NSQ1, Session 4

MongoDB Database design

You forgot transactions

Modification: INSERT, UPDATE, DELETE

2^+ Modifications <u>require</u> explicit transactions

Example:

INSERT INTO Order ...

UPDATE Book SET count = count - 1 ← What happens when this fails? Inconsistency!

BEGIN

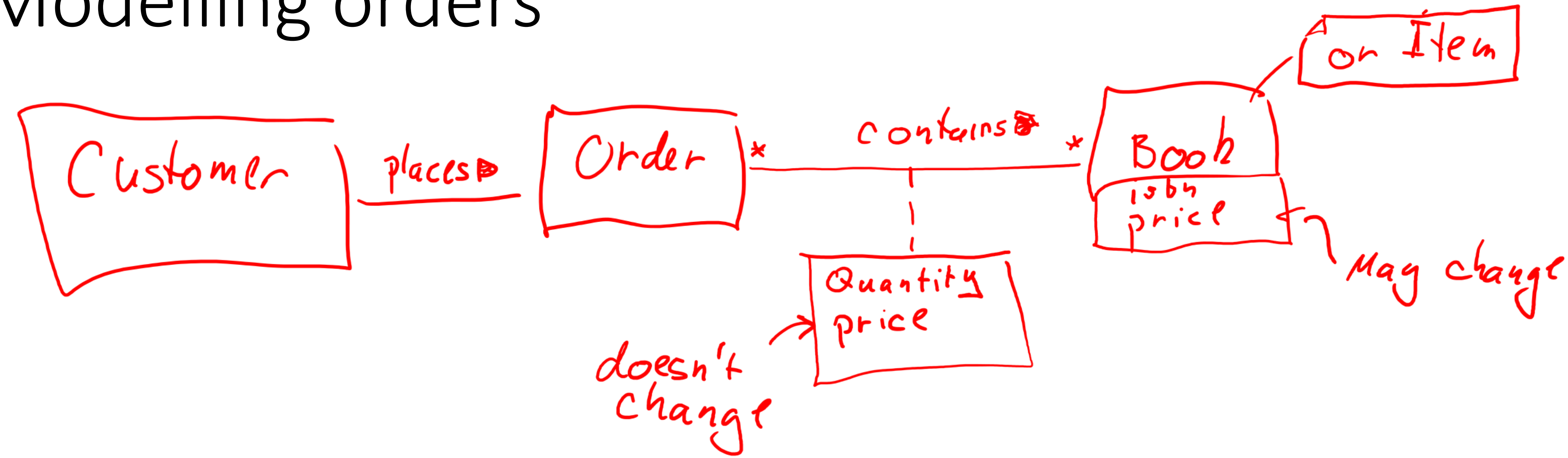
INSERT ...

UPDATE ...

COMMIT ← both operations at the "same time"

<u>Also true for MongoDB</u>

Modelling orders



Modelling Principles

- “The optimal grouping of objects into collections is determined by the workload.” - *Jay Runkel, Distinguished Solutions Architect, MongoDB*
- Nothing is wrong, but everything has consequences
- The choice of embedding is determined by queries and how frequent they are
- Data that's retrieved together should be stored together
- Data that's not retrieved together should not be stored together
- Redundancy is okay, but only when needed

Modelling options

- 1-1
    1. Embed one object in the other
- 1-*
    1. Refer to the parent object id from the child
    2. Refer to the child objects as an array of object ids in the parent
    3. Embed the parent object in the child
    4. Embed an array of the child objects in the parent
- *-*
    1. Refer to the related objects as an array of object ids
    2. Embed an array of the related objects

Embedding Pros and Cons

- Pros:
  - Avoiding joins — $lookup
  - Time performance
  - Easier queries
- Cons
  - Redundancy
  - Worse space performance (disk space)
  - 16MB limit

