#mongodb

MongoDB

Modified from slides provided by S. Parikh, A. Im, G. Cai, H. Tunc, J. Stevens, Y. Barve, S. Hei

mongoDB

History

- mongoDB = “Hu**mongo**us DB”
  - Open-source
  - Document-based
  - “High performance, high availability”
  - Automatic scaling
  - C-P on CAP

mongoDB

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

Design Goals

- Scale horizontally over commodity systems
- Incorporate what works for RDBMSs
  - Rich data models, ad-hoc queries, full indexes
- Move away from what doesn't scale easily
  - Multi-row transactions, complex joins
- Use idomatic development APIs
- Match agile development and deployment workflows

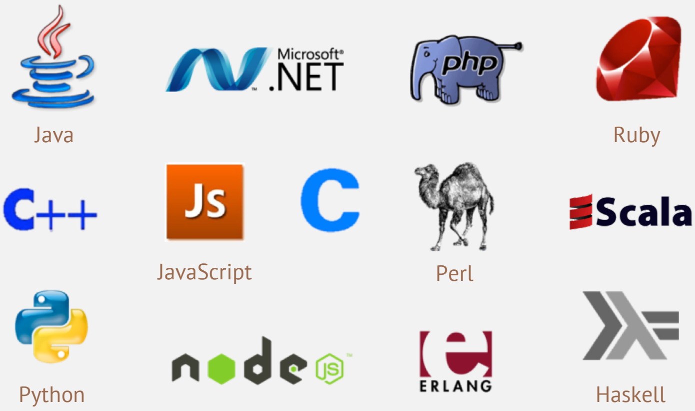
Key Features

- Data stored as documents (JSON)
  - Dynamic-schema
- Full CRUD support (Create, Read, Update, Delete)
  - Ad-hoc queries: Equality, RegEx, Ranges, Geospatial
  - Atomic in-place updates
- Full secondary indexes
  - Unique, sparse, TTL
- Replication – redundancy, failover
- Sharding – partitioning for read/write scalability

