MongoDB 3.2 – Document Validation

Andrew Morgan
@andrewmorgan | andrew.morgan@mongodb.com

29th October 2015

DISCLAIMER: MongoDB's product plans are for informational purposes only. MongoDB's plans may change and you should not rely on them for delivery of a specific feature at a specific time.

Agenda

Value of flexible schemas

Downside of flexible schemas

What 3.2 adds

What 3.2 doesn’t add

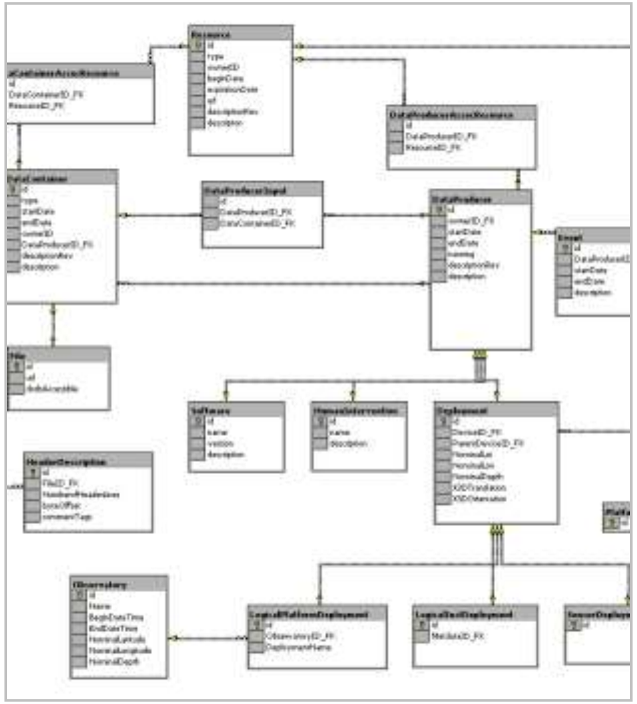
Options

Production lifecycle to add Document Validation

Walkthrough

Power of flexible schemas

RDBMS



MongoDB

```
{
    _id: ObjectId("4c4ba5e5e8aabf3"),
    employee_name: {First: "Billy",
                    Last: "Fish"},
    department: "Engineering",
    title: "Aquarium design",
    pay_band: "C",
    benefits: [
        {  type:  "Health",
           plan: "PPO Plus" },
        {  type:   "Dental",
           plan: "Standard" }
        ]
}
```

Power of flexible schemas

```
{
  _id: ObjectId("4c4ba5e5e8aabf3"),
  employee_name: {
    First: "Billy",
    Last:  "Fish"},
  department: "Engineering",
  title: "Aquarium design",
  pay_band: "C",
  benefits: [
    {type: "Health",
     plan: "PPO Plus" },
    {type: "Dental",
     plan: "Standard" }
    ]
}
```

- Relational
  - Up-front schema definition phase
  - Adding new column takes time to develop & *lots* of time to roll out in production
    - Existing rows must be reformatted
- MongoDB:
  - Start hacking your app right away
  - Want to store new type of information?
    - Just start adding it
    - If it doesn't apply to all instances – just leave it out

mongoDB

Power of flexible schemas

```
{
  _id: ObjectId("4c4ba5e5e8aabf3"),
  employee_name: {
    First: "Billy",
    Last:  "Fish"},
  department: "Engineering",
  title: "Aquarium design",
  pay_band: "C",
  benefits: [
    {type: "Health",
     plan: "PPO Plus" },
    {type: "Dental",
     plan: "Standard" }
    ]
}
```

- Relational
  - Up-front schema definition phase
  - Adding new column takes time to develop & *lots* of time to roll out in production
    - Existing rows must be reformatted
- MongoDB:
  - Start hacking your app right away
  - Want to store new type of information?
    - Just start adding it
    - If it doesn’t apply to all instances – just leave it out

mongoDB

Why validate documents?

