc2, {out: "Книга_ЧислоАвторов"})

На входе map-функции всего было 7 документов. Проверка this.Книги!= null оставила два последних документа. По ним было эмитировано ("emit":4) четыре пары вида (<Название книги>, 1), которые функцией reduce были объединены в две пары и записаны в коллекцию "Книга_ЧислоАвторов".

Запрос к полученной коллекции возвращает результат обработки:

> db.Книга_ЧислоАвторов.find ({}, {})

{ "_id" : "12 стульев", "value" : 2 }

{ "_id" : "Золотой теленок", "value" : 2 }

Задание для самостоятельной работы

1. Создайте коллекцию документов, аналогичную рассмотренным выше, для обработки её с помощью MapReduce.

2. Наполните коллекцию документами.

3. Произведите обработку коллекции с использованием модели распределенных вычислений MapReduce.

Вопросы для защиты работы

1. Что означает термин MapReduce?
2. Из каких шагов состоит работа MapReduce?
3. Какими преимуществами обладает MapReduce по сравнению с обычными вычислениями?
4. Опишите работу с MapReduce в MongoDB.

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения заданий самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты

Лабораторная работа №7

Создание и использование ссылок

Нереляционные СУБД позволяют проектировать модель предметной области в виде набора объектов. При этом информация об одной сущности, разбросанная по различным таблицам РБД, в нереляциионной БД будет собрана в одном объекте. Главным отличие БД mongo от РБД является отсутствие аналога операции соединения (JOIN). Если существует необходимость использовать соединения в базе данных, то они реализуются в программном коде приложения. Для того чтобы найти данные, связанные с каким-либо документом, как правило, необходимо выполнить второй запрос.

Для связывания документов можно сохранять их вместе с «_id» связанных документов.

В качестве примера проиллюстрируем сохранение информации о производителе телефонов в виде связанной записи.

Далее следует выполнять команды, выделенные **жирным шрифтом.**

Создадим новую коллекцию

> db.createCollection("Model")

Добавим в нее документ

> db.Model.insert({_id: ObjectId ("000000000000000000000001"), "Name": "Nokia", "BrandName": "Nokia", "BrandCountry": "Finland"})

На документ с именем «Nokia» будут ссылаться другие документы. Для создания связанного документа необходимо знать

поле «_id» документа «Nokia». Запись с указанием фирмы производителя будет выглядеть следующим образом:

> db.Model.insert({_id: ObjectId ("000000000000000000000002"), "Name": "L920", "Model": "Lumia 920", "OSFamily": "Windows", "OSVersion": "8", "Brand": ObjectId ("000000000000000000000001")})

Обратите внимание, что значение поля «Brand» документа с именем «L920» и поля «_id» документа «Nokia» совпадают. Поле «_id» может быть любым уникальным значением. Чтобы найти все телефоны, произведенные под брендом «Nokia», необходимо выполнить запрос с указанием значения его поля «_id»:

> db.Model.find ({Brand: ObjectId ("000000000000000000000001")})

В результате получим

{ "_id" : ObjectId("000000000000000000000002"), "Name" : "L920", "Model" : "Lumia 920", "OSFamily" : "Windows", "OSVersion" : "8", "Brand" : ObjectId("000000000000000000000001") }

Введем еще один документ, значение поля «Brand» которого также совпадает с полем «_id» документа «Nokia»

>db.Model.insert({_id: ObjectId ("000000000000000000000003"), "Name":"Samsung","Model":"Galaxy S3","OSFamily":"Android","OSVersion":"4.0 ice Cream Sandwich","Brand": ObjectId ("000000000000000000000001")})

Тогда в результате выполненного запроса

> db.Model.find ({Brand: ObjectId ("000000000000000000000001")})

получим оба связанных с документом «Nokia» документа

{ "_id" : ObjectId("000000000000000000000002"), "Name" : "L920", "Model" : "Lumia 920", "OSFamily" : "Windows", "OSVersion" : "8", "Brand" : ObjectId("000000000000000000000001") }

{ "_id" : ObjectId("000000000000000000000003"), "Name" : "Samsung", "Model" : "Galaxy S3", "OSFamily" : "Android", "OSVersion" : "4.0 ice Cream Sandwich", "Brand" : ObjectId("000000000000000000000001") }

