MongoDB

Scope: Basics

Ramu RC

1

Data Basics

2

JSON

- "JavaScript Object Notation"
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - Ordered list of values

http://json.org/

3

Document(JSON) structure

- The document has simple structure and very easy to understand the content
- JSON is smaller, faster and lightweight compared to XML.
- For data delivery between servers and browsers, JSON is a better choice
- Easy in parsing, processing, validating in all languages
- JSON can be mapped more easily into object oriented system.

4

BSON

- "Binary JSON"
- Binary-encoded serialization of JSON-like docs
- Also allows "referencing"
- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

http://bsonspec.org/

5

BSON Example

```
{
"_id" :    "37010"
"city" :   "ADAMS",
"pop" :    2660,
"state" :  "TN",
"councilman" : {
            name: "John Smith"
            address: "13 Scenic Way"
          }
}
```

6

BSON Types

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The number can be used with the $type operator to query by type!

http://docs.mongodb.org/manual/reference/bson-types/

7

XML And JSON

XML	JSON
It is a markup language.	It is a way of representing objects.
This is more verbose than JSON.	This format uses less words.
It is used to describe the structured data.	It is used to describe unstructured data which include arrays.
JavaScript functions like *eval*(), *parse*() doesn't work here.	When *eval* method is applied to JSON it returns the described object.

Example:
<car> <company>Volkswagen</company> <name>Vento</name>
<price>800000</price> </car>
	{
	"company": Volkswagen,
	"name": "Vento",
	"price": 800000
	}

8

Why JSON?

- JSON is faster and easier than XML when you are using it in AJAX web applications:
- **Steps involved in exchanging data from web server to browser involves:**

Using XML

1. Fetch an XML document from web server.
2. Use the XML DOM to loop through the document.
3. Extract values and store in variables.
4. It also involves type conversions.

Using JSON

1. Fetch a JSON string.
2. Parse the JSON string using eval() or parse() JavaScript functions.

9

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is "small, likely unique, fast to generate, and ordered." Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

http://docs.mongodb.org/manual/reference/bson-types/

10

DB Families

11

The family of NoSQL DBs

- **Key-values Stores**
  - Hash table where there is a unique key and a pointer to a particular item of data.
  - Focus on scaling to huge amounts of data
  - *E.g. Redis, Oracle BDB*
- **Column Family Stores**
  - To store and process very large amounts of data distributed over many machines
  - *E.g. Cassandra, HBase*
- **Document Databases**
  - Collections of Key-Value collections
  - The next level of Key/value, allowing nested values associated with each key.
  - Appropriate for Web apps.
  - *E.g. CouchDB, MongoDB*
- **Graph Databases**
  - Bases on property-graph model
  - Appropriate for Social networking, Recommendations
  - *E.g. Neo4J, Infinite Graph, Orient DB*

12

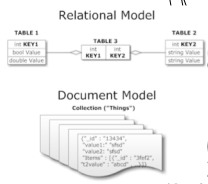
12

Other NoSQL Types

Key/value (Dynamo)

Columnar/tabular (HBase)

Document (mongoDB)



http://www.aaronstannard.com/post/2011/06/30/MongoDB-vs-SQL-Server.aspx

13

Types of No SQl data base

- **Key Value pair**

Dynamo DB
Azure Table Storage (ATS)

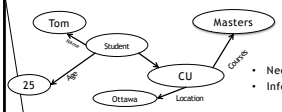
(#key,#value)
(Name, Tom)
(Age,25)
(Role, Student)
(University, CU)

- **Document Based**

```
[
  {
    "Name": "Tom",
    "Age": 30,
    "Role": "Student",
    "University": "CU",
  }
]
```

Mango Db
AmazonSimple DB
Couch DB

- **Graph database**



- Neo4j
- Infogrid

- **Column Oriented database**

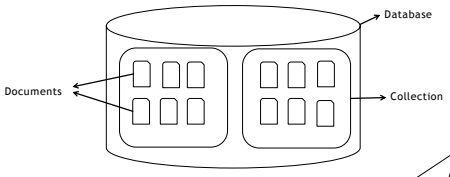
Row Id	Columns	
1	Name	Tom
	Age	25
	Role	Student

Bigtable (Google)
HBase

14

Mongo DB architecture

Details



15

Why we select MongoDB?