MongoDB Drivers and Shell

Drivers

Drivers for most popular programming languages and frameworks



Shell

Command-line shell for interacting directly with database

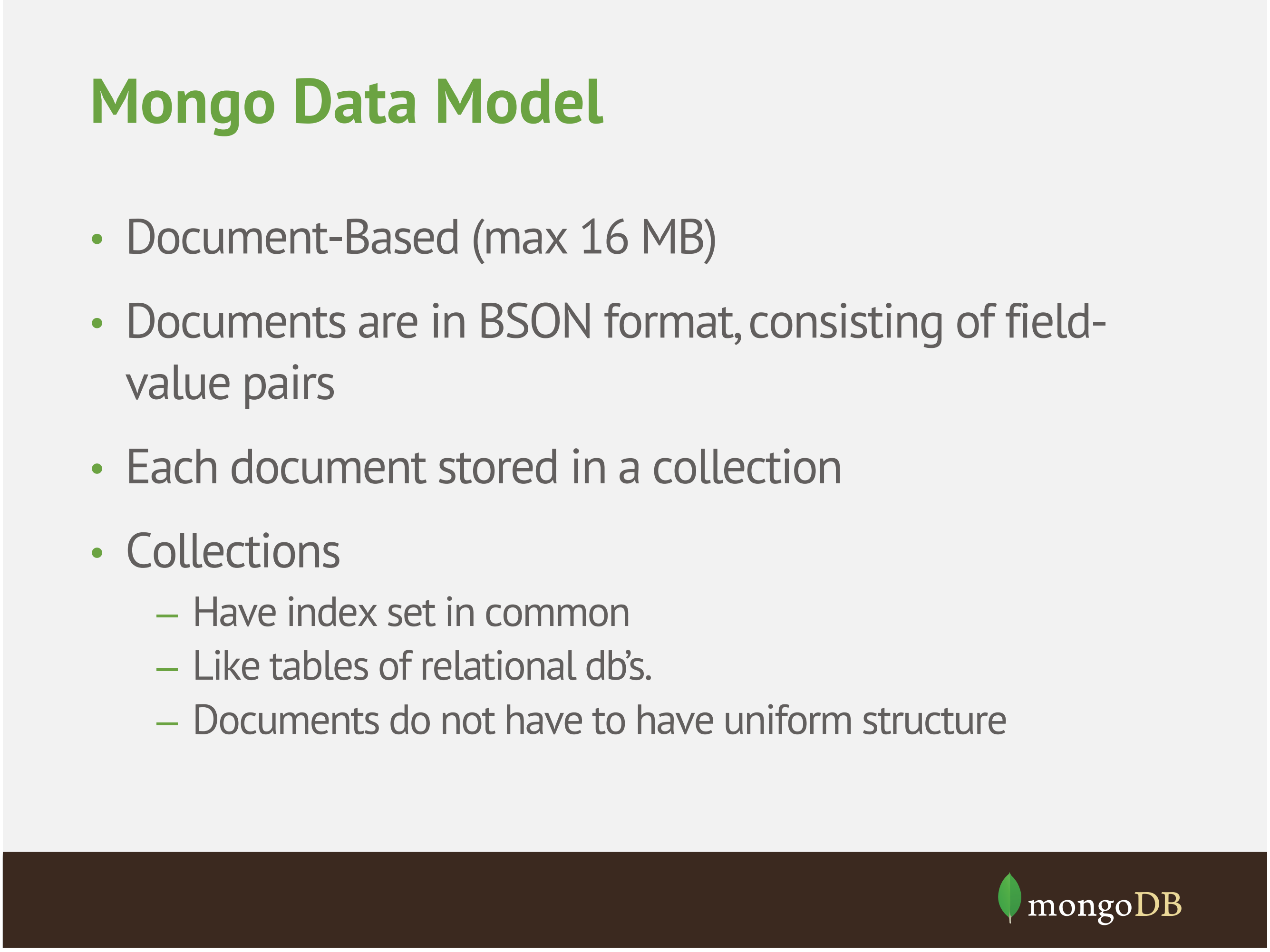
```
> db.collection.insert({product:"MongoDB",
type:"Document Database"})
>
> db.collection.findOne()
{
      "_id"        : ObjectId("5106c1c2fc629bfe52792e86"),
      "product"    : "MongoDB"
      "type"       : "Document Database"
}
```

Getting Started with Mongo

Installation

- Install Mongo from: http://www.mongodb.org/downloads
  - Extract the files
  - Create a data directory for Mongo to use
- Open your mongodb/bin directory and run the binary file (name depends on the architecture) to start the database server.
- To establish a connection to the server, open another command prompt window and go to the same directory, entering in mongo.exe or mongo for macs and Linuxes.
- This engages the mongodb shell—it's that easy!

MongoDB Design Model



Mongo 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

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

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)

BSON Example

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

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!

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.

MongoDB vs. Relational Databases

Why Databases Exist in the First Place?

- Why can’t we just write programs that operate on objects?
  - Memory limit
  - We cannot swap back from disk merely by OS for the page based memory management mechanism
- Why can’t we have the database operating on the same data structure as in program?
  - That is where Mongo comes in

Mongo is basically schema-free

- The purpose of schema in SQL is for meeting the requirements of tables and quirky SQL implementation
- Every “row” in a database “table” is a data structure, much like a “struct” in C, or a “class” in Java.
  - A table is then an array (or list) of such data structures
- So what we design in Mongo is basically similar to how we design a compound data type binding in JSON

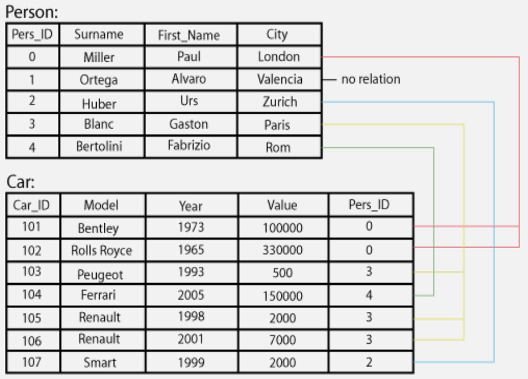
RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Row	→	Document
Index	→	Index
Join	→	Embedded Document
Foreign Key	→	Reference