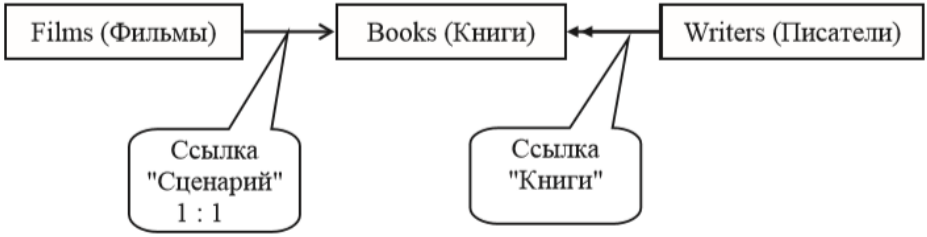
Возможность создавать ненормализованные документы сложной структуры, содержащие вложенные документы и/или их массивы, позволяет адекватно представлять в базе отдельные объекты, имеющие сложную структуру. Для представления взаимодействий между объектами используются ссылки. Ссылка — атрибут в документе, значение которого используется для доступа к другому документу. Ссылки являются средством прямого представления связей и используются для навигационной обработки документов.

Рассмотрим «ручные» ссылки. Для однозначной идентификации документа в MongoDB ручная ссылка содержит три атрибута: $db:<Имя базы>, $ref:<Имя коллекции>, $id:<Ключ документа>.

В атрибуте $db указывается имя базы, которая содержит ссылочный документ. При обращении к документу ссылка $db необязательна. При ее отсутствии поиск документа ведется в установленной базе. Атрибуты $ref и $id определяют коллекцию и идентификатор ссылочного документа в коллекции. Для

обращения к документу по хранящейся в БД ссылке необходимо сначала найти исходный документ и извлечь атрибуты ссылки, а затем использовать их в новом запросе к ссылочному документу.

Пусть создана БД, которая содержит информацию о кинофильмах в коллекции Films (Фильмы), данные о писателях (сценаристах) в коллекции Writers (Писатели), а сведения об их произведениях размещены в документах коллекции Books (Книги). Связи данных показаны на рис. . В документе о фильме предусмотрен атрибут-ссылка "Сценарий", указывающий на книгу, по которой поставлен фильм. В документе, представляющем автора, предусмотрен массив ссылок на книги, написанные этим автором.



Рассмотрим способ создания и использования ручных ссылок на примере ссылки "Сценарий". Пусть коллекция Films содержит информацию о фильме "Ночной дозор":

> db.createCollection("Films")

> db.Films.insert({"_id": ObjectId ("55c8214cb7f9469326e345cb"), "Название": "Ночной дозор", "Режиссер": "Бекмамбетов Т. Н.", "Год": 2004})

В коллекцию Books помещаем сведения о романе "Ночной дозор", на основе которого был снят фильм

> db.createCollection("Books")

> db.Books.insert({"_id": ObjectId ("55c82635b7f9469326e345cc"), "Название": "Ночной дозор", "Год": 1998, "Тип": "Фантастика"})

В соответствии со схемой данных (рис.) для представления сведений о книге в записи о фильме следует создать ссылку на роман "Ночной дозор". Добавим в фильм "Ночной дозор" атрибут "Сценарий", содержащий ссылку на роман "Ночной дозор" в коллекции Books.

> db.Films.update ({"Название":"Ночной дозор"}, {$set:{"Сценарий":{"$ref": "Books", "$id": ObjectId ("55c82635b7f9469326e345cc")}}});

Проверка добавленной ссылки вернет измененную запись о фильме:

> db.Films.find ()

{ "_id" : ObjectId("55c8214cb7f9469326e345cb"), "Название" : "Ночной дозор", "Режиссер" : "Бекмамбетов Т. Н.", "Год" : 2004, "Сценарий" : DBRef("Books", ObjectId("55c82635b7f9469326e345cc")) }

Обратите внимание: значение ссылки в атрибуте "Сценарий" теперь представлено функцией DBRef (…).

Для проверки ссылки найдем роман, по которому поставлен фильм "Ночной дозор". Поиск выполняется в два этапа.

1. Сначала надо получить ссылку. Для этого извлекаем из документа "Ночной дозор" атрибут "Сценарий", содержащий ссылку на книгу и сохраняем в переменной "Ссылка":

> var Ссылка = db.Films.findOne ({"Название": "Ночной дозор"}, {"Сценарий": 1, _id: 0});

Получаем в переменной "Ссылка": {"Сценарий":

D B R e f (" B o o k s " , O b j e c t I d ("55c82635b7f9469326e345cc"))}

2. Затем выбираем книгу по хранящейся в переменной ссылке:

> db [Ссылка.Сценарий.$ref].findOne ({"_id": (Ссылка.Сценарий.$id)});

{

Перед выполнением команда findOne модифицируется подстановкой из атрибута-ссылки $ref имени коллекции, а из атрибута $id — ключа документа таким образом, что выполняется метод: db.Books.findOne ({"_id": ObjectId

("55c82635b7f9469326e345cc")}), который возвращает сведения о книге:

"_id" : ObjectId("55c82635b7f9469326e345cc"),

"Название" : "Ночной дозор",

"Год" : 1998,

"Тип" : "Фантастика"

}

Задание для самостоятельной работы

1, Создать базу данных, включающую документы, содержащие анкетные данные студентов Вашей группы (фамилия и

инициалы, пол, возраст, телефон, семейное положение) и документы, содержащие информацию о результатах сдачи последней экзаменационной сессии каждым студентом группы.