16

Why Mongo DB?

- All the modern applications deals with huge data.
- Development with ease is possible with mongo DB.
- Flexibility in deployment.
- Rich Queries.
- Older database systems may not be compatible with the design.

And it's a document oriented storage:- Data is stored in the form of JSON Style.

17

MongoDB

- Document-oriented NoSQL database.
- Schema-free.
- Based on Binary JSON; BSON.
- Organized in Group of Documents → Collections
  - Informal namespacing
- Auto-Sharding in order to scale horizontally.
- Simple query language. Rich, document-based queries.
- Map/Reduce support
- Open Source (GNU AGPL v3.0.)

18

Brief

- mongoDB = "Humongous DB"
  - Open-source
  - Document-based
  - High performance, high availability
  - Automatic scaling
  - C-P on CAP

19

Motivations

- Problems with SQL
  - Rigid schema
  - Not easily scalable (designed for 90's technology or worse)
  - Requires unintuitive joins
- Perks of mongoDB
  - Easy interface with common languages (Java, Javascript, PHP, etc.)
  - DB tech should run anywhere (VM's, cloud, etc.)
  - Keeps essential features of RDBMS's while learning from key-value noSQL systems

http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

20

Modeling

21

Data Model

- Document-Based (max 16 MB)
- Documents are in BSON format, consisting of field-value pairs
- Each document stored in a collection
- Collections
  - Have index set in common
  - Like tables of relational db's.
  - Documents do not have to have uniform structure

-docs.mongodb.org/manual/

22

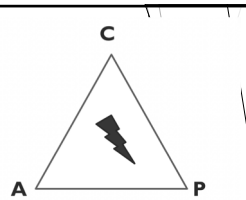
Ops Basics

23

CAP

GIVEN:
- Many nodes
- Nodes contain ***replicas of partitions*** of the data

- Consistency
  - All replicas contain the same version of data
  - Client always has the same view of the data (no matter what node)
- Availability
  - System remains operational on failing nodes
  - All clients can always read and write
- Partition tolerance
  - multiple entry points
  - System remains operational on system split (communication malfunction)
  - System works well across physical network partitions



CAP Theorem: satisfying all three at the same time is impossible

24

MongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition

25

CRUD Ops

26

CRUD: Using the Shell (cont.)

To insert documents into a collection/make a new collection:

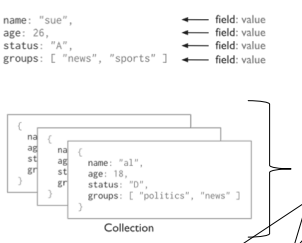
```
db.<collection>.insert(<document>)
```

<=>

```
INSERT INTO <table>
VALUES(<attributevalues>);
```

27

Basic operations



28

CRUD: Inserting Data

Insert one document

```
db.<collection>.insert({<field>:<value>})
```

Inserting a document with a field name new to the collection is inherently supported by the BSON model.

To insert multiple documents, use an array.

29

CRUD operations - create

Insert a new user.

SQL

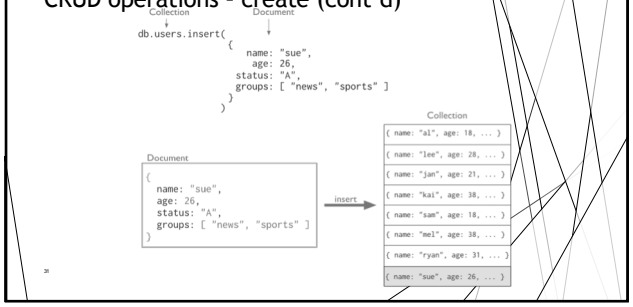
```
INSERT INTO users                    <-- table
        ( name, age, status )        <-- columns
VALUES  ( "sue", 26, "A" )           <-- values/row
```