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

Document Oriented, Dynamic Schema

Relational



Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB

```
{
  first_name: ‘Paul’,
  surname: ‘Miller’
  city: ‘London’,
  location: [45.123,47.232],
  cars: [
    { model: ‘Bentley’,
      year: 1973,
      value: 100000, … },
    { model: ‘Rolls Royce’,
      year: 1965,
      value: 330000, … }
  ]
}
```

CRUD:

Create, Read, Update, Delete

CRUD: Using the Shell

- To check which db you're using → db
- Show all databases → show dbs
- Switch db's/make a new one → use <name>
- See what collections exist → show collections

- Note: db's are not actually created until you insert data!

CRUD: Using the Shell (cont.)

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

- db.<collection>.insert(<document>)
- <=>
- INSERT INTO <table>
- VALUES(<attributevalues>);

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.

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()

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>;
```

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>]}})
```

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}})
```

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
```

Bulk.find.upsert(): if true, creates a new doc when none matches search criteria.

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

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.

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)
```

CRUD: Isolation

- By default, all writes are atomic only on the level of a single document.
- This means that, by default, all writes can be interleaved with other operations.
- You can isolate writes on an unsharded collection by adding $isolated:1 in the query area:
  - db.<collection>.remove({<field>:<value>, $isolated: 1})

mongoDB

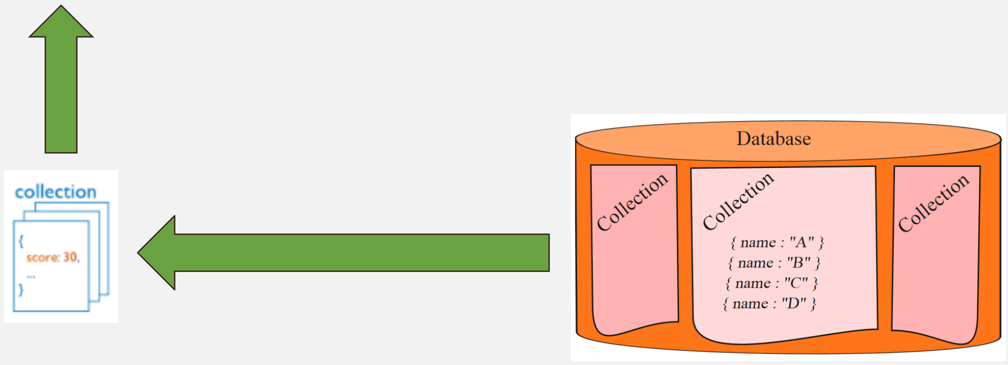
Index in MongoDB

mongoDB

Before Index

- What does database normally do when we query?
  - MongoDB must scan every document.
  - Inefficient because process large volume of data