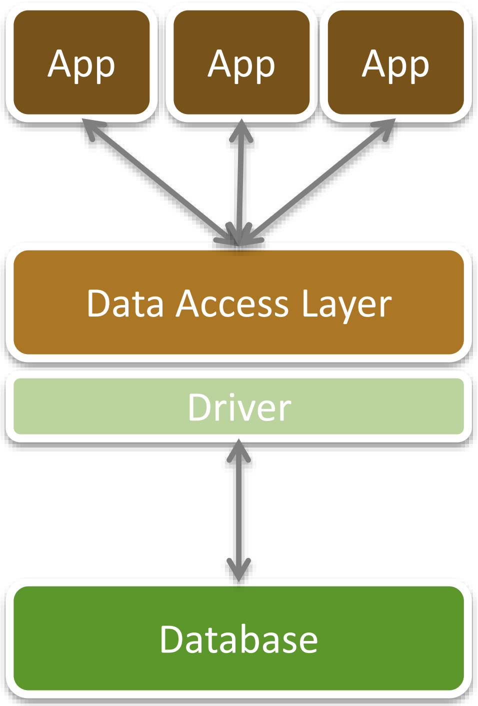
- Many people writing to the database
  - Many developers
  - Many teams
  - Many companies
  - Many development languages
- Multiple applications want to exploit the same data, need to agree on what’s there
- Usually a core subset of keys you always want to be there
- For any key may care about:
  - Existence
  - Type
  - Format
  - Value
  - Existence in combination with other keys (e.g. need a phone number **or** an email address)

Why validate documents?

- Good to have a ‘contract’ for what’s in a collection
  - When reading from the “subscriber” collection, I know that every document will include a subscription plan name:

```
db.subscriptions.find(
    name: "Billy Fish",
    $and:[{plan:{$exists: true}},
          {plan:{$type: 2}}]})
```

- < MongoDB 3.2, this is an application responsibility
  - 3rd party tools like Mongoose can help
- Best implemented as a layer between the application and driver (Data Access Layer)



Get the database to do the work!

mongoDB

Document Validation - MongoDB 3.2

- Configure Document **Validation within the database**
- Use familiar **MongoDB Query Language**
- Automatically tests each insert/update; delivers **warning or error** if a rule is broken
- You choose **what keys to validate and how**

```
db.runCommand({
   collMod: "contacts",
   validator: {
      $and: [
        {year_of_birth: {$lte: 1994}},
        {$or: [
                 {phone: { $type: ”string"}},
                 {email: { $type: ”string"}}
             ]}]
    }})
```

Document Validation - MongoDB 3.2

```
db.getCollectionInfos({name:"contacts"})
[
   {
     "name": "contacts",
     "options": {
       "validator": {
         "$and": [
           {"year_of_birth": {
               "$lte": 1994}},
           {"$or": [
               {"phone": {"$type": "string"}},
               {"email": {"$type": "string"}}
             ]}
         ]},
       "validationLevel": "strict",
       "validationAction": "error”
     }
   }
 ]
```

Document Validation - MongoDB 3.2

```
db.contacts.insert(
   name: "Fred",
   email: "fred@clusterdb.com",
   year_of_birth: 2012
})

Document failed validation
WriteResult({
  "nInserted": 0,
  "writeError": {
    "code": 121,
    "errmsg": "Document failed validation”}})
```

Document Validation - MongoDB 3.2

```
db.runCommand({collMod: "bleh",
               validator: {
                  rogue: {$exists:false}
                  }
                }
              });
```

Document Validation - MongoDB 3.2

- Can check most things that work with a `find` expression
  - Existence
  - Non-existence
  - Data type of values
  - <, <=, >, >=, ==, !=
  - AND, OR
  - Regular expressions
  - Some geospatial operators (e.g. `$geoWithin` & `$geoIntersects`)
  - …

MongoDB 3.2 Limitations

- Generic error message
  - Application needs to figure out what part of the constraints failed
- Cannot compare 1 key with another
  - Either within the same document or between documents
- Some operations not supported:
  - `$text`, `$geoNear`, `$near`, `$nearSphere`, `$where`
- Applications responsibility to bring legacy data into compliance with new rules
  - No audit or tools

What validations remain in the app

- User interface
  - Don’t have the database be the first place to detect that an email is poorly formatted
- Any validations that involve comparisons with
  - Other data in the same document
  - Data from other documents
  - External information (e.g. time of day)
- Semantic checks that are designed to fail frequently
  - e.g. user is in wrong country to use this service
  - Database should typically be testing for coding errors rather than implementing your business logic
- Determining why the database rejected a document in order to provide a meaningful error to the user