MongoDB

```
db.users.insert (                    <-- collection
   {
     name: "sue",                    <-- field: value
     age: 26,                        <-- field: value
     status: "A"                     <-- field: value
   }                                 } document
)
```

30

CRUD operations – create (cont'd)



31

CRUD: Querying

- Done on collections.
- Get all docs: db.<collection>.find()
  - Returns a cursor, which is iterated over shell to display first 20 results.
  - Add .limit(<number>) to limit results
  - SELECT * FROM <table>;
- Get one doc: db.<collection>.findOne()

32

CRUD: Querying

To match a specific value:

```
db.<collection>.find({<field>:<value>})
"AND"
db.<collection>.find({<field1>:<value1>,
                      <field2>:<value2>
                     })
```

```
SELECT *
FROM <table>
WHERE <field1> = <value1> AND <field2> = <value2>;
```

33

CRUD: Querying : OR

```
db.<collection>.find({ $or: [
<field>:<value1>
<field>:<value2>        ]
})
```

```
SELECT *
FROM <table>
WHERE <field> = <value1> OR <field> = <value2>;
```

Checking for multiple values of same field

```
db.<collection>.find({<field>: {$in [<value>, <value>]}})
```

34

CRUD: Querying

Including/excluding document fields

```
db.<collection>.find({<field1>:<value>}, {<field2>: 0})
```

```
SELECT field1
FROM <table>;
```

```
db.<collection>.find({<field>:<value>}, {<field2>: 1})
```

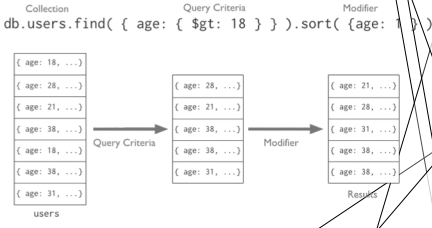
Find documents with or w/o field

```
db.<collection>.find({<field>: { $exists: true}})
```

35

CRUD operations - read

Find the users of age greater than 18 and sort by age.



36

CRUD: Updating

```
db.<collection>.update(
{<field1>:<value1>}, //all docs in which field = value
{$set: {<field2>:<value2>}},        //set field to value
{multi:true} )          //update multiple docs
```

upsert: if true, creates a new doc when none matches search criteria.

```
UPDATE <table>
SET <field2> = <value2>
WHERE <field1> = <value1>;
```

37

CRUD: Updating

To remove a field

```
db.<collection>.update({<field>:<value>},
        { $unset: { <field>: 1}})
```

Replace all field-value pairs

```
db.<collection>.update({<field>:<value>},
        { <field>:<value>,
      <field>:<value>})
```

*NOTE: This overwrites ALL the contents of a document, even removing fields.

38

CRUD operations - update

Update the users of age greater than 18 by setting the status field to A.

SQL

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

MongoDB

```
db.users.update(                  <-- collection
   { age: { $gt: 18 } },          <-- update criteria
   { $set: { status: "A" } },     <-- update action
   { multi: true }                <-- update option
)
```

39

CRUD: Removal

Remove all records where field = value

```
db.<collection>.remove({<field>:<value>})
```

```
DELETE FROM <table>
WHERE <field> = <value>;
```

As above, but only remove first document

```
db.<collection>.remove({<field>:<value>},true)
```

40

CRUD operations - delete

Delete the users with status equal to D.

SQL

```
DELETE FROM users     <-- table
WHERE  status = 'D'   <-- delete criteria
```

MongoDB

```
db.users.remove(      <-- collection
   { status: "D" }    <-- remove criteria
)
```

41

For more info:

- http://docs.mongodb.org/manual/core/write-operations/
- http://docs.mongodb.org/manual/crud/
- https://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1
- https://mongodb.org/manual

42