2.Связать документы, содержащие информацию об одном и том же студенте

Вопросы для защиты работы

1. Для чего нужны ссылки?
2. Какие атрибуты содержит ручная ссылка ?
3. Что определяет атрибут $db?
4. Что определяет атрибут $ref ?
5. Что определяет атрибут и $id?

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения заданий самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты

Лабораторная работа №8

Алгоритм k-внутригрупповых средних

Рассмотрим один из популярных алгоритмов кластеризации, основанный на минимизации функционала суммарной выборочной дисперсии разброса элементов относительно центров тяжести кластеров. Этот алгоритм представляет собой пошаговое (итерационное) нахождение центров тяжести кластеров и разбиение обучающей выборки на кластеры до тех пор, пока некоторый функционал Q не перестанет уменьшаться.

Алгоритм k-внутригрупповых средних

1. Выделяются некоторые образы из обучающей выборки – начальные центры кластеров $\mathbf{c}_1^{(0)},...,\mathbf{c}_m^{(0)}$ и полагается $k=0$.
2. Вся обучающая выборка разбивается на m кластеров (клеток Вороного) по *методу ближайшего соседа* – получаются некоторые кластеры $X_1^{(k)},...,X_m^{(k)}$.
3. Рассчитываются новые центры – центры тяжести кластеров по формуле $\mathbf{c}_i^{(k+1)}=\frac{1}{\left|X_i^{(k)}\right|}\sum_{\mathbf{x}\in X_i^{(k)}}\mathbf{x}$.
4. Проверяется выполнение условия останова: $\mathbf{c}_i^{(k+1)}=\mathbf{c}_i^{(k)}$ для всех $k=1,...,m$. В противном случае – переход к пункту 2.

Пример. Предположим, что на плоскости R^2 заданы векторы-образы $\mathbf{x}_1=(1,1)$, $\mathbf{x}_2=(0,0)$, $\mathbf{x}_3=(2,0)$, $\mathbf{x}_4=(4,4)$, $\mathbf{x}_5=(5,5)$, $\mathbf{x}_6=(5,3)$ (рис. 4.1). Найдем кластеризацию этих образов по двум классам. Для этого выполним последовательно шаги рассмотренного алгоритма.

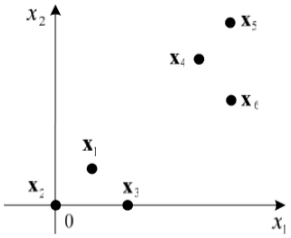


Рис. 4.1

1. В качестве начальных центров кластеров выберем образы $\mathbf{c}_1^{(0)}=\mathbf{x}_1$ и $\mathbf{c}_2^{(0)}=\mathbf{x}_2$. Тогда, разбивая выборку $\{x_1,...,x_6\}$ на два подмножества по методу ближайшего соседа, получим начальные кластеры $X_1^{(0)}=\{\mathbf{x}_1,\mathbf{x}_3,\mathbf{x}_4,\mathbf{x}_5,\mathbf{x}_6\}$ и $X_2^{(0)}=\{\mathbf{x}_2\}$.

2. Вычисляем новые центры – центры тяжести кластеров

$$\mathbf{c}_1^{(1)}=\frac{1}{5}\begin{pmatrix}x_{11}+x_{31}+x_{41}+x_{51}+x_{61}\\x_{12}+x_{32}+x_{42}+x_{52}+x_{62}\end{pmatrix}=\frac{1}{5}\begin{pmatrix}1+2+4+5+5\\1+0+4+5+3\end{pmatrix}=\begin{pmatrix}17/5\\13/5\end{pmatrix},\ \mathbf{c}_2^{(1)}=\mathbf{x}_2=\begin{pmatrix}0\\0\end{pmatrix}.$$

3. Сравниваем: $\mathbf{c}_1^{(0)}\neq\mathbf{c}_1^{(1)}$ и $\mathbf{c}_2^{(0)}=\mathbf{c}_2^{(1)}$. Продолжаем выполнение алгоритма.

4. Разбиваем выборку $\{\mathbf{x}_1,...,\mathbf{x}_6\}$ на два подмножества с новыми центрами по методу ближайшего соседа, получим кластеры $X_1^{(1)}=\{\mathbf{x}_4,\mathbf{x}_5,\mathbf{x}_6\}$ и $X_2^{(1)}=\{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3\}$.