Where MongoDB Validation excels (vs. RDBMS)

- Simple
  - Use familiar search expressions (MQL)
  - No need for stored procedures
- Flexible
  - Only enforced on mandatory parts of the schema
  - Can start adding new data at any point and then add validation later if needed
- Practical to deploy
  - Simple to role out new rules across thousands of production servers
- Light weight
  - Negligible impact to performance

Cleaning up legacy data

- Validator does not check if existing documents in the collection meet the new validation rules
- User/app can execute a query to identify & update any document which don't meet the new rules
  - Use `$nor` on the full expression
- Be cautious about system impacts:
  - Could push working data set out of memory
  - Extra load if many documents need to be updated
  - Execute on secondary

```
secondary> db.runCommand({collMod: "bleh",
               validator: {
                  a: {$lt:4}
               }
             });

secondary> db.bleh.find({
             a:{$not:{$lt:4}}}).count()

secondary> db.bleh.update(
             {a:{$not:{$lt:4}}},
             {$set:{a:3}},
             {multi:true})
```

Controlling validation

		validationLevel		
		off	moderate	strict
validationAction	warn	No checks	Warn on validation failure for inserts & updates to existing valid documents. Updates to existing invalid docs OK.	Warn on any validation failure for any insert or update.
	error	No checks	Reject invalid inserts & updates to existing valid documents. Updates to existing invalid docs OK.	Reject any violation of validation rules for any insert or update. **DEFAULT**

Controlling validation

- Set behavior:

```
    db.bleh.runCommand("collMod",
                       {validationLevel: "moderate",
                        validationAction: "warn"})
```

  - Note that the warnings are written to the log

Lifecycle

Hacking (Day one)

- No document validation
- Release quick & often

Analyze De facto Schema

- MongoDB Compass

Add document validation rules

- Query & fix existing docs
- Log any new documents that break rules:

```
{validationLevel:
"moderate",
validationAction:
"warn"}
```

Fix application

- Follow all rules
- When no new problems being logged, enter strict mode:

```
{validationLevel:
”strict",
validationAction:
”error}
```

Application uses new data (Application evolves/additional app)

- If not mandated, stop here

?

Analyze De-facto Schema

- MongoDB Compass

Add document validation rules

- Query & fix existing docs
- Log any new documents that break rules:

```
{validationLevel:
"moderate",
validationAction:
"warn"}
```

Fix application

- Follow all rules
- When no new problems being logged, enter strict mode:

