Schema Design

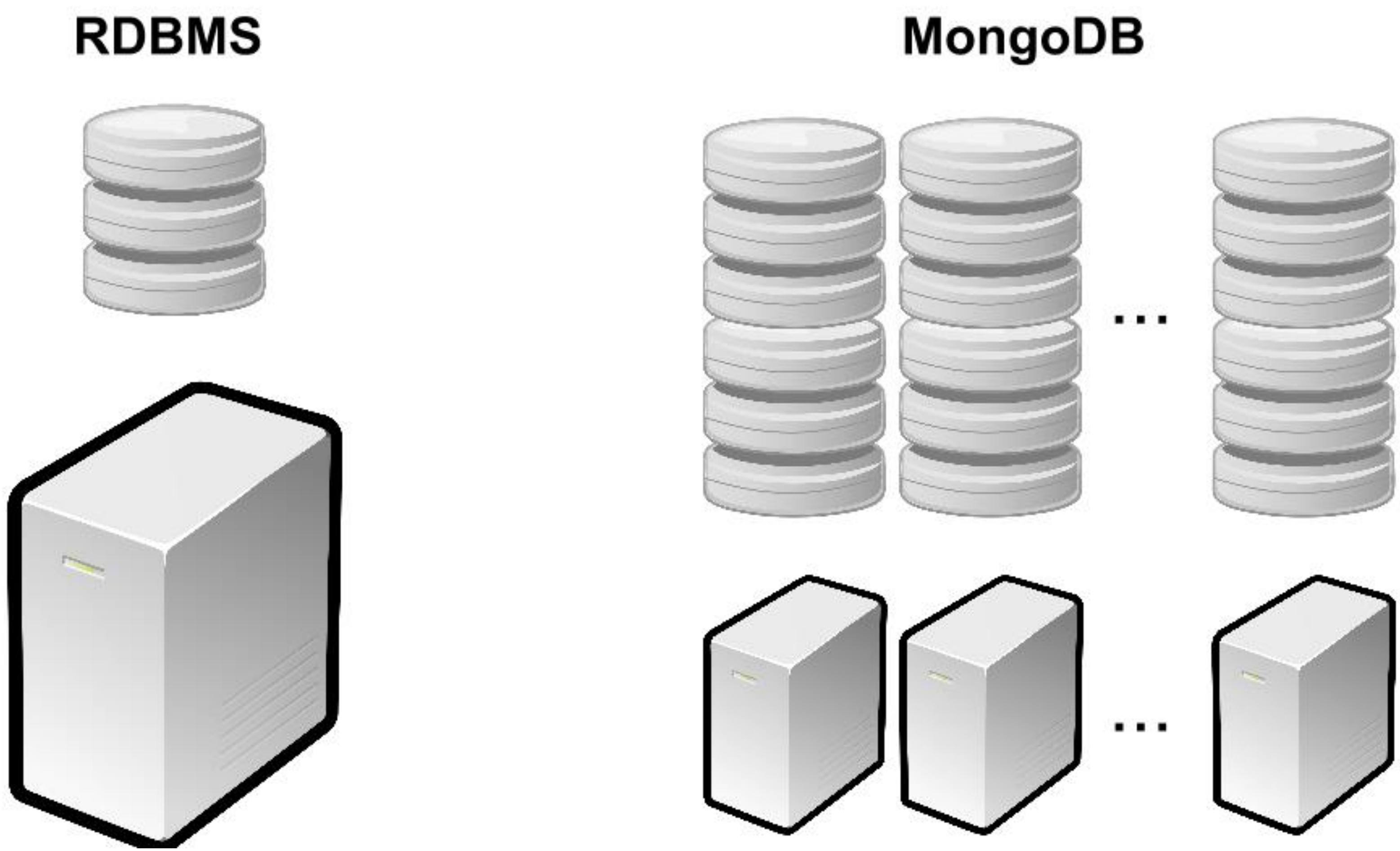
http://docs.mongodb.org/manual/data-modeling/

A new database type

- **NoSQL** or **Post-relational**

  "NoSQL databases are often very fast, do not require fixed table schemas, avoid join operations by storing denormalized data, and are designed to scale horizontally" (Wikipedia)

The NoSQL World is different than the RDBMS World



Get things in the right order

Relational World	NoSQL World
1 - Model the data	1 - Define the access to the data
2 - Write the queries to access the data	2 - Model the data

MongoDB has a *flexible* schema

Schemaless just means that collections do not enforce document structure

This makes it easy to get started (just shove in your data and start querying) and easy to make changes later as you come to understand your application.

If you don't have a schema, you're probably doing it wrong!

Design the document schema to suit your application.

Model is dependent on limitations of the product

- Core Server
  - Max size of document: 16 MB
  - Simple update may result in full copy of document in replication protocol
  - Full documents get read from disk
  - Atomic update at the document level
  - Can't query elements in an array, within another array
- Storage engine
  - MMAPv1
    - Growing documents may move (update index entries)
  - WiredTiger
    - Each update rewrites the whole document

Model: the Collections

<u>movies</u>
_id: ObjectId
title: string
year: int32
...
actual:

- revenues: int64
- cost: int64

ratings: []

- source: string
- rating: double

```
movies
{
 _id: ObjectId("56fe9ed11349fb4a9972b2ac")
 title: "Star Wars"
 year: 1977
 ...
 actual: {
  revenues: 787,000,000
  cost: 11,000,000
 }
 ratings: [
  { source: "NYT",
    rating: 4 },
  { source: "Pravda",
    rating: 1 },
    ...
 ]
}
```