MongoDB Design Patterns

Pattern	Link
Approximation	https://www.mongodb.com/blog/post/building-with-patterns-the-approximation-pattern
* Attribute	https://www.mongodb.com/blog/post/building-with-patterns-the-attribute-pattern
Bucket	https://www.mongodb.com/blog/post/building-with-patterns-the-bucket-pattern
* Computed	https://www.mongodb.com/blog/post/building-with-patterns-the-computed-pattern
Document Versioning	https://www.mongodb.com/blog/post/building-with-patterns-the-document-versioning-pattern
* Extended Reference	https://www.mongodb.com/blog/post/building-with-patterns-the-extended-reference-pattern
Outlier	https://www.mongodb.com/blog/post/building-with-patterns-the-outlier-pattern
Pre-allocation	https://www.mongodb.com/blog/post/building-with-patterns-the-preallocation-pattern
* Polymorphic	https://www.mongodb.com/blog/post/building-with-patterns-the-polymorphic-pattern
Schema Versioning	https://www.mongodb.com/blog/post/building-with-patterns-the-schema-versioning-pattern
* Subset	https://www.mongodb.com/blog/post/building-with-patterns-the-subset-pattern
* Tree	https://www.mongodb.com/blog/post/building-with-patterns-the-tree-pattern

Attribute Pattern

weak entity

Instead of this

```
{ name: "John Doe",
  "mobile phone": "2885 6543",
  "work phone": "8755 1234",
  "land line": "7525 9137"
}
```

Do this

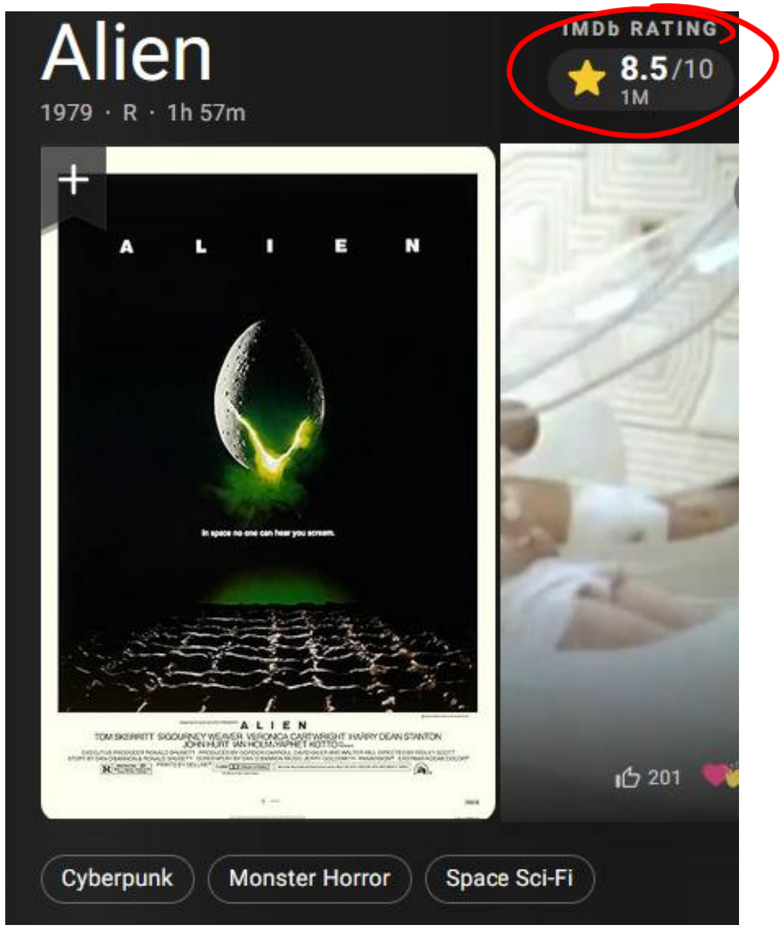
```
{ name: "John Doe",
  phones: [
    { type: "mobile",
      number: "2885 6543"},
    { type: "work",
      number: "8755 1234"},
    { type: "land line",
      number: "7525 9137"},
  ]
}
```