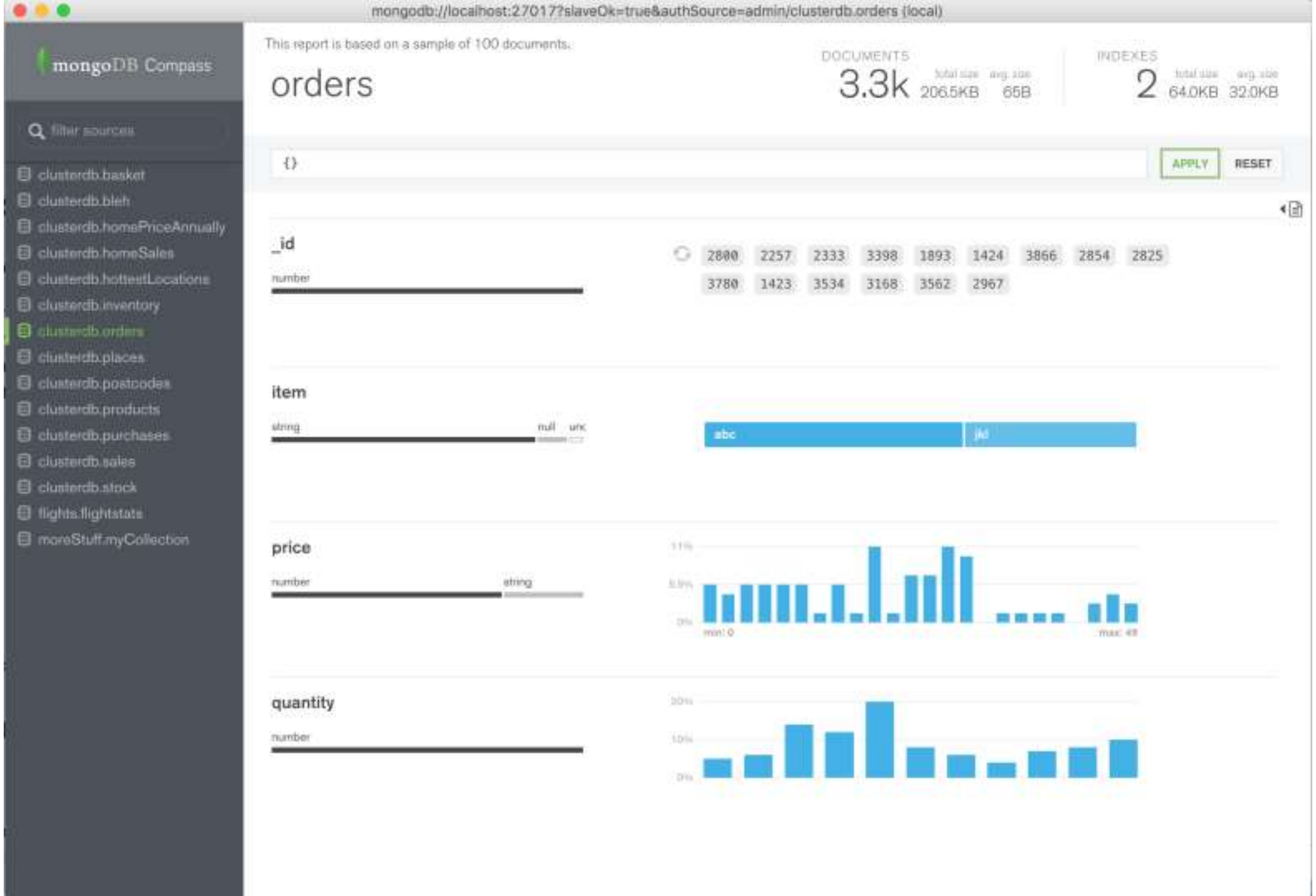
```
{validationLevel:
”strict",
validationAction:
”error}
```

Versioning of Validations (optional)

```
db.runCommand({
   collMod: "contacts",
   validator:
     {$or: [{version: {"$exists": false}},
            {version: 1,
             $and: [{Name: {"$exists": true}}]
            },
            {version: 2,
             $and: [{Name: {"$type": ”string"}}]
            }
          ]
      }
})
```

- Application can lazily update documents with an older version or with no version set at all

Step through selecting and deploying a simple Document Validation Rule



mongodb://localhost:27017?slaveOk=true&authSource=admin/clusterdb.orders (local)

mongoDB Compass

filter sources

- clusterdb.basket
- clusterdb.bleh
- clusterdb.homePriceAnnually
- clusterdb.homeSales
- clusterdb.hottestLocations
- clusterdb.inventory
- clusterdb.orders
- clusterdb.places
- clusterdb.postcodes
- clusterdb.products
- clusterdb.purchases
- clusterdb.sales
- clusterdb.stock
- flights.flightstats
- moreStuff.myCollection

This report is based on a sample of 100 documents.

