

CSCI235 Database Systems

Validation with JSON Schema

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Validations

Outline

Overview

JSON Schema validations

\$jsonSchema operator

Validation of any BSON documents

Validation of flat BSON documents

Validation of nested BSON documents

Overview

Semistructured and **schemaless** properties of **JSON/BSON data model** allow for very flexible manipulation of database structures

In the same moment very flexible and uncontrolled manipulations of database structures open the possibilities for pretty easy **corruption** of database structures and database contents, for example, due to the random mistakes

Practice shows that certain level of verification of database consistency constraints is always needed

MongoDB provides the capability to **validate documents** during **updates** and **insertions** (and **not deletions**)

Validation rules are specified within **createCollection()** method using the **validator** option

It is possible use **collMod** command with the **validator** option to turn on/off **validation rules**

Validations

Outline

[Overview](#)

[JSON Schema validations](#)

[\\$jsonSchema operator](#)

[Validation of any BSON documents](#)

[Validation of flat BSON documents](#)

[Validation of nested BSON documents](#)

JSON Schema validation

JSON Schema is a **BSON document** that defines the structure of JSON data for validation, documentation, and interaction control

JSON Schema validation of a **JSON document** is performed through verification of consistency of the structures and contents of the document with **JSON Schema**

JSON Schema is based on the concepts "borrowed" from **XML Schema**

A method `createCollection()` creates an empty collection and associates `validator` with a collection

If validation with **JSON Schema** is required then `validator` is associated with a **JSON schema** that determines the structures and the contents of the documents in the collection

It is possible use `collMod` method with the `validator` option to turn on/off **validation rules**

Validations

Outline

[Overview](#)

[JSON Schema validations](#)

[\\$jsonSchema operator](#)

[Validation of any BSON documents](#)

[Validation of flat BSON documents](#)

[Validation of nested BSON documents](#)

\$jsonSchema operator

In MongoDB an operator `$jsonSchema` is associated with a **JSON Schema** to be used for validation of **BSON documents**

`$jsonSchema` operator can also be used to query with `find` command or with `$match` aggregation stage

The following application of `$jsonSchema` operator validates any **BSON document** in a collection `department`

```
db.createCollection("department",
  {"validator":{"$jsonSchema":{"bsonType":"object"}} }
);
```

`$jsonSchema` operator

For example, the following **BSON document** validates well

```
{"name":"School of Astronomy",
  "address":{"street":"Franz Josef Str",
    "bldg":4},
  "courses":[{"code":"SOA101",
    "title":"Astronomy for Kids",
    "credits":3} ] }
```

A sample BSON document that validates with `$jsonSchema` validator above

Validations

Outline

[Overview](#)

[JSON Schema validations](#)

[\\$jsonSchema operator](#)

[Validation of any BSON documents](#)

[Validation of flat BSON documents](#)

[Validation of nested BSON documents](#)

Validation of any BSON documents

The following **JSON Schema** validates well any document in a collection **anyDocument**

```
db.createCollection("anyDocument",  
    {"validator":{"$jsonSchema":{"bsonType":"object"}} }  
    );
```

JSON Schema validator that validates any document

Note the application of **bsonType** in MongoDB validation instead of **type** in standard **JSON schema**

Validations

Outline

[Overview](#)

[JSON Schema validations](#)

[\\$jsonSchema operator](#)

[Validation of any BSON documents](#)

[Validation of flat BSON documents](#)

[Validation of nested BSON documents](#)

Validation of flat BSON documents

The following BSON document in a collection `tiny`

```
db.tiny.insert({"name":"Harry Potter"});
```

A tiny BSON document

fails validation against the following JSON schema

```
db.createCollection( "tiny",  
                    {"validator":{"$jsonSchema":  
{"bsonType":"object",  
 "properties":{"name":{"bsonType":"string"} },  
 "required":["name"],  
 "additionalProperties":false  
} } } );
```

\$jsonSchema validator

A validation fails because a key `additionalProperties` is associated with `false` and a key `"_id"` included in every BSON document is not on `required` list

Validation of flat BSON documents

The following BSON document in a collection `tiny`

```
db.tiny.insert({"_id":"HP666","name":"Harry Potter"});
```

A tiny BSON document

passes validation against the following JSON schema

```
db.createCollection( "tiny",  
                    {"validator":{"$jsonSchema":  
{"bsonType":"object",  
  "properties":{"_id":{"bsonType":"string"},  
                "name":{"bsonType":"string"} },  
  "required":["_id","name"],  
  "additionalProperties":false  
} } } );
```

\$jsonSchema validator

A validation passes because a key `additionalProperties` is associated with `false` and a key `"_id"` included in every BSON document is on `required` list

Validation of flat BSON documents

The following BSON documents in a collection `empty`

BSON documents in a collection empty

```
db.empty.insert({"_id":"HP666"});  
db.empty.insert({"_id":"HP667","name":"Harry Potter"});  
db.empty.insert({"_id":"HP668","name":"Harry Potter","occupation":"wizard"});
```

pass validation against the following JSON schema

\$jsonSchema validator

```
db.createCollection( "empty",  
                    {"validator":{"$jsonSchema":  
{"bsonType":"object",  
  "properties":{"_id":{"bsonType":"string"} },  
  "required":["_id"],  
  "additionalProperties":true  
} } } );
```

A validation passes because a key `additionalProperties` is associated with `true` and a key `"_id"` included in every BSON document is on `required` list

Validation of flat BSON documents

The following **BSON document** in a collection **flat**

```
db.flat.insert({"_id":"HP666","name":"Harry Potter","age":NumberInt("100")});
```

A tiny BSON document

passes validation against the following **JSON schema**

```
db.createCollection( "flat",
    {"validator":{"$jsonSchema":
{"bsonType":"object",
  "properties":{"name":{"bsonType":"string",
    "maxLength":100},
    "age":{"bsonType":"int",
    "maximum":200,
    "exclusiveMaximum":true} },
  "required":["name","age","_id"],
  "additionalProperties":false
} } } );
```

\$jsonSchema validator

A default type of a value associated with a key **"_id"** is **string**

Validation of nested BSON documents

The following **BSON documents** in a collection **department**

```
db.department.insert({"_id":"Finance","budget":123});  
db.department.insert({"_id":"Sales","budget":123,"fee":456});
```

BSON document

pass validation against the following **JSON schema**

```
db.createCollection("department",  
  { "validator":{$jsonSchema:  
    {"bsonType":"object",  
      "properties":{"_id":{"bsonType":"string"},  
                    "budget":{"bsonType":"double",  
                              "description":"Budget of department"},  
                    "fee":{"bsonType":"double",  
                           "description":"Fee paid"}}  
    },  
    "required":["_id","budget"],  
    "additionalProperties":false  
  } } } );
```

\$jsonSchema validator

Validations

Outline

[Overview](#)

[JSON Schema validations](#)

[\\$jsonSchema operator](#)

[Validation of any BSON documents](#)

[Validation of flat BSON documents](#)

[Validation of nested BSON documents](#)

Validation of nested BSON documents

The following **BSON document** in a collection **nested**

```
db.nested.insert({ "_id": "HP667",  
  "name": "Harry Potter",  
  "address": { "city": "Dapto",  
    "code": NumberInt("2530") }  
});
```

A nested BSON document

passes validation against the following **JSON schema**

```
db.createCollection( "nested",  
  {"validator": {$jsonSchema:  
{"bsonType": "object",  
  "properties": { "_id": { "bsonType": "string",  
    "name": { "bsonType": "string",  
      "maxLength": 100,  
    "address": { "bsonType": "object",  
      "properties": { "city": { "bsonType": "string",  
        "minLength": 5, "maxLength": 30 },  
        "code": { "bsonType": "int",  
          "maximum": 9999,  
          "exclusiveMaximum": false } },  
      "required": [ "city", "code" ],  
      "additionalProperties": false } },  
    "required": [ "name", "address", "_id" ],  
    "additionalProperties": false } } } } );
```

\$jsonSchema validator

Validation of nested BSON documents

The following **BSON document** in a collection **nested**

```
db.nested.insert({"_id":"HP667",
                  "name":"Harry Potter",
                  "cars":["Ferrari","Mercedes","Hyundai"]
                });
```

A nested BSON document

passes validation against the following **JSON schema**

```
db.createCollection( "nested",
                    {"validator":{"$jsonSchema":
{"bsonType":"object",
 "properties":{"_id":{"bsonType":"string"},
               "name":{"bsonType":"string",
                        "maxLength":100},
               "cars":{"bsonType":"array",
                        "items":{"bsonType":"string"},
                        "uniqueItems":true }
               },
 "required":["name","cars","_id"],
 "additionalProperties":false
} } } );
```

\$jsonSchema validator

Validation of nested BSON documents

The following **BSON document** in a collection **month**

```
db.month.insert( { "_id": "Winter",  
                  "months": [ "June", "July", "August" ] } );
```

BSON document

passes validation against the following **JSON schema**

```
db.createCollection( "month",  
                    { "validator": { $jsonSchema:  
  { "bsonType": "object",  
    "properties": { "_id": { "bsonType": "string",  
                          "months": { "bsonType": "array",  
                                      "description": "Winter",  
                                      "items": { "bsonType": "string" } } },  
    "required": [ "_id", "months" ],  
    "additionalProperties": false  
  } } } );
```

\$jsonSchema validator

Validation of nested BSON documents

The following **BSON document** in a collection **addressBook**

```
db.month.insert( { "_id": "addressBook",  
  "address": [ { "city": "Sydney",  
    "street": "25 Victoria St.",  
    { "city": "Wollongong",  
      "street": "125 Northfields Ave." } ] } } );
```

BSON document

passes validation against the following **JSON schema**

```
db.createCollection("department",  
  {"validator": { $jsonSchema:  
    {"bsonType": "object",  
      "properties": { "_id": {"bsonType": "string"},  
        "address": {"bsonType": "array",  
          "description": "Cities and streets",  
          "items": {"bsonType": "object",  
            "properties": { "city": {"bsonType": "string"},  
              "street": {"bsonType": "string"} },  
            "required": ["city", "street"],  
            "additionalProperties": false  
          }  
        },  
      "required": ["_id", "address"],  
      "additionalProperties": false  
    } } } } );
```

\$jsonSchema validator

Validation of nested BSON documents

The following **BSON documents** in a collection **school**

```
db.school.insert(  
  { "name":"School of Astronomy",  
    "code":"S0A",  
    "total staff number":25,  
    "budget":10000,  
    "address":{"street":"Franz Josef Str",  
              "bldg":4,  
              "city":"Vienna",  
              "country":"Austria"},  
    "courses": [ {"code":"S0A101",  
                  "title":"Astronomy for Kids",  
                  "credits":3},  
                  {"code":"S0A201",  
                  "title":"Black Holes",  
                  "credits":6},  
                  {"code":"S0A301",  
                  "title":"Dark Matter",  
                  "credits":12}  
    ]  
  } );
```

BSON document

validates well against the following **JSON schema** ...

References

[JSON Schema](#)

[Understanding JSON Schema](#)

[MongoDB - JSON Schema validation](#)

[MongoDB - \\$jsonSchema operator](#)