Computed Pattern

Instead of averaging 1M reviews on every read …

… update with every new review

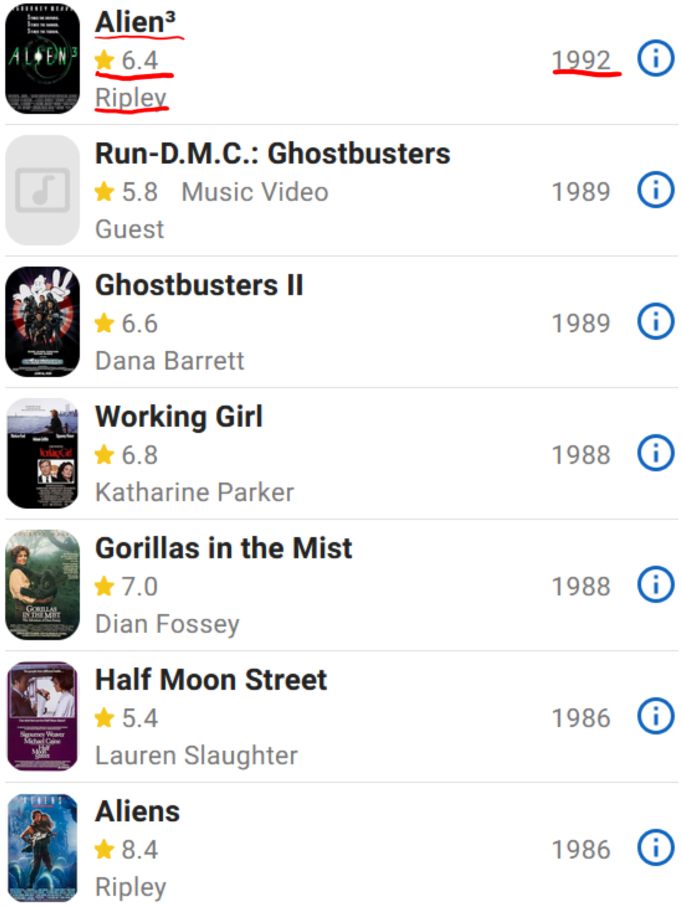
```
{ title: "Alien",
  year: 1979,
  "average score": 8.5,
  "#reviews": 1020399,
  reviews: [{score: 7}, …]
}
```



Extended Reference

```
[
  {
    role: ObjectId("AB53-…"),
    title: "Alien³",
    score: 6.4,
    character: "Ripley",
    year: 1992
  },
  …
]
```

Part of the object



Alien³
★ 6.4
Ripley
1992

Run-D.M.C.: Ghostbusters
★ 5.8 Music Video
Guest
1989

Ghostbusters II
★ 6.6
Dana Barrett
1989

Working Girl
★ 6.8
Katharine Parker
1988

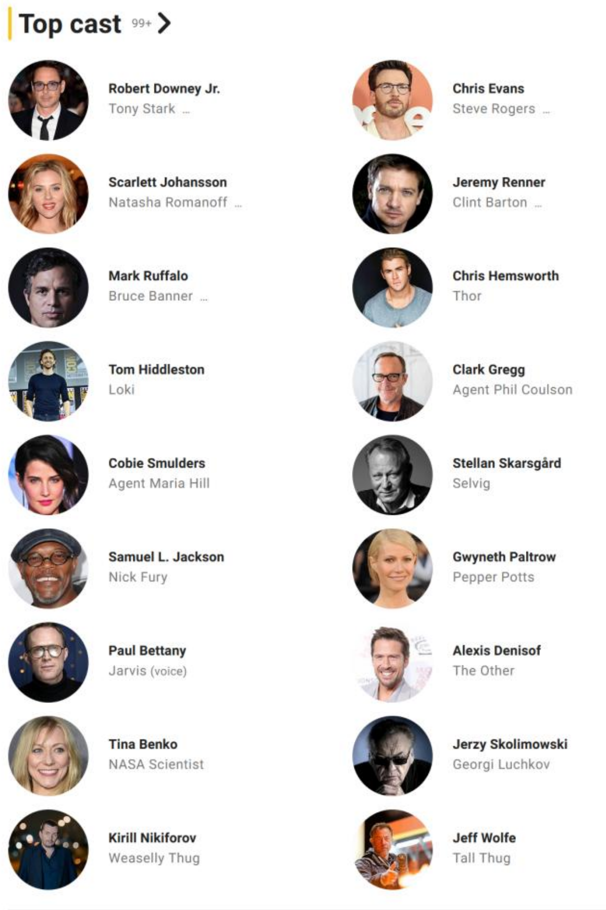
Gorillas in the Mist
★ 7.0
Dian Fossey
1988