orders

DOCUMENTS 3.3k total size 206.5KB avg. size 65B

INDEXES 2 total size 64.0KB avg. size 32.0KB

```
{}
```

APPLY RESET

_id

number

2800 2257 2333 3398 1893 1424 3866 2854 2825 3780 1423 3534 3168 3562 2967

item

string null unk

abc jkl

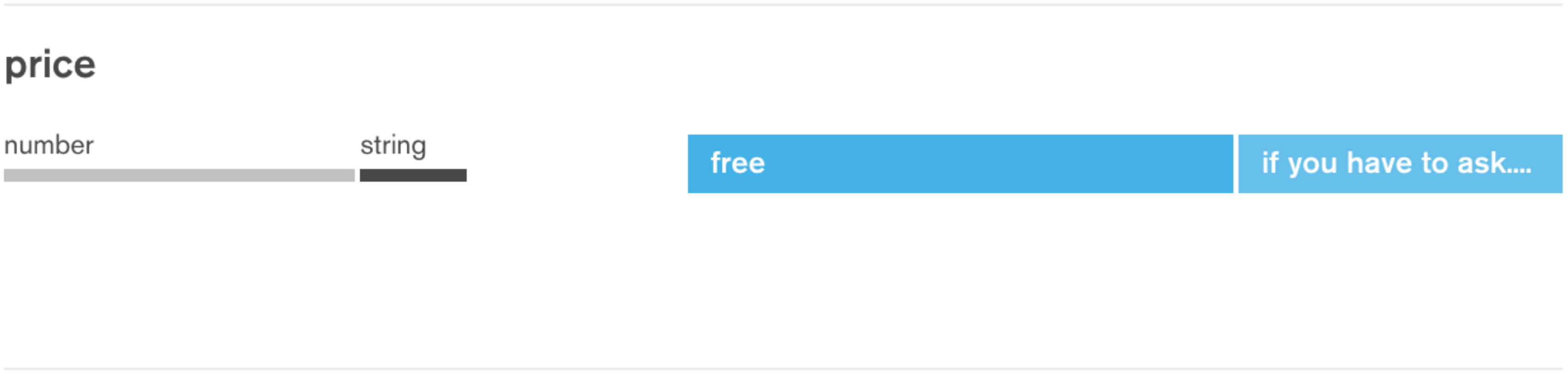
price

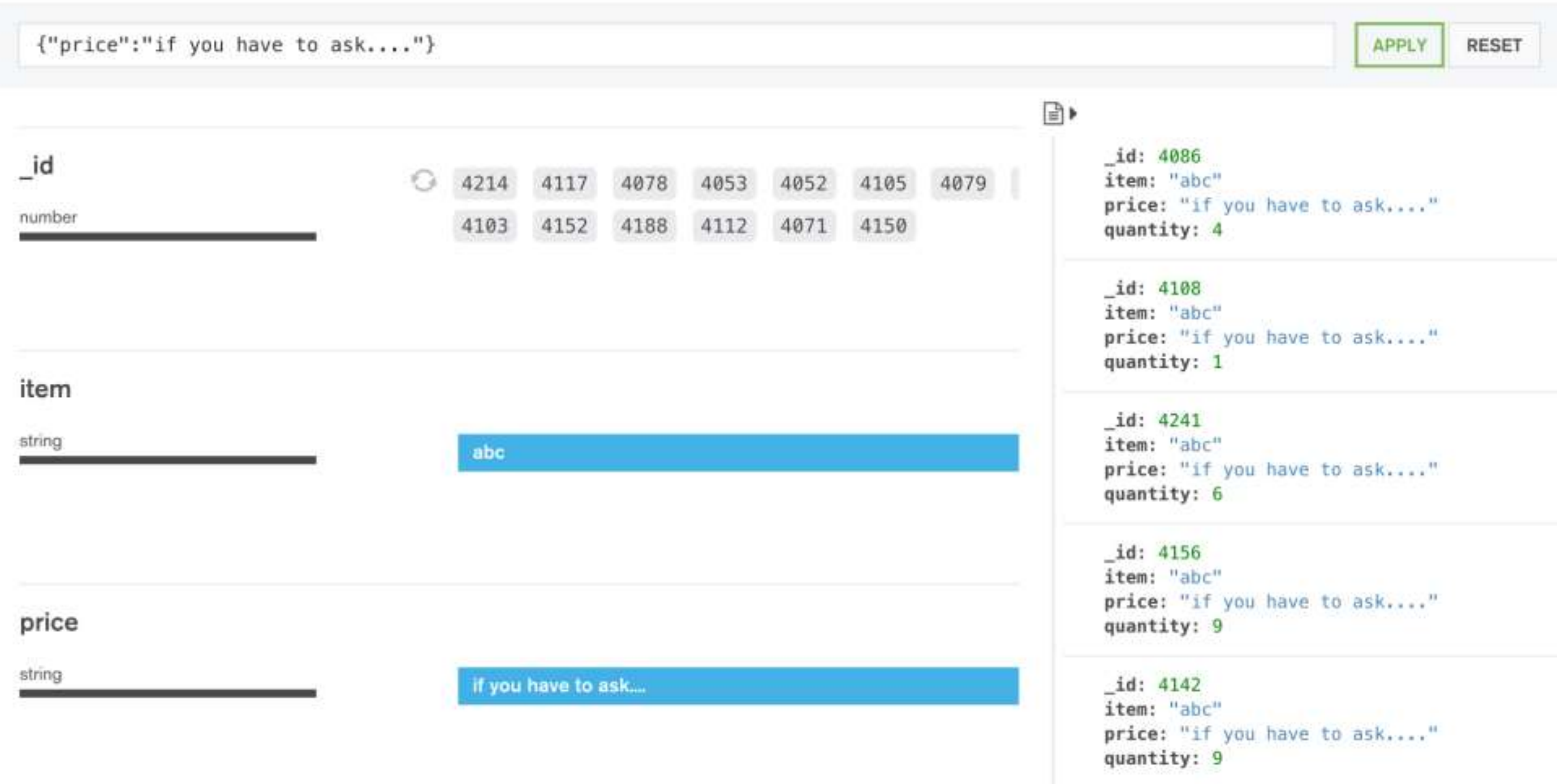
number string

quantity

number

Drill down into anomalies





```
{"price":"if you have to ask...."}
```

_id
number

4214 4117 4078 4053 4052 4105 4079
4103 4152 4188 4112 4071 4150

item
string

abc

price
string

if you have to ask....

```
_id: 4086
item: "abc"
price: "if you have to ask...."
quantity: 4

_id: 4108
item: "abc"
price: "if you have to ask...."
quantity: 1

_id: 4241
item: "abc"
price: "if you have to ask...."
quantity: 6

_id: 4156
item: "abc"
price: "if you have to ask...."
quantity: 9

_id: 4142