Cardinalities

- [10]
  - Exactly 10 elements
- [0, 20]
  - Minimum of 0 elements
  - Maximum of 20 elements
- [0, 10, 1000000]
  - Minimum of 0 elements
  - Median of 10 elements
  - Maximum of 1000000 elements

One-to-One Relationship (Embedding)

movies
_id: ObjectId
title: string
year: int32
...
estimate:

> revenues: int64
> cost: int64
> advertising: int64

actual:

> revenues: int64
> cost: int64
> advertising: int64

mongoDB

One-to-One Relationship (Linking)

<u>movies</u>
_id: ObjectId
title: string
year: int32
…

<u>movieDollars</u>
_id: ObjectId
movieTitle: string
movieId: ObjectId
…
estimate:

- revenues: int64
- budget: int64
- advertising: int64

actual:

- revenues: int64
- budget: int64
- advertising: int64

One-to-One Relationship (Embedding and Linking)

<u>movies</u>
_id: ObjectId
title: string
year: int32
…
actual
- revenues: int64
- budget: int64

<u>movieDollars</u>
_id: ObjectId
movieTitle: string
movieId: ObjectId
…
estimate:
- revenues: int64
- budget: int64
- advertising: int64

actual:
- revenues: int64
- budget: int64
- advertising: int64

One-to-Many (Few) Relationship (Embedding)

<u>movies</u>
_id: ObjectId
title: string
…
ratings: []

> source: string
> rating: double

One-to-Many Relationship (Left Array of IDs)

<u>movies</u>
_id: ObjectId
title: string
…
quotesIds: ObjectId[]

<u>quotes</u>
_id: ObjectId
actorId: ObjectId
actorName: string
…
quote: string

mongoDB

One-to-Many (a lot) Relationship (Right Linking)

movies
_id: ObjectId
title: string
…

viewings
_id: ObjectId
movieId: ObjectId
dateTime: datetime
…
viewers: int32
revenues: int64
location:
- theater: string
- city: string
- country: string
- …

mongoDB

Many-to-Many Relationship (Left Array of IDs and Right Array of IDs)

movies
_id: ObjectId
title: string
…
actors: []

- actorId: ObjectId
- name: string
- role: string

actors
_id: ObjectId
name: string
birth: datetime
…
movies: []

- movieId: ObjectId
- movieName: string

mongoDB

Many-to-Many Relationship (Left Array of IDs)

movies
_id: ObjectId
title: string
…
actors: []

    actorId: ObjectId
    name: string
    role: string

actors
_id: ObjectId
name: string
birth: datetime
…

Many-to-Many (a lot) Relationship (Right Array of IDs)

<u>movies</u>
_id: ObjectId
title: string
…

<u>viewers</u>
_id: ObjectId
name: string
…
movies: []
    movieId: ObjectId
    dates: datetime[]

Subset Pattern

movies
_id: ObjectId
title: string
…
mainActors: [0…20]

moviePeopleId: ObjectId
role: string

moviePeople
_id: ObjectId
name: string
…
roles: []

movie: ObjectId
role: string

mongoDB

Aggregate Pattern

movies
_id: ObjectId
title: string
…
viewers: int64
revenues: int64
viewingCount: int32

viewings
_id: ObjectId
movieId: ObjectId
dateTime: datetime
…
viewers: int64
revenues: int64
location:
theater: string
city: string
country: string
…

Cache Pattern

movies
_id: ObjectId
title: string
…
topReviews: [0…20]

> reviewId: ObjectId
> reviewer: string
> rating: int32
> …
> review: string

reviews
_id: ObjectId
reviewer: string
rating: int32
…
review: string
comments: []

> viewerId: ObjectId
> comment: string

Tree (node) Pattern

categories
_id: string
parent: string

Tree (leaves) Pattern

categories
_id: string
children: string[]

Tree (branch) Pattern

<u>categories</u>
_id: string
parent: string
ancestors: string[]

Miscellaneous notes

- Field names take up space
  - This is not very important with fewer documents, but when you get into the billions of records, they have a meaningful impact on your index size.

- Disk space is cheap but RAM is not, and you want as much data in memory as possible.

The bad

- We don't have a representation to suggest to our customers other than model by example

- A schema is optimized for one application, a second application, sharing the same data, may not be optimized for the schema

Group Exercise

- Description
  - Come up with a basic but reasonable data model for an e-commerce site. For users of RDBMSs, the most challenging part of the exercise will be figuring out how to construct a data model when database-level joins aren't allowed.
- Deliverables
  - Sample document and schema for each collection
  - Queries the application will use
  - Index definitions

Group Exercise

Model for the following entities and features**. Other requirement? Use Amazon's web site**

1. **Products.** Products vary quite a bit. In addition to the standard production attributes, we will allow for variations of product type and custom attributes. E.g., users may search for blue jackets, 11-inch macbooks, or size 12 shoes. The product catalog will contain millions of products.
2. **Product pricing**. Current prices as well as price histories.
3. **Product categories.** Every e-commerce site includes a category hierarchy. We need to allow for both that hierarchy and the many-to-many relationship between products and categories.
4. **Product reviews.** Every product has zero or more reviews and each review can receive votes and comments.
5. **Product views and purchases**. Keep track of the number of times each product is viewed and when each product is purchased.
6. **Top 10 lists**. Create queries for top 10 viewed products, top 10 purchased products.
7. **Graph historical trends.** Create a query to graph how a product is viewed/purchased over the past 30 days with 1 hour granularity. This graph will appear on every product page, the query must be very fast.

References

- http://docs.mongodb.org/manual/reference/sql-comparison/
- http://docs.mongodb.org/manual/applications/data-models-relationships/
- http://www.slideshare.net/danielcoupal/semi-formal-model-for-document-oriented-databases