```
db.users.find( { score: { “$lt” : 30} } )
```



Index in MongoDB: Operations

- Creation index
  - db.users.ensureIndex({ score: 1 })
- Show existing indexes
  - db.users.getIndexes()
- Drop index
  - db.users.dropIndex({score: 1})
- Explain—Explain
  - db.users.find().explain()
  - Returns a document that describes the process and indexes
- Hint
  - db.users.find().hint({score: 1})
  - Overide MongoDB’s default index selection

Index in MongoDB

- Types
  - **Single Field Indexes**
  - Compound Field Indexes
  - Multikey Indexes
- Single Field Indexes
  - db.users.ensureIndex({ score: 1 })

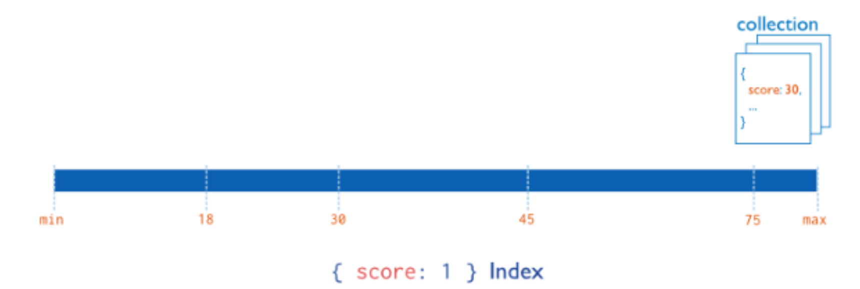


Diagram of an index on the score field (ascending).

Index in MongoDB

- Types
  - Single Field Indexes
  - **Compound Field Indexes**
  - Multikey Indexes
- Compound Field Indexes
  - db.users.ensureIndex({ userid:1,score:-1 })

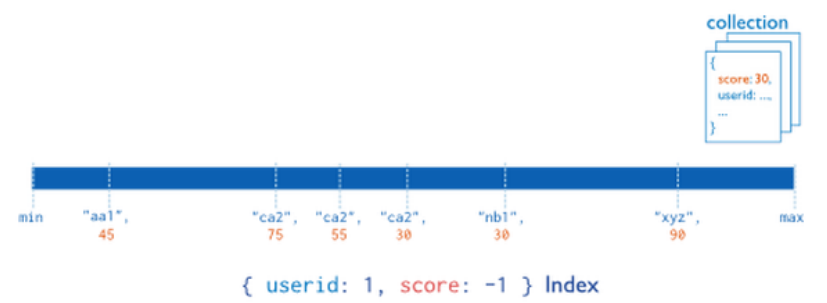


Diagram of a compound index on the userid field (ascending) and the score field (descending). The index sorts first by the userid field and then by the score field.

Index in MongoDB

- Types
  - Single Field Indexes
  - Compound Field Indexes
  - Multikey Indexes
- Compound Field Indexes
  - db.users.ensureIndex({ userid:1,score:-1 })

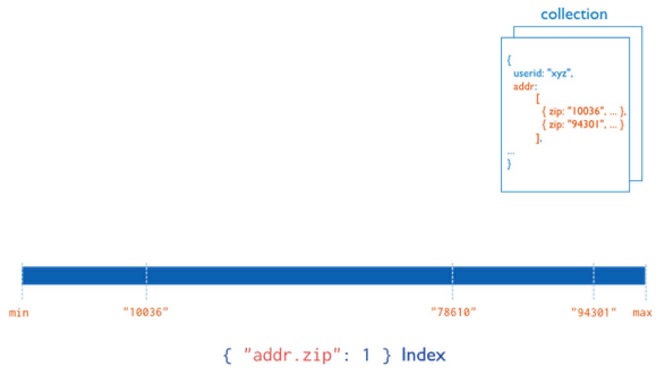


Diagram of a multikey index on the `addr.zip` field. The `addr` field contains an array of address documents. The address documents contain the `zip` field.

Index in MongoDB

- Types
  - Single Field Indexes
  - Compound Field Indexes
  - Multikey Indexes
- Multikey Indexes
  - db.users.ensureIndex({ addr.zip:1})

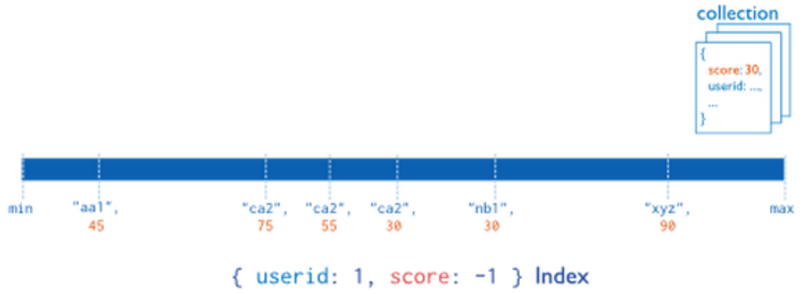


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

Mongo Examples

Documents

