MongoDB

What is it

> MongoDB (from *humongous*) is a scalable, high-performance, open source, schema-free, document-oriented database.
> -- mongodb.org

Thinking in documents

When building applications and data, it's about more than the DB.

- Easily understood structures
- Harmonized with software and real world use
- Planning for the future

Ideas behind MongoDB

RDBMS's are sometimes really awkward to fit to some of today's problems.

- static, uniform scalar data
- rectangles
- low-level, leverage/match the hardware design AND programming languages of the day
- scaling RDB can be a challenge
- difficult to make an RDB an "on demand" service

MongoDB's focus is on performance and flexibility and scaling

- flexible rich shapes of data
- easily match objects to database entries
- higher level, business entity representation
- non-platform specific
- more often via dynamic programming languages like python, ruby, java/javascript/node.js
- returning cursors (Python) or maps (Java)
- while retaining many of the more useful RDB ideas
    - indexing, distributed, etc.

It is a document-oriented database

EVERYTHING is a document: `{ ... }`

- Data models, data stored and schemas are all described in Json
- Flexible format
- you can design data models supporting common data access patterns
    - e.g., you can design the data driving a website so that the most frequently viewed pages require a simple query to the database.

HOWEVER ...

- No referential integrity
- No focus on normalization can mean updating something in many places instead of one
- Lack of predefined schema is a double- edged sword
    - You must have a model in your app
    - Objects within a collection can be completely inconsistent in their fields

JSON and BSON

Everything is expressed in BSON ("binary" JSON)

MongoDB understands JSON **natively**

- From MongoDB Documentation:

> JavaScript Object Notation (JSON) is an open, human and machine-readable standard that facilitates data interchange, and along with XML is the main format for data interchange used on the modern web. JSON supports all the basic data types you'd expect: numbers, strings, and boolean values, as well as arrays and hashes.
>
> Document databases such as MongoDB use JSON documents in order to store records, just as tables and rows store records in a relational database.
>
> A JSON database returns query results that can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages – reducing the amount of logic you need to build into your application layer.
>
> MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes.
> BSON extends the JSON model
> to provide additional data types, ordered fields, and to be efficient for encoding and decoding within different languages.

For example if we were representing blog articles...

```
{
  "_id": "No-Sushi-Mondays",
  "title": "Why I never eat sushi on Mondays",
  "date": "2017-02-13T21:27:22.104Z",
  "author": {
    "name": "Tony Stark",
    "title": "CEO",
    "company": "Stark Industries"
  },
  "text": "Do you ever wonder just how fresh fish could be when ...",
  "tags": ["sushi", "Mondays", "nausea", "Vibrio parahaemolyticus"],
  "comments": [
    {
      "name": "FishMonger Phil",
      "comment": "What do you know about ..."
    },
    {
      "name": "Dr.Nick",
      "comment": "What's the big deal, a little ..."
    }
  ]
}
```

To do the same thing with a relational database would require several joins across different relational tables.

And because Mongo DB Doesn't have to do these joins, it can be more easily distributed across several databases as shards and applications don't have to know.

Moving from RDB to MongoDB

SLIDES 26-32

- value of any field is a BSON datatype.
- for MongoDB, the `{ "key":"value", ... }` is a *document*
- Example in javascript (which also knows JSON):

```
var mydoc = {
              _id: ObjectId("5099803df3f4948bd2f98391"),
              name: { first: "Alan", last: "Turing" },
              birth: new Date('Jun 23, 1912'),
              death: new Date('Jun 07, 1954'),
              contribs: [ "Turing machine", "Turing test", "Turingery" ],
              views : NumberLong(1250327)
            }
db.posts.insertOne(mydoc);
var c = {author: “eliot”, date: new Date(), text: “great post!”}
db.posts.update({_id: post._id}, {$push: {comments: c}})
```

- `_id` holds an ObjectId.
- `name` holds an embedded document that contains the fields first and last.
- `birth` and death hold values of the Date type.
- `contribs` holds an array of strings.
- `views` holds a value of the NumberLong type.

Naming guidelines

- Use *camelCase* for names
- The field name `_id` is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names cannot start with the dollar sign (`$`) character.
- The field names cannot contain the dot (`.`) character.
- The field names cannot contain the null character.

SLIDES 33-57