5. Вновь вычисляем центры тяжести кластеров

$$\mathbf{c}_1^{(2)}=\frac{1}{3}\begin{pmatrix}x_{41}+x_{51}+x_{61}\\x_{42}+x_{52}+x_{62}\end{pmatrix}=\frac{1}{3}\begin{pmatrix}4+5+5\\4+5+3\end{pmatrix}=\begin{pmatrix}14/3\\4\end{pmatrix},$$

$$\mathbf{c}_2^{(2)}=\frac{1}{3}\begin{pmatrix}x_{11}+x_{21}+x_{31}\\x_{12}+x_{22}+x_{32}\end{pmatrix}=\frac{1}{3}\begin{pmatrix}1+0+2\\1+0+0\end{pmatrix}=\begin{pmatrix}1\\1/3\end{pmatrix}.$$

6. Сравниваем: $\mathbf{c}_1^{(1)}\neq\mathbf{c}_1^{(2)}$ и $\mathbf{c}_2^{(1)}\neq\mathbf{c}_2^{(2)}$. Продолжаем выполнение алгоритма.

Используя возможности программирования в системе MATLAB, составим программу, реализующую указанный алгоритм.

Прежде всего, добавим каждому из векторов $x_1=(1,1)$, $x_2=(0,0)$, $x_3=(2,0)$, $x_4=(4,4)$, $x_5=(5,5)$, $x_6=(5,3)$ для удобства третью координату, указывающую номер кластера, к которому относится данный вектор.

При реализации указанного алгоритма часто будет необходимо вычислять расстояние между векторами. Для этого составим функцию, вычисляющую расстояние между двумя векторами, учитывая при этом, что третьи координаты не должны принимать участие в этих расчетах (они добавлены нами и в исходном состоянии у векторов отсутствовали).

```
function [R] = DS(X,Y)
 [M,N]=size(X);
 R=0;
for i=1:N-1
  R=R+ (X(i)-Y(i))^2;
 end
 R=sqrt(R);
 end
```

Функция size(X) определяет число строк и число столбцов матрицы X (в данном случае X вектор).

```
%Кластеризация метлдом k-внутригрупповых средних
n=6;
X1=[1,1,0];X2=[0,0,0];X3=[2,0,0];
X4=[4,4,0];X5=[5,5,0];X6=[5,3,0];
XX=[X1;X2;X3;X4;X5;X6];
C11=zeros(1,3);C11(1,3)=1;
C21=zeros(1,3);C21(1,3)=2;
C10=XX(1,:); C20=XX(2,:);
```

Здесь n – число векторов, C10 и C20 – вектора начальных центров первого и второго кластеров, XX – матрица, строки которой совпадают с векторами x_1, x_2, x_3, x_4, x_5, x_6, C11 и C21- вектора текущих, вычисленных в процессе реализации алгоритма центров первого и второго кластеров (вначале их компоненты полагают равными нулю). После этого можно приступить непосредственно к реализации алгоритма.

1. Выполняем начальную итерацию. Сравнивая расстояние каждого из векторов до каждого из центров
   [d1]=DS(XI,C10);
   [d2]=DS(XI,C20);
   , определяем к какому кластеру принадлежат вектора (выбирается тот кластер, расстояние до центра которого от рассматриваемого вектора d1,d2 меньше), определяем число векторов, относящихся к каждому кластеру

```
 9 —    n1=0;n2=0;
10 —  for i=1:n
11 —        XI=XX(i,:);
12 —        [d1]=DS(XI,C10);
13 —        [d2]=DS(XI,C20);
14 —        if d1<d2
15 —            XX(i,3)=1;
16 —            n1=n1+1;
17 —        else
18 —           XX(i,3)=2;
19 —           n2=n2+1;
20 —        end
21 —  end
```

Здесь же определяется число векторов, принадлежащих каждому кластеру n1 и n2

В третий столбец матрицы XX вносится номер кластера, которому принадлежит соответствующий вектор

2.После этого определяем новые центры кластеров

```
22 -        for j=1:2
23 -            for i=1:n
24 -                if XX(i,3)==1
25 -                   C11(j)=C11(j)+XX(i,j)/n1;
26 -                end
27 -                if XX(i,3)==2
28 -                   C21(j)=C21(j)+XX(i,j)/n2;
29 -                end
30 -            end
31 -        end
32 -        [dC1]=DS(C10,C11);
33 -        [dC2]=DS(C20,C21);
```

Здесь же определяем расстояния между начальными центрами кластеров

Последующие итерации можно выполнить, использую оператор цикла while

Опять, сравнивая расстояние каждого из векторов до каждого из найденных новых центров