```
> var new_entry = {
  firstname: “John”,
  lastname: “Smith”,
  age: 25,
  address: {
    street: “21 2nd Street”,
    city: “New York”,
    state: “NY”,
    zipcode: 10021
  }
}
> db.addressBook.save(new_entry)
```

Querying

```
> db.addressBook.find()
{
  _id: ObjectId(“4c4ba5c0672c685e5e8aabf3”),
  firstname: “John”,
  lastname: “Smith”,
  age: 25,
  address: {
    street: “21 2nd Street”, city: “New York”,
    state: “NY”, zipcode: 10021
  }
}
// _id is unique but can be anything you like
```

Indexes

```
// create an ascending index on “state”
> db.addressBook.ensureIndex({state:1})

> db.addressBook.find({state:”NY”})
{
  _id: ObjectId(“4c4ba5c0672c685e5e8aabf3”),
  firstname: “John”,
  …
}

> db.addressBook.find({state:”NY”, zip: 10021})
```

Queries

```
// Query Operators:
// $all, $exists, $mod, $ne, $in, $nin, $nor, $or,
// $size, $type, $lt, $lte, $gt, $gte

// find contacts with any age
> db.addressBook.find({age: {$exists: true}})

// find entries matching a regular expression
> db.addressBook.find( {lastname: /^smi*/i } )

// count entries with “John”
> db.addressBook.find( {firstname: ‘John’} ).count()
```

Updates

```
// Update operators
// $set, $unset, $inc, $push, $pushAll, $pull,
// $pullAll, $bit

> var new_phonenumber = {
  type: “mobile”,
  number: “646-555-4567”
}

> db.addressBook.update({ _id: “...” }, {
  $push: {phonenumbers: new_phonenumber}
});
```

Nested Documents

```
{
  _id: ObjectId("4c4ba5c0672c685e5e8aabf3"),
  firstname: “John”, lastname: “Smith”,
  age: 25,
  address: {
    street: “21 2nd Street”, city: “New York”,
    state: “NY”, zipcode: 10021
  }
  phonenumbers : [ {
    type: “mobile”, number: “646-555-4567”
  } ]
}
```

Secondary Indexes

```
// Index nested documents
> db.addressBook.ensureIndex({“phonenumbers.type”:1})

// Geospatial indexes, 2d or 2dsphere
> db.addressBook.ensureIndex({location: “2d”})
> db.addressBook.find({location: {$near: [22,42]}})