item: "abc"
price: "if you have to ask...."
quantity: 9
```

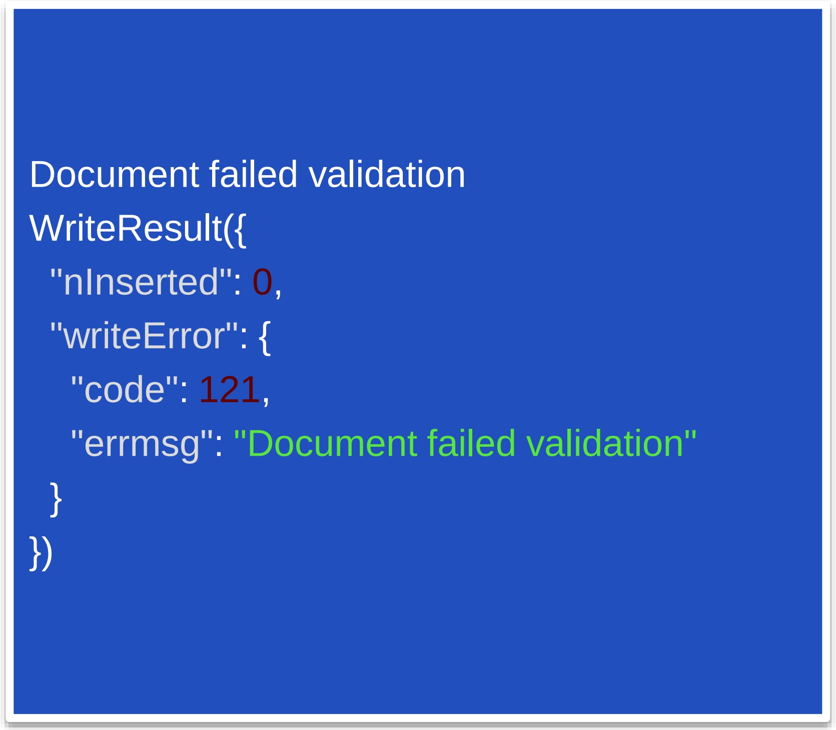
1. Prevent New Malformed Documents

```
> db.orders.runCommand("collMod",
                  {validationLevel: "moderate",
                   validationAction: "error"});

> db.runCommand({collMod: "orders”,
                validator: {
                     $or: [
                        {price: {$type: 1}},
                        {price: {$type: 16}},
                        {price: {$type: 18}}
                     ]}});

> db.getCollectionInfos({name:"orders"})
```

```
{
  "name": "orders",
  "options": {
    "validator": {
      "$or": [
        { "price": { "$type": 1  }},
        { "price": { "$type": 16 }},
        { "price": { "$type": 18 }}
      ]
    },
    "validationLevel": "moderate",
    "validationAction": "error"
  }
}
```