[d1]=DS(XI,C11);
[d2]=DS(XI,C21);

, определяем к какому кластеру принадлежат вектора (выбирается тот кластер, расстояние до центра которого от рассматриваемого вектора d1,d2 меньше) , определяем число векторов, относящихся к каждому кластеру

```
34 -      while dC1+dC2>0
35 -      n1=0;n2=0;
36 -  for i=1:n
37 -      XI=XX(i,:);
38 -      [d1]=DS(XI,C11);
39 -      [d2]=DS(XI,C21);
40 -      if d1<d2
41 -          XX(i,3)=1;
42 -          n1=n1+1;
43 -      else
44 -         XX(i,3)=2;
45 -         n2=n2+1;
46 -      end
47 -  end
48 -   C10=C11; C20=C21;
49 -   C11=zeros(1,3);C21=zeros(1,3);
```

После этого определяем новые центры кластеров

```
50 -    for j=1:2
51 -          for i=1:n
52 -              if XX(i,3)==1
53 -                 C11(j)=C11(j)+XX(i,j)/n1;
54 -              end
55 -              if XX(i,3)==2
56 -                 C21(j)=C21(j)+XX(i,j)/n2;
57 -              end
58 -          end
59 -    end
60 -      [dC1]=DS(C10,C11);
61 -      [dC2]=DS(C20,C21);
62 -      end
```

Определяем расстояния между старыми центрами и новыми, только что определенными центрами

[dC1]=DS(C10,C11);
[dC2]=DS(C20,C21);

Если эти центры совпадают, то dC1=dC2=0 и dC1+dC2=0 и итерационный процесс прекращается. Если новые центры не совпадают с новыми, то или dC1>0 или dC2>0 или обе эти величины больше нуля. Тогда dC1+dC2>0 и итерационный процесс продолжается (while dC1+dC2>0)
Далее выводим результаты вычислений на экран

```
63 -        disp('Кластер 1 образуют вектора с номерами')
64 -        for i=1:n
65 -        if XX(i,3)==1
66 -            disp(i);
67 -        end
68 -        end
69 -        disp('Кластер 2 образуют вектора с номерами')
70 -        for i=1:n
71 -        if XX(i,3)==2
72 -            disp(i);
73 -        end
74 -        end
```

Здесь команда disp выводит на экран значение переменной (disp(i)) или текст

disp('Кластер 2 образуют вектора с номерами')

Задание для самостоятельной работы

1.Проверить работу программы для других исходных данных векторов x_1=(1,1), x_2=(0,0), x_3=(2,0), x_4=(-4,3), x_5=(-4,5), x_6=(-3,5)

2.Разработать программу, на любом алгоритмическом языке, отличном от языка используемого выше, осуществляющую кластеризацию образов по трем классам.

Проверить работу программы на множестве образов (векторов) $x_1=(1,1)$, $x_2=(0,0)$, $x_3=(2,0)$, $x_4=(4,4)$, $x_5=(5,5)$, $x_6=(5,3)$, $x_7=(-4,3)$, $x_8=(-4,5)$, $x_9=(-3,5)$.

3.Составить отчет по лабораторной работе. В отчете необходимо описать разработанную Вами программу так, как это сделано выше, и представить результаты работы разработанных программ.

Контрольные вопросы

1. По какой формуле вычисляется расстояние между двумя векторами
   в рассмотренном выше методе?
2. Какие еще существуют формулы для вычисления расстояний между двумя векторами?
3. Какими свойствами должна обладать функция ρ(x,y), определяющая расстояние между векторами x и y ?
4. По какой формуле вычисляются центры тяжести кластеров?
5. На каком шаге останавливается итерационный процесс рассмотренного выше алгоритма?

Лабораторная работа № 9

Метод опорных векторов

Цель работы изучить функционирование метода опорных векторов в задачах классификации

Краткие теоретические сведения

Рассмотрим задачу классификации на два непересекающихся класса, в которой объекты описываются n-мерными вещественными векторами, выход D={-1;+1}.

Значение 1 соответствует принадлежности одному классу, а -1 — другому. Будем строить линейный пороговый классификатор:

$$y(X) = \text{sign}(w*X+w_0),$$

где $X=(x_1,x_2,\ldots x_n)$ — признаковое описание объекта X ; вектор $w=(w_1,w_2,\ldots w_n)$ и скалярный порог w_0 являются параметрами алгоритма; sign — функция знака. Напомним, что уравнение $w*X+w_0=0$ описывает гиперплоскость, разделяющую классы в n-мерном пространстве.

Критерий и методы настройки параметров в SVM радикально отличаются от персептронных (градиентных) методов обучения.