// Unique and Sparse indexes
> db.addressBook.ensureIndex({field:1}, {unique:true})
> db.addressBook.ensureIndex({field:1}, {sparse:true})
```

Additional Features

- Geospatial queries
  - Simple 2D plane
  - Or accounting for the surface of the earth (ellipsoid)
- Full Text Search
- Aggregation Framework
  - Similar to SQL GROUP BY operator
- Javascript MapReduce
  - Complex aggregation tasks

mongoDB

MongoDB Development

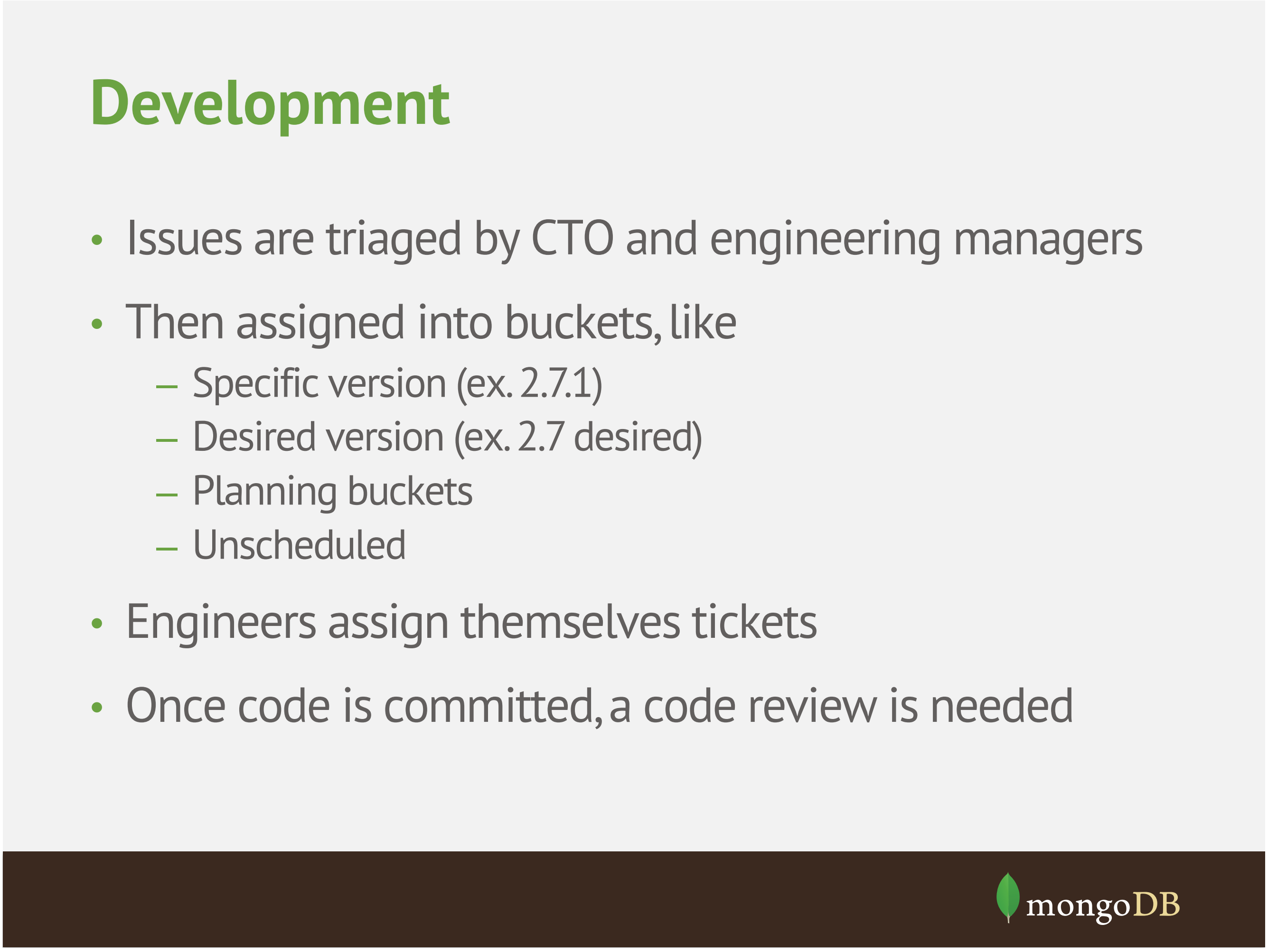
mongoDB

Open Source

- MongoDB source code is on Github
  - https://github.com/mongodb/mongo
- Issue tracking for MongoDB and drivers
  - http://jira.mongodb.org

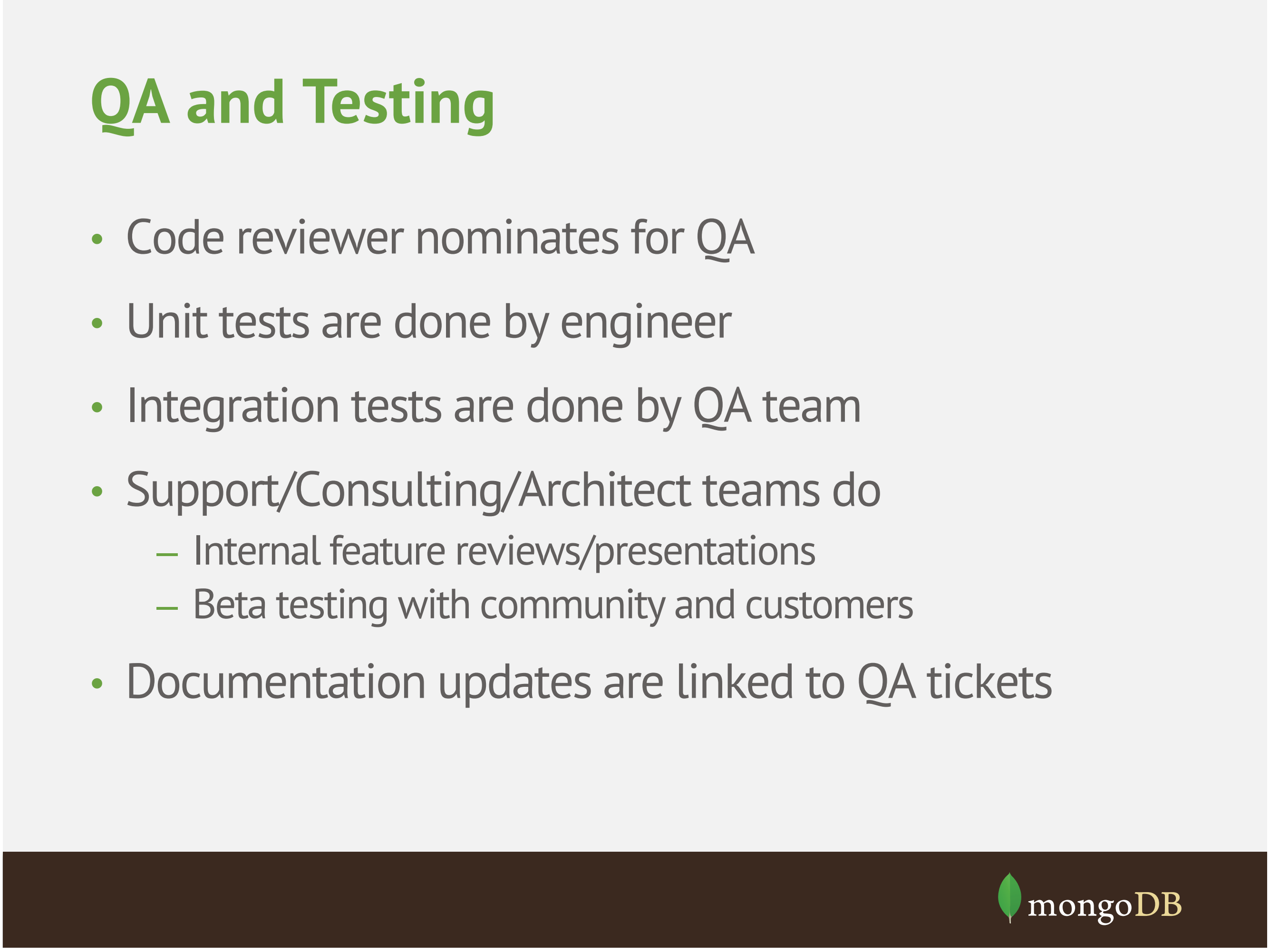
Support

- Tickets are created by
  - Customer support
  - Community support (Google Groups, StackOverflow)
  - Community members
  - MongoDB employees
- Tickets can be voted on and watched to track progress
- Follow-the-Sun support
- All technical folks spend time doing community and customer support



Development

- Issues are triaged by CTO and engineering managers
- Then assigned into buckets, like
  - Specific version (ex. 2.7.1)
  - Desired version (ex. 2.7 desired)
  - Planning buckets
  - Unscheduled
- Engineers assign themselves tickets
- Once code is committed, a code review is needed



QA and Testing

- Code reviewer nominates for QA
- Unit tests are done by engineer
- Integration tests are done by QA team
- Support/Consulting/Architect teams do
  - Internal feature reviews/presentations
  - Beta testing with community and customers
- Documentation updates are linked to QA tickets

Questions?

- Sandeep Parikh
  - sap@mongodb.com
  - @crcsmnky
- MongoDB
  - MongoDB, drivers, documentation
    - http://www.mongodb.org
    - http://docs.mongodb.org
  - Free online training, presentations, whitepapers
    - http://www.mongodb.com