Half Moon Street
★ 5.4
Lauren Slaughter
1986

Aliens
★ 8.4
Ripley
1986

Subset

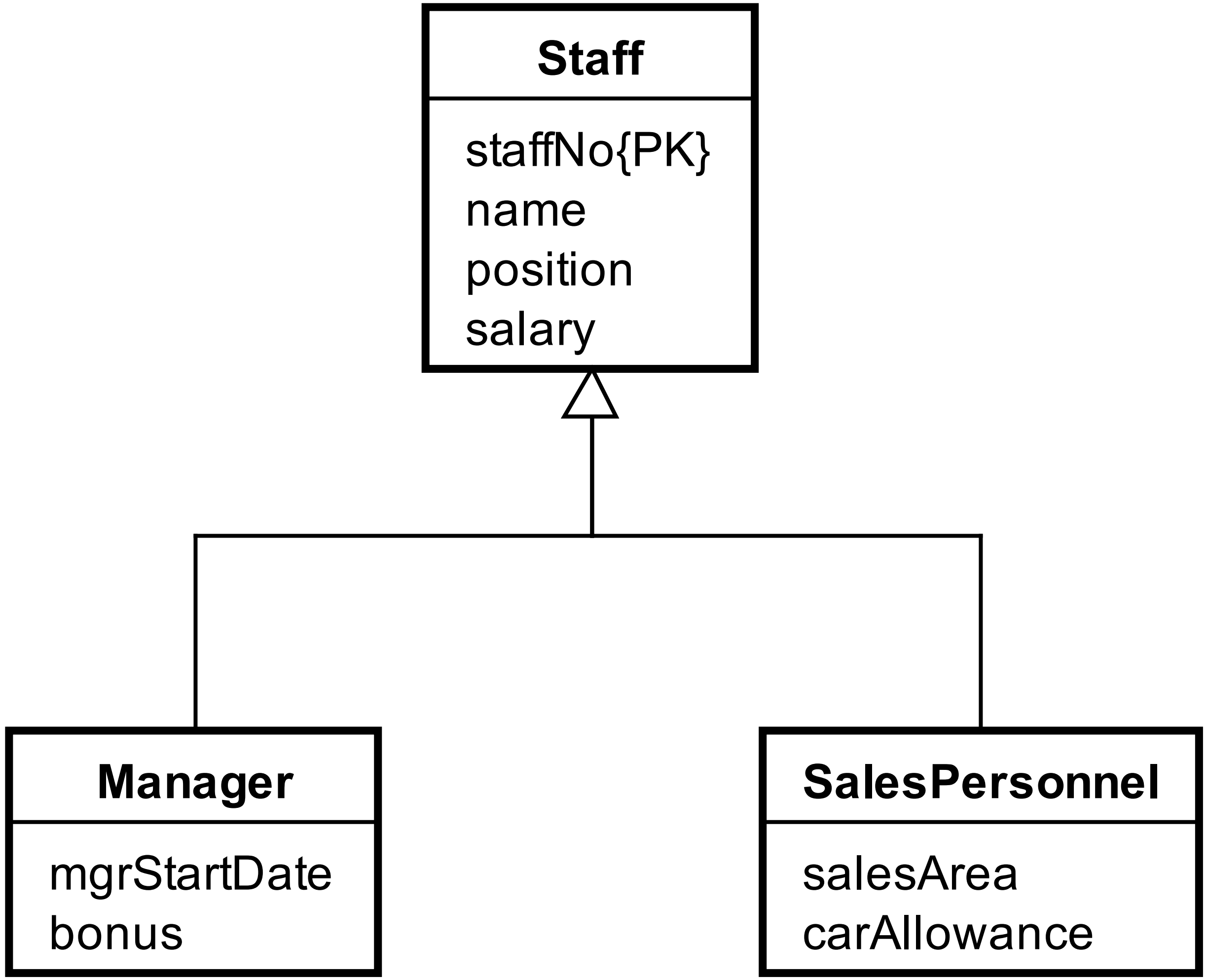
- An array of only the objects you want to show right now
- The full set of objects are modelled according to the normal mapping options