Метод опорных векторов использует три основные идеи:

- оптимальная разделяющая гиперплоскость;
- возможно неточное разделение, но за ошибки в разделении платится штраф (математический, конечно);
- нелинейное отображение данных.

Рассмотрим каждую из этих идей подробнее.

Оптимальная разделяющая гиперплоскость

Предположим, что выборка линейно разделима, то есть существуют такие значения параметров w, w_0 , при которых функционал числа ошибок принимает нулевое значение. Но тогда разделяющая гиперплоскость не единственна, поскольку существуют и другие положения разделяющей гиперплоскости, реализующие то же самое разбиение выборки. Идея метода заключается в том, чтобы разумным образом распорядиться этой свободой выбора. Потребуем, чтобы разделяющая

гиперплоскость максимально далеко отстояла от ближайших к ней точек обоих классов.

Это приводит к задаче квадратичного программирования, поскольку среднее удаление разделяющей плоскости от разделяемых точек пропорционально квадрату длины вектора неизвестных весов W. Задача квадратичного программирования, вообще говоря, трудная (значительно более сложная, чем линейное программирование), но во многих практических случаях успешно решается.

Штраф за ошибки в разделении

Вторая важная идея — можно применять метод даже тогда, когда множества линейно неразделимы. В этом случае предлагается ввести понятие штрафа и за ошибки в разделении платить этот штраф. Это классическая идея в оптимизации, идущая еще от Лагранжа.

Пусть X^i — i-й пример. Тогда система уравнений метода SVM принимает вид:

$$\|w\| + C\sum_i T\xi_i \to min$$
$$y_i\left(w \cdot X^i + w_0\right) \geq 1 - \xi_i$$

В этой системе неизвестными являются ξ_i и коэффициенты w.

Неотрицательные переменные ξ_i описывают штрафные санкции за то, что пример X^i неправильно классифицирован; здесь wX^i — скалярное произведение. Если $\xi_i = 0$, то мы получаем обычную задачу разделения для персептрона, где ищем оптимальную гиперплоскость:

$$\|w\| \to min$$
$$y_i\left(w \cdot X^i + w_0\right) \geq 1$$

Последнее условие означает, что выход персептрона

$$\left(w \cdot X^i + w_0\right)$$

и реальный выход y_i имеют одинаковый знак.

Это сложная задача квадратичного программирования. При наличии штрафов имеется еще параметр C, который надо подбирать. Для этой, вообще говоря, непростой задачи, разработаны методы ее решения.

Нелинейное отображение данных. Ядра (kernels)

Нелинейное отображение в другое пространство с другим скалярным произведением может превращать линейно неразделимые множества в линейно разделимые: $x \to \psi(x)$.

Вообще говоря, если размерность ψ выше, чем размерность x, то мы можем получить линейное разделение образов гиперплоскостью в пространстве ψ .

Как это происходит?

Пусть имеются два множества A и B. Они могут быть неразделимы гиперплоскостью. Рассмотрим их образы ψ (A) и ψ (B) в результате действия некоторого нелинейного отображения $x \to \psi(x)$.

В качестве примера рассмотрим разделение внутренности A и внешности B эллипса, определенного уравнением.

$$2x^2 + 3y^2 - xy = 1$$

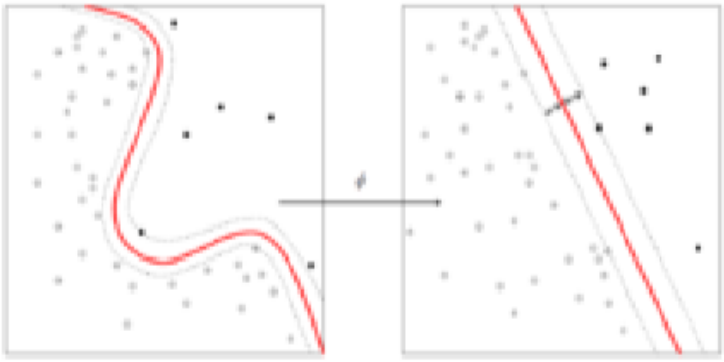
Множества A и B линейно неразделимы в исходном множестве (точек плоскости). Однако, они разделимы в пространстве ψ , к которому можно перейти с помощью отображения

$$\psi = (\psi_1, \psi_2, \psi_3) = (x^2, y^2, xy)$$

Уравнение плоскости в пространстве ψ для классификатора принимает вид

$$2\psi_1 + 3\psi_2 - \psi_3 = 1.$$

Общий подход к применению нелинейных отображений основан на теореме Ковера о разделимости. Неформально говоря, эта теорема утверждает, что нелинейное преобразование данных сложной задачи классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов (графическая иллюстрация теоремы представлена на рис..