2. Prevent New Malformed Documents

```
> db.orders.insert({
    "_id": 6666,
    "item": "jkl",
    "price": "rogue",
    "quantity": 1 });
```



```
Document failed validation
WriteResult({
  "nInserted": 0,
  "writeError": {
    "code": 121,
    "errmsg": "Document failed validation"
  }
})
```

mongoDB

3. Clean-Up Legacy Documents

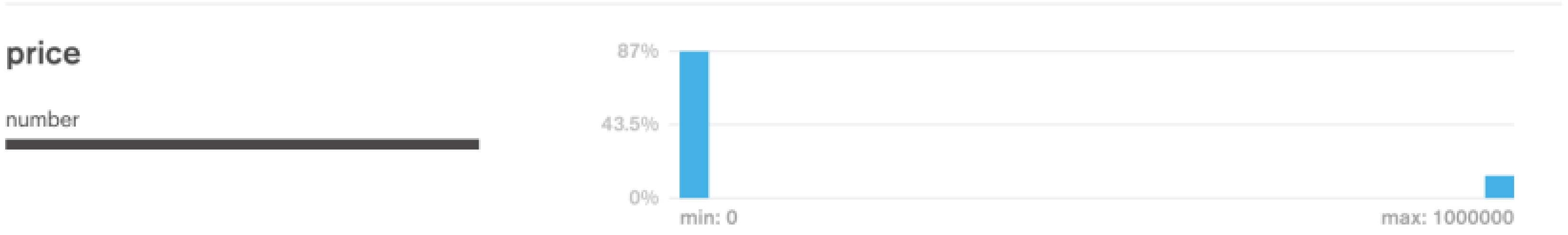
```
> db.orders.findOne(
 {price:
   {$type: "string"}}
);

> db.orders.update(
 {_id: 3500},
 {$set:{quantity: 12}}
);
```

```
{
  "_id": 3500,
  "item": "abc",
  "price": "free",
  "quantity": 8
}

Updated 1 existing record(s) in 6ms
WriteResult({
  "nMatched": 1,
  "nUpserted": 0,
  "nModified": 1
})
```

3. Clean-Up Legacy Documents

```
> db.orders.update(
    {price:"free"},
    {$set: {price: 0}},
    {multi: true});

> db.orders.update(
    {price:"if you have to ask...."},
    {$set: {price: 1000000}},
    {multi: true});
```

4. Confirm Results



```
> db.orders.find(
  {$nor: [
          {price: {$type: 1}},
          {price: {$type: 16}},
          {price: {$type: 18}}
         ]})
```

Fetched 0 record(s) in **5ms**

mongoDB

Next Steps

- Document Validation - Adding Just the Right Amount of Control Over Your Documents
  - https://www.mongodb.com/blog/post/document-validation-part-1-adding-just-the-right-amount-of-control-over-your-documents
- “Document Validation and What Dynamic Schema Means” – Eliot Horowitz
  - http://www.eliothorowitz.com/blog/2015/09/11/document-validation-and-what-dynamic-schema-means/
- “Bulletproof Data Management” – MongoDB World 2015
  - https://www.mongodb.com/presentations/data-management-3-bulletproof-data-management
- Documentation
  - http://docs.mongodb.org/manual/release-notes/3.1-dev-series/#document-validation
- Not yet ready for production but download and try MongoDB 3.2 RC
  - https://www.mongodb.org/downloads#development
- Feedback
  - https://www.mongodb.com/blog/post/announcing-the-mongodb-3-2-bug-hunt
  - https://jira.mongodb.org/

DISCLAIMER: MongoDB's product plans are for informational purposes only. MongoDB's plans may change and you should not rely on them for delivery of a specific feature at a specific time.

mongoDB