



WELCOME

Beyond the Basics

Next level database development



Daniel Coupal
Senior Staff Developer Advocate
MongoDB

Three key things I wish all developers knew about

- How to work with arrays
- Database Expressions
- Schema Validation



Working with arrays





Arrays 101

A query will match **any** element in an array

```
query = {"booked_weeks.bookingId": "B75421" }  
db.villas.find(query)
```

We are using 'Week Number' instead of date in these example to make it easier to read! It would be Date() types normally.

```
{  
  _id: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    {  
      from: 2,  
      to: 3,  
      bookingId: "B66462"  
    },  
    {  
      from: 3,  
      to: 4,  
      bookingId: "B76251"  
    },  
    {  
      from: 7,  
      to: 8,  
      bookingId: "B75421"  
    }  
  ]  
}
```



Arrays 101

You can use a dollar (**\$**) in a **projection** to return just that element from