Ядра

Выяснилось, что после нелинейного преобразования ψ во всех основных формулах классификации используется выражение ψ(x)* ψ(y)

.

Именно, с помощью функции ψ строится так называемое ядро вида

$$K(u,v)=\psi(u)\cdot\psi(v)$$

Данная форма является канонической, тем не менее, также используют и другие ядра. Например:

- $K(u,v)=(u\cdot v+1)^p$
- гауссовская радиальная функция (Gaussian Radial Basis Function)

$$K(u,v)=exp\left(-\frac{(u-v)^2}{2\sigma^2}\right)$$

- гиперболическая разделяющая поверхность (Sigmoidal)

$$K(u,v)=tanh(k\cdot u\cdot v-\delta)$$

К сожалению, четких методов поиска ядер K нет.

Преимущества SVM. Принцип оптимальной разделяющей гиперплоскости приводит к максимизации ширины разделяющей полосы между классами, и, следовательно, к более уверенной классификации. Градиентные нейросетевые методы выбирают положение разделяющей гиперплоскости произвольным образом, как придется.

Недостатки. Метод опорных векторов неустойчив по отношению к шуму в исходных данных. Если обучающая выборка содержат шумовые выбросы, они будут существенным образом учтены при построении разделяющей гиперплоскости.

Построение нейронной сети для разделения внешности и внутренности эллипса в пакете Matlab

В качестве практического примера рассмотрим приведенную выше задачу о разделении внутренности и внешности произвольного эллипса.

Ниже приведен скрипт, решающий эту задачу с помощью метода опорных векторов средствами пакета Matlab; и исполь-зуемая в нем пользовательская функция ellipse_Train_SVM_test. Оба приведенных скрипта (основной и m-функция) снабжены подробными комментариями. Основной скрипт заканчивается проверкой нейронной сети, то есть проведением классификации произвольно взятой точки плоскости с помощью построенной сети. Также для большей наглядности приведены соответствующие построения на плоскости — исходная кривая, точки двух множеств (помеченные различным образом), построенная сетью разделяющая кривая, тестовая точка.

Эллипс задается уравнением

$$a \cdot x^2 + b \cdot xy + c \cdot y^2 + d \cdot x + e \cdot y = 1.$$

```
% разделим точки плоскости на два класса:
% внутри и вне произвольного эллипса ax^2 + bxy + cy^2 + dx + ey = 1

% N - число точек для обучения
N = 300;

% ввод коэффициентов эллипса с помощью диалогового окна
prompt = {'a','b','c','d','e'};
dlg_title = 'Input'; num_lines = 1;

% зададим эллипс
def={'1.5', '2', '1.1', '-2','-3'}; % значения коэффициентов по умолчанию
% V - параметры эллипса (массив ячеек)
% в каждой ячейке содержится соответствующий коэффициент
% окно для ввода параметров эллипса
V = inputdlg(prompt, dlg_title, num_lines, def);

% преобразование к типу double
A = str2double(V);
% запись каждого коэффициента в соответствующую переменную
a = A(1); b = A(2); c = A(3); d=A(4); e = A(5);
% построим график исходного эллипса
hold on, syms X Y; f = a*X.^2 + b*X.*Y + c*Y.^2 + d*X + e*Y-1;
ellips = ezplot(f); set (ellips,'Color','b');
```

```
% определим ядро SVM-сети
kernel = 'quadratic';
% обращение к пользовательской функции построения SVM-сети
net_SVM = ellipse_Train_SVM_test(N, A, kernel)

% зададим координаты произвольной точки плоскости
new_dot = [randn(1)*(a) - d/(2*a), randn(1)*(2*c) - e/(c)]
% проведем классификацию точки с помощью построенной сети
output = svmclassify(net_SVM,new_dot,'Showplot',true)

function [SVMStruct] = ellipse_Train_SVM_test(N, A, kernel);
% метод опорных векторов
% строит SVM для классификации точек плоскости
% в соответствии с заданным эллипсом
% входные данные:
```