Top cast 99+ >

Robert Downey Jr. Tony Stark …	Chris Evans Steve Rogers …
Scarlett Johansson Natasha Romanoff …	Jeremy Renner Clint Barton …
Mark Ruffalo Bruce Banner …	Chris Hemsworth Thor
Tom Hiddleston Loki	Clark Gregg Agent Phil Coulson
Cobie Smulders Agent Maria Hill	Stellan Skarsgård Selvig
Samuel L. Jackson Nick Fury	Gwyneth Paltrow Pepper Potts
Paul Bettany Jarvis (voice)	Alexis Denisof The Other
Tina Benko NASA Scientist	Jerzy Skolimowski Georgi Luchkov
Kirill Nikiforov Weaselly Thug	Jeff Wolfe Tall Thug

Specializations ("inheritance")



Staff

staffNo{PK}
name
position
salary

Manager

mgrStartDate
bonus

SalesPersonnel

salesArea
carAllowance

Modelling specializations

- In schemaless, no problem:
  - Manager: {staffNo, name, position, salary, mgrStartDate, bonus}
  - SalesPersonel: {staffNo, name, position, salary, salesArea, carAllowance}
- These can live side by side in the same Staff collection
- The problem is with creating a schema:
  - We can't forbid strange mixes:
    {staffNo, name, position, salary, salesArea, mgrStartDate}

Polymorphic Pattern

```
{
  staffNo,
  name,
  position,     type: "manager"
  salary,
  manager: {    Not required
    mgrStartDate, - required
    bonus         - required
  }
}
```

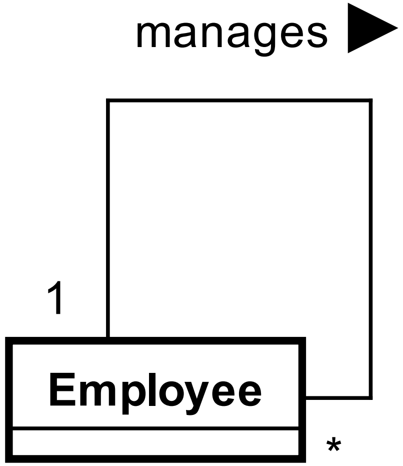
```
{
  staffNo,
  name,
  position,
  salary,
  salesPersonel: {
    salesArea,
    carAllowance
  }
}
```

Recursive relationships

- In relational mapping, dealt with like normal relationships
  - Typically, a reference to the parent
- We can do the same in MongoDB:

```
Employee: {
  _id,
  manager_id,
  …
}
```

