$jsonSchema

On this page

Definition

`$jsonSchema`

New in version 3.6.

The `$jsonSchema` operator matches documents that satisfy the specified JSON Schema.

The `$jsonSchema` operator expression has the following syntax:

```
{ $jsonSchema: <JSON Schema object> }
```

Where the JSON Schema object is formatted according to draft 4 of the JSON Schema standard.

```
{ <keyword1>: <value1>, ... }
```

For example:

```
{
   $jsonSchema: {
      required: [ "name", "major", "gpa", "address" ],
      properties: {
         name: {
            bsonType: "string",
            description: "must be a string and is required"
         },
         address: {
            bsonType: "object",
            required: [ "zipcode" ],
            properties: {
                "street": { bsonType: "string" },
                "zipcode": { bsonType: "string" }
            }
         }
      }
   }
}
```

For a list of keywords supported by MongoDB, see Available Keywords.

> **NOTE:**
>
> MongoDB supports draft 4 of JSON Schema, including core specification and validation specification, with some differences. See Extensions and Omissions for details.
>
> For more information about JSON Schema, see the official website.

Behavior

Feature Compatibility

The featureCompatibilityVersion must be set to `"3.6"` or higher in order to use `$jsonSchema`.

Document Validator

You can use `$jsonSchema` in a document validator operations:

```
db.createCollection( <collection>, { validator: { $jsonSchema: <schema> } } )
db.runCommand( { collMod: <collection>, validator:{ $jsonSchema: <schema> } } )
```

Query Conditions

You can use `$jsonSchema` in query conditions for read and write operations to find documents in the collection that satisfy the specified schema:

```
db.collection.find( { $jsonSchema: <schema> } )
db.collection.aggregate( [ { $match: { $jsonSchema: <schema> } } ] )
db.collection.updateMany( { $jsonSchema: <schema> }, <update> )
db.collection.deleteOne( { $jsonSchema: <schema> } )
```

To find documents in the collection that do *not* satisfy the specified schema, use the `$jsonSchema` expression in a `$nor` expression. For example:

```
db.collection.find( { $nor: [ { $jsonSchema: <schema> } ] } )
db.collection.aggregate( [ { $match: { $nor: [ { $jsonSchema: <schema> } ] } }, ... ] )
db.collection.updateMany( { $nor: [ { $jsonSchema: <schema> } ] }, <update> )
db.collection.deleteOne( { $nor: [ { $jsonSchema: <schema> } ] } )
```

Examples

Schema Validation

The following `db.createCollection()` method creates a collection named `students` and uses the `$jsonSchema` operator to set schema validation rules:

```
db.createCollection("students", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "address" ],
         properties: {
            name: {
               bsonType: "string",
               description: "must be a string and is required"
            },
            year: {
               bsonType: "int",
               minimum: 2017,
               maximum: 3017,
               description: "must be an integer in [ 2017, 3017 ] and is required"
            },
            major: {
               enum: [ "Math", "English", "Computer Science", "History", null ],
               description: "can only be one of the enum values and is required"
            },
            gpa: {
               bsonType: [ "double" ],
               description: "must be a double if the field exists"
            },
            address: {
               bsonType: "object",
               required: [ "city" ],
               properties: {
                  street: {
                     bsonType: "string",
                     description: "must be a string if the field exists"
                  },
                  city: {
                     bsonType: "string",
                     "description": "must be a string and is required"
                  }
```

```
            }
mongoDB.  Documentation
         }
      }
   }
})
```

Given the created `validator` for the collection, the following insert operation will fail because `gpa` is an integer when the `validator` requires a `double`.

```
db.students.insert({
   name: "Alice",
   year: NumberInt(2019),
   major: "History",
   gpa: NumberInt(3),
   address: {
      city: "NYC",
      street: "33rd Street"
   }
})
```

The operation returns the following error:

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

After changing the `gpa` to a double, the insert succeeds:

```
db.students.insert({
   name: "Alice",
   year: NumberInt(2019),
   major: "History",
   gpa: 3.0,
   address: {
      city: "NYC",
      street: "33rd Street"
   }
})
```

The operation returns the following:

```
WriteResult({ "nInserted" : 1 })
```

Query Conditions

You can use `$jsonSchema` in query conditions for read and write operations to find documents in the collection that satisfy the specified schema.

For example, create a sample collection `inventory` with the following documents:

```
db.inventory.insertMany([
   { item: "journal", qty: NumberInt(25), size: { h: 14, w: 21, uom: "cm" }, instock: tru
   { item: "notebook", qty: NumberInt(50), size: { h: 8.5, w: 11, uom: "in" }, instock: t
   { item: "paper", qty: NumberInt(100), size: { h: 8.5, w: 11, uom: "in" }, instock: 1 }
   { item: "planner", qty: NumberInt(75), size: { h: 22.85, w: 30, uom: "cm" }, instock:
   { item: "postcard", qty: NumberInt(45), size: { h: 10, w: 15.25, uom: "cm" }, instock:
   { item: "apple", qty: NumberInt(45), status: "A", instock: true },
   { item: "pears", qty: NumberInt(50), status: "A", instock: true }
])
```

Next, define the following sample schema object:

```
let myschema = {
      required: [ "item", "qty", "instock
      properties: {
         item: { bsonType: "string" },
         qty: { bsonType: "int" },
         size: {
            bsonType: "object",
            required: [ "uom" ],
            properties: {
               uom: { bsonType: "string" },
               h: { bsonType: "double" },
               w: { bsonType: "double" }
            }
          },
          instock: { bsonType: "bool" }
      }
}
```

You can use `$jsonSchema` to find all documents in the collection that satisfy the schema:

```
db.inventory.find( { $jsonSchema: myschema } )
db.inventory.aggregate( [ { $match: { $jsonSchema: myschema } } ] )
```

You can use `$jsonSchema` with the `$nor` to find all documents that do not satisfy the schema:

```
db.inventory.find( { $nor: [ { $jsonSchema: myschema } ] } )
```

Or, you can update all documents that do not satisfy the schema:

```
db.inventory.updateMany( { $nor: [ { $jsonSchema: myschema } ] }, { $set: { isValid: fals
```

Or, you can delete all documents that do not satisfy the schema:

```
db.inventory.deleteMany( { $nor: [ { $jso
```

mongoDB. Documentation

JSON Schema

MongoDB supports draft 4 of JSON Schema, including core specification and validation specification, with some differences. See Extensions and Omissions for details.

For more information about JSON Schema, see the official website.

Available Keywords

NOTE:

MongoDB implements a subset of keywords available in JSON Schema. For a complete list of omissions, see Omissions.

Keyword	Type	Definition	Behavior
bsonType	all types	string alias or array of string aliases	Accepts same string aliases used for the `$type` operator
enum	all types	array of values	Enumerates all possible values of the field
type	all types	string or array of unique strings	Enumerates the possible JSON types of the field. Available types are “object”, “array”, “number”, “boolean”, “string”, and “null”. MongoDB’s implementation of the JSON Schema does not support the “integer” type. Use the `bsonType` keyword and the “int” or “long” types instead.
allOf	all types	array of JSON Schema objects	Field must match all specified schemas

Keyword	Type	Definition
anyOf	all types	array of JSON Schema objects	Field must match at least one of the specified schemas
oneOf	all types	array of JSON Schema objects	Field must match exactly one of the specified schemas
not	all types	a JSON Schema object	Field must not match the schema
multipleOf	numbers	number	Field must be a multiple of this value
maximum	numbers	number	Indicates the maximum value of the field
exclusiveMaximum	numbers	boolean	If true and field is a number, `maximum` is an exclusive maximum. Otherwise, it is an inclusive maximum.
minimum	numbers	number	Indicates the minimum value of the field
exclusiveMinimum	numbers	boolean	If true, `minimum` is an exclusive minimum. Otherwise, it is an inclusive minimum.
maxLength	strings	integer	Indicates the maximum length of the field
minLength	strings	integer	Indicates the minimum length of the field
pattern	strings	string containing a regex	Field must match the regular expression
maxProperties	objects	integer	Indicates the field's maximum number of properties
minProperties	objects	integer	Indicates the field's minimum number of properties
required	objects	array of unique strings	Object's property set must contain all the specified elements in the array

Keyword	Type	Definition	Documentation
additionalProperties	objects	boolean or object	If `true`, additional fields are allowed. If `false`, they are not. If a valid JSON Schema object is specified, additional fields must validate against the schema. Defaults to `true`.
properties	objects	object	A valid JSON Schema where each value is also a valid JSON Schema object
patternProperties	objects	object	In addition to `properties` requirements, each property name of this object must be a valid regular expression
dependencies	objects	object	Describes field or schema dependencies
additionalItems	arrays	boolean or object	If an object, must be a valid JSON Schema
items	arrays	object or array	Must be either a valid JSON Schema, or an array of valid JSON Schemas
maxItems	arrays	integer	Indicates the maximum length of array
minItems	arrays	integer	Indicates the minimum length of array
uniqueItems	arrays	boolean	If true, each item in the array must be unique. Otherwise, no uniqueness constraint is enforced.
title	N/A	string	A descriptive title string with no effect.
description	N/A	string	A string that describes the schema and has no effect.

Extensions

MongoDB's implementation of JSON Schema includes the addition of the `bsonType` keyword, which allows you to use all BSON types in the `$jsonSchema` operator. `bsonType` accepts the same string aliases used for

the `$type` operator.

mongoDB. Documentation

Omissions