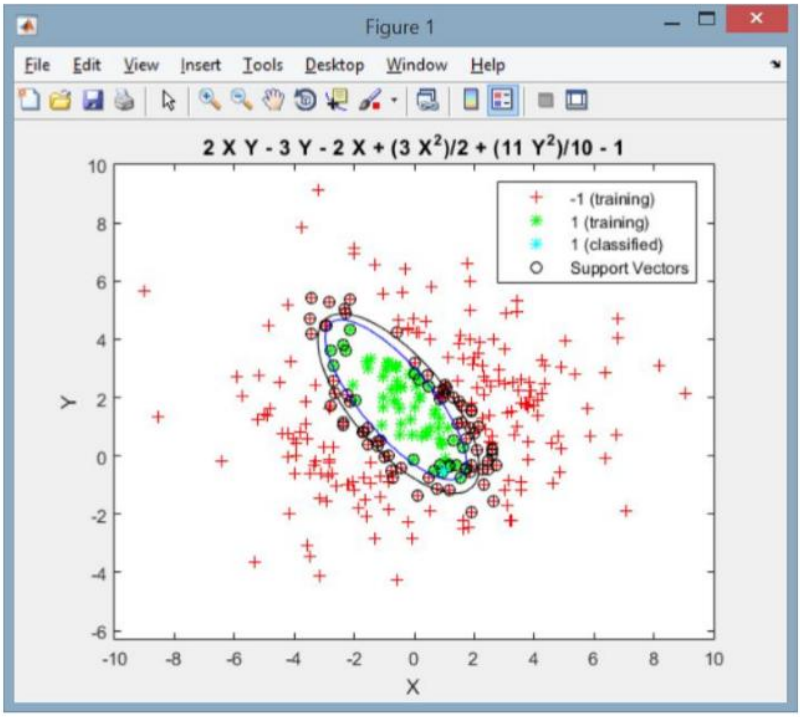
```
% A - массив коэффициентов произвольного эллипса
% N - число точек для обучения
```

В окне команд будет выведена следующая информация: координаты произвольно взятой точки new_dot и результат классификации: 1 или 0, где 1 означает, что точка new_dot лежит внутри эллипса, 0 — вне эллипса. Кроме того, будет построена графическая иллюстрация процесса (рис. 18), а именно: тренировочное множество точек отображается красными «крестиками» (точки вне эллипса) и зелеными «звездочками» (точки внутри эллипса); голубой «звездочкой» отмечена произвольно взятая для классификации точка; черный эллипс — построенная с помощью SVM разделяющая кривая; синий эллипс — заданный эллипс.

```
 new_dot =

1.0        -0.5000
output =

1
```



Для обучения SVM-классификатора нами была использована встроенная функция svmtrain, имеющая в общем случае следующий синтаксис: svmtrain(Training, Group, Name, Value)

где Training — тренировочное множество, для каждой строки которого определено значение классификатора (элемент массива Group). При необходимости могут быть заданы дополнительные параметры, которые записываются парами — имя параметра, значение параметра.

Так, параметр 'kernel_function' позволяет определить функцию ядра SVM-сети и может принимать одно из следующих значений

 'linear' — линейное ядро (по умолчанию);

 'quadratic' — квадратичное ядро;

 'polynomial' — полиномиальное ядро (по умолчанию 3-го порядка), данное значение параметра требует указания порядка полинома посредством определения параметра polyorder (то есть пары 'polyorder', значение);

 'rbf' — гауссовская радиальная функция (Gaussian Radial Basis Function) с параметром σ=1; при необходимости указания другого значения σ, требуется определить параметр 'rbf_sigma' (то есть пару 'rbf_sigma', значение);

 'mlp' — многослойное ядро персептрона со шкалой [-1; 1] (по умолчанию); при необходимости указания другого значения необходимо определить параметр 'mlp_params' (то есть пару 'mlp_params', значение).

Порядок выполнения работы

1. Воспроизведите приведенный в качестве примера скрипт. Измените ядро SVM-сети ('linear', 'polynomial', 'rbf'), проведите классификацию в каждом случае, проанализируйте полученные результаты.

2. Проведите классификацию точек плоскости для произвольно заданной гиперболы: точки плоскости, лежащие между двумя ветвями гиперболы, и точки плоскости, лежащие «снаружи» ветвей.

3.Проанализируйте полученные результаты

Содержание отчета

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы

1. Возможности метода опорных векторов
2. Какие основные идеи положены в основу метода опорных векторов?
3. Оптимальная разделяющая гиперплоскость
4. Преимущества и недостатки метода опорных векторов
5. Штраф за ошибки в разделении (суть идеи)

ПЕРЕЧЕНЬ ИСПОЛЬЗОВАННЫХ ИНФОРМАЦИОННЫХ РЕСУРСОВ

1. Ю. П. Парфенов, Постреляционные хранилища данных : учеб. пособие. Екатеринбург : Изд-во Урал. ун-та, 2016. — 120 с.
2. А.Е Лепский , А.Г. Броневич . Математические методы распознавания образов: Курс лекций. – Таганрог: Изд-во ТТИ ЮФУ, 2009. – 155 с.
3. С.А Вакуленко, А.А. Жихарева Практический курс по нейронным сетям – СПб: Университет ИТМО, 2018. – 71 с.