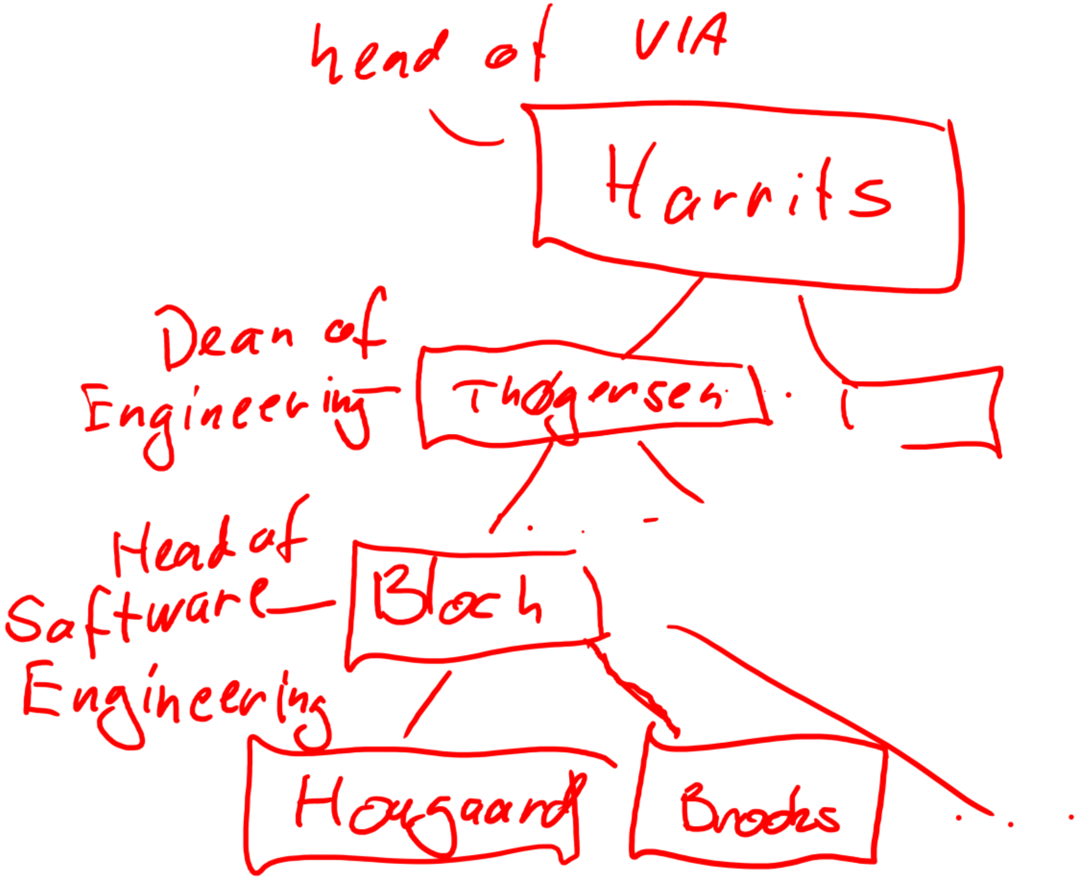
- There is another option: Tree Pattern



Tree Pattern

```
{
  name: "Ole Hougaard",
  title: "Associate Prof.",
  managers: [
    "Helle Bloch",
    "Lotte Thøgersen",
    "Gitte Sommer Harrits"
  ]
}
```

ids



MongoDB JSON Schema

- An example of a *validator*
- A validator validates new (inserted) data
    - Either an error or a warning
- A validator does *not* validate existing data
- May or may not validate updated data

JSON Schema example

```
db.createCollection("performers", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name"],
      properties: {
        name: {bsonType: "string"},
        url: {bsonType: "string"}
      }
    }
  }
})
```

Always on top-level

required, string

– string, if present

Allows for any number of other properties

{name:"", url:"", catchphrase:""} ✓

{name:"", url:0, catchphrase:""} X

error

Indexes

Automatic on _id

- Single Field Index
  - Used for filtering +sorting on a single field
  - Important for fields used in a $lookup (Join)
- Compound Field Index
  - Used for filtering on multiple fields
  - Used to create a "covering index"
- Multikey Index
  - Used to index embedded arrays
- Text Indexes
  - Used to speed up text search

Covering

- A query is *covered* by a compound index if all properties from the query are in the index
- Example:

```
db.cars.find(
   { manufacturer: 'Ford',
     cost: { $gt: 15000 }
   }).sort( { model: 1 } )
```

is covered by `{ manufacturer: 1, model: 1, cost: 1 }`

Match these

sort order of the index (usu: 1)

Other compound index optimizations

- ESR rule
  - When designing a compound index for a query, use this order
    - Equality - first the fields used to test for equality
    - Sort - then the fields used to sort the output (in the desired sort order)
    - Range - then the fields tested with $gt, $lt, etc.
- Prefix of compound indexes
  - Like with relational databases, a prefix of a compound index is also used to speed up operations
  - `{manufacturer: 1, model: 1, cost: 1}` covers queries using only manufacturer and model (but not manufacturer and cost)
  - Use this to save on indexes

Embedding

```
[
   { "name": "Alice",
      "gpa": 3.6,
      "location": {
        city: "Sacramento",
        state: "California" }
   },
   { "name": "Bob",
      "gpa": 3.2,
      "location": {
        city: "Albany",
        state: "New York" }
   }
]
```

```
db.students.createIndex({
  location:1
})
```

- works w/ find({location: {city:"", state:""}})

```
db.students.createIndex({
  "location.city": 1
})
```

- works w/ find ({"location.city": "Albany"}) ←

```
db.students.createIndex({
  "location.$**": 1
})
```

or find({"location.state": "New York"})