Нереляционные базы данных

Лабораторная работа 1

Установка и настройка MongoDB. Знакомство с нереляционными базами данных.

Установка MongoDB

Запустите эмулятор терминала.

Для установки MongoDB в окне терминала наберите команду sudo apt-get install mongodb

После установки станут доступны два исполняемых файла: сервер mongod и интерпретатор mongo.

Настройка MongoDB

Создайте путь /data/db для хранения данных.

Для этого введите команду sudo mkdir -p /data/db

Запустите сервер mongod, если все было установлено и настроено правильно, появится сообщение вида [initandlisten] waiting for connections on port 27027

Запустите интерпретатор mongo в новом окне терминала, убедитесь, что появилось сообщение Welcome to the MongoDB shell. и приглашение для пользовательского ввода >.

В окне с сервером должно появиться сообщение вида [initandlisten] connection accepted from 127.0.0.1:55278 #1 (1 connection now open)

Знакомство с MongoDB

Начнём с изучения основных механизмов работы с MongoDB. Это самое основное, что понадобится для понимания MongoDB, но также мы коснёмся высокоуровневых вопросов — о том, где применима MongoDB.

Для начала нужно понять шесть основных концепций.

1. MongoDB — концептуально то же самое, что обычная, привычная нам база данных (или в терминологии Oracle — схема). Внутри MongoDB может быть ноль или более баз данных, каждая из которых является контейнером для прочих сущностей.

2. База данных может иметь ноль или более «коллекций». Коллекция настолько похожа на традиционную «таблицу», что можно смело считать их одним и тем же.
3. Коллекции состоят из нуля или более «документов». Опять же, документ можно рассматривать как «строку».
4. Документ состоит из одного или более «полей», которые — как можно догадаться — подобны «колонкам».
5. «Индексы» в MongoDB почти идентичны таковым в реляционных базах данных.
6. «Курсоры» отличаются от предыдущих пяти концепций, но они очень важны (хотя порой их обходят вниманием) и заслуживают отдельного обсуждения. Важно понимать, что когда мы запрашиваем у MongoDB какие-либо данные, то она возвращает курсор, с которыми мы можем делать все что угодно — подсчитывать, пропускать определённое число предшествующих записей — при этом не загружая сами данные.

Подводя итог, 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 отдаёт эти данные по мере надобности.

Вы можете спросить — зачем придумывать новые термины (коллекция вместо таблицы, документ вместо записи и поле вместо колонки)? Не излишнее ли это усложнение? Ответ в том, что эти термины, хоть и близки своим «реляционным» аналогам, но не полностью идентичны им. Основное различие в том, что реляционные базы данных определяют «колонки» на уровне «таблицы», в то время как документ-ориентированные базы данных определяют «поля» на уровне «документа». Это значит, что любой документ внутри коллекции может иметь свой собственный уникальный набор полей. В этом смысле коллекция «глупее» чем таблица, тогда как документ имеет намного больше информации, чем строка.

Хоть это и важно понять, не волнуйтесь, если не сможете сразу. После нескольких вставок вы увидите, что имеется в виду. В конечном счёте дело в том, что коллекция не содержит информации о структуре содержащихся в ней данных. Информацию о полях содержит каждый отдельный документ. Преимущества и недостатки этого станут понятны из следующей главы.

Приступим. Запустите сервер mongod и консоль mongo, если ещё не запустили. Консоль работает на JavaScript. Есть несколько глобальных команд, например help или exit. Команды, которые вы запускаете применительно к текущей базе данных исполняются у объекта db, например db.help() или db.stats() . Команды, которые вы запускаете применительно к конкретной коллекции, исполняются у объекта db.ИМЯ_КОЛЛЕКЦИИ, например db.unicorns.help() или db.unicorns.count() .

Введите db.help() и получите список команд, которые можно выполнить у объекта db.

Заметка на полях. Поскольку консоль интерпретирует JavaScript, если вы попытаетесь выполнить метод без скобок, то в ответ получите тело метода, но он не выполнится. Не удивляйтесь, увидев function (...){, если случайно сделаете так. Например, если введёте db.help (без скобок), вы увидите внутреннее представление метода help.

Сперва для выбора базы данных воспользуемся глобальным методом use — введите use learn. Неважно, что база данных пока ещё не существует. В момент создания первой коллекции создастся база данных learn. Теперь, когда вы внутри базы данных, можно вызывать у неё команды, например db.getCollectionNames(). В ответ увидите пустой массив ([ ]). Поскольку коллекции бесструктурны *(в оригинале «schema-less». Здесь и далее — прим. перев.)*, мы не обязаны создавать их явно. Мы просто можем вставить документ в новую коллекцию. Чтобы это сделать, используйте команду insert, передав ей вставляемый документ:

```
db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})
```

Данная строка выполняет метод insert («вставить») в коллекцию unicorns, передавая ему единственный аргумент. MongoDB у себя внутри использует бинарный сериализированный JSON формат. Снаружи это означает, что мы широко используем JSON, как, например, в случае с нашими параметрами. Если теперь выполнить db.getCollectionNames(), мы увидим две коллекции: unicorns и system.indexes. system.indexes создаётся в каждой базе данных и содержит в себе информацию об индексах этой базы.

Теперь у коллекции unicorns можно вызвать метод find, который вернёт список документов:

```
db.unicorns.find()
```

Заметьте, что кроме данных, которые мы задавали, появилось дополнительное поле _id . Каждый документ должен иметь уникальное поле _id. Можете генерировать его сами или позволить MongoDB самой сгенерировать для вас ObjectId. В большинстве случаев вы скорее всего возложите эту задачу на MongoDB. По умолчанию _id — индексируемое поле, вследствие чего и создается коллекция system.indexes. Давайте взглянем на system.indexes:

```
db.system.indexes.find()
```

Вы увидите имя индекса, базы данных и коллекции, для которой индекс был создан, а также полей, которые включены в него.

Вернёмся к обсуждению бесструктурных коллекций. Давайте вставим кардинально отличный от предыдущего документ в unicorns, вот такой:

```
db.unicorns.insert({name: 'Leto', gender: 'm', home: 'Arrakeen', worm: false})
```

И снова воспользуемся find для просмотра списка документов. Теперь, узнав чуть больше, мы можем обсудить это интересное поведение MongoDB, но, надеюсь, вы уже начинаете понимать, почему традиционная терминология здесь не совсем применима.

Осваиваем селекторы

В дополнение к изученным ранее шести концепциям, есть ещё один немаловажный практический аспект MongoDB, который следует освоить, прежде чем переходить к более сложным темам: это — селекторы запросов. Селектор запросов MongoDB аналогичен предложению where SQL-запроса. Как таковой он используется для поиска, подсчёта, обновления и удаления документов из коллекций. Селектор — это JSON-объект, в простейшем случае это может быть даже {}, что означает выборку всех документов (аналогичным образом работает null). Если нам нужно выбрать всех единорогов (англ. «unicorns») женского рода, можно воспользоваться селектором {gender:'f'}.

Прежде, чем мы глубоко погрузимся в селекторы, давайте сначала создадим немного данных, с которыми будем экспериментировать. Сперва давайте удалим всё, что до этого вставляли в коллекцию unicorns с помощью команды: db.unicorns.remove() (поскольку мы не передали селектора, произойдёт удаление всех документов). Теперь давайте произведём следующие вставки, чтобы получить данные для дальнейших экспериментов (можете скопировать и вставить это в консоль):

```
db.unicorns.insert({name: 'Horny', dob: new Date(1992,2,13,7,47), loves:
['carrot','papaya'], weight: 600, gender: 'm', vampires: 63});
db.unicorns.insert({name: 'Aurora', dob: new Date(1991, 0, 24, 13, 0), loves: ['carrot',
'grape'], weight: 450, gender: 'f', vampires: 43});
db.unicorns.insert({name: 'Unicrom', dob: new Date(1973, 1, 9, 22, 10), loves:
['energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});
db.unicorns.insert({name: 'Roooooodles', dob: new Date(1979, 7, 18, 18, 44), loves:
['apple'], weight: 575, gender: 'm', vampires: 99});
db.unicorns.insert({name: 'Solnara', dob: new Date(1985, 6, 4, 2, 1), loves:['apple',
'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});
db.unicorns.insert({name:'Ayna', dob: new Date(1998, 2, 7, 8, 30), loves:
['strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});
db.unicorns.insert({name:'Kenny', dob: new Date(1997, 6, 1, 10, 42), loves: ['grape',
'lemon'], weight: 690,  gender: 'm', vampires: 39});
db.unicorns.insert({name: 'Raleigh', dob: new Date(2005, 4, 3, 0, 57), loves: ['apple',
'sugar'], weight: 421, gender: 'm', vampires: 2});
db.unicorns.insert({name: 'Leia', dob: new Date(2001, 9, 8, 14, 53), loves: ['apple',
'watermelon'], weight: 601, gender: 'f', vampires: 33});
db.unicorns.insert({name: 'Pilot', dob: new Date(1997, 2, 1, 5, 3), loves: ['apple',
'watermelon'], weight: 650, gender: 'm', vampires: 54});
db.unicorns.insert({name: 'Nimue', dob: new Date(1999, 11, 20, 16, 15), loves:
['grape', 'carrot'], weight: 540, gender: 'f'});
db.unicorns.insert({name: 'Dunx', dob: new Date(1976, 6, 18, 18, 18), loves: ['grape',
'watermelon'], weight: 704, gender: 'm', vampires: 165});
```

Теперь, когда данные созданы, можно приступать к освоению селекторов. {поле: значение}используется для поиска всех документов, у которых поле равно значение.{поле1: значение1, поле2: значение2} работает как логическое И. Специальные операторы $lt,$lte, $gt, $gte и $ne используются для выражения операций «меньше», «меньше или равно», «больше», «больше или равно», и «не равно». Например, чтобы получить всех самцов единорога, весящих более 700 фунтов, мы можем написать:

```
db.unicorns.find({gender: 'm', weight: {$gt: 700}})
//или (что не полностью эквивалентно, но приведено здесь в демонстрационных
целях)
db.unicorns.find({gender: {$ne: 'f'}, weight: {$gte: 701}})
```

Оператор $exists используется для проверки наличия или отсутствия поля, например:

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

Вернёт единственный документ. Если нужно ИЛИ вместо И, мы можем использовать оператор $or и присвоить ему массив значений, например:

db.unicorns.find({gender: 'f', $or: [{loves: 'apple'}, {loves: 'orange'}, {weight: {$lt: 500}}]})

Вышеуказанный запрос вернёт всех самок единорогов, которые или любят яблоки, или любят апельсины, или весят менее 500 фунтов.

В нашем последнем примере произошло кое-что интересное. Вы заметили — поле loves это массив. MongoDB поддерживает массивы как объекты первого класса. Это потрясающе удобная возможность. Начав это использовать, вы удивитесь, как вы раньше жили без этого. Самое интересное это та простота, с которой делается выборка по значению массива: {loves: 'watermelon'} вернёт нам все документы, у которых watermelon является одним из значений поля loves.

Это ещё не все операторы. Самый гибкий оператор — $where, позволяющий нам передавать JavaScript для его выполнения на сервере. Это описано в разделе Сложные запросы на сайте MongoDB. Мы изучили основы, которые нам нужны для начала работы. Это также то, что вы будете использовать большую часть времени.

Мы видели, как эти селекторы могут быть использованы с командой find. Они также могут быть использованы с командой remove, которую мы кратко рассмотрели, командой count, на которую мы пока не взглянули, но которую вы скорее всего изучите, и командой update, с которой в дальнейшем мы проведём большую часть времени.

ObjectId, сгенерированный MongoDB для поля _id, подставляется в селектор следующим образом:

db.unicorns.find({_id: ObjectId("TheObjectId")})

Выводы к лабораторной работе

Мы пока ещё не рассматривали команду update или более интересные вещи, которые можно сделать с помощью find. Однако мы подняли MongoDB, кратко изучили команды insert и remove (изучив практически всё, что о них можно изучить) . Мы также начали исследовать find и узнали что такое селекторы MongoDB. Это неплохо для начала, и основы для дальнейшего изучения заложены. Верите или нет, но вы уже изучили практически всё, что нужно знать о MongoDB — настолько она проста и легка в изучении. Я настоятельно

рекомендую вам поэкспериментировать с вашими данными, прежде, чем можно будет двигаться дальше. Вставьте несколько новых документов — возможно в новые коллекции — и поэкспериментируйте с селекторами. Используйте find, count и remove. После нескольких ваших собственных попыток вещи, казавшиеся непонятными, станут на свои места.

Требования к сдаче лабораторной работы

1. Отчет к лабораторной работе, содержащий пошаговое исполнение заданий и промежуточные результаты

Задания к лабораторной работе

1. Установите MongoDB.
2. С помощью интерпретатора mongo создайте базы данных фамилия_db_test_1, фамилия_db_test_2, фамилия_db_test_3.
3. В каждой базе данных создайте коллекцию. Назовите их col_test_N1 (где N — номер базы данных).
4. В каждой коллекции добавьте по 5-10 документов. Используйте 3-5 текстовых и числовых полей, и по крайней мере одно поле-массив. Список полей может немного различаться в пределах одной коллекции.
5. Выведите список коллекций каждой базы данных.
6. Выведите каждую коллекцию.
7. Для коллекции из первой базы данных проведите 10 запросов, используя условия равенства, неравенства, больше, меньше, больше или равно, меньше или равно, наличия и отсутствия определенного поля.
8. Удалите все документы из коллекции второй базы данных.
9. Для коллекции из третьей базы данных проведите 5 запросов, используя условия вхождения в массив, логических связок И и ИЛИ.
10. Выведите каждую коллекцию.
11. Проведите один любой запрос на удаление для коллекции третьей базы данных.
12. Выведите коллекцию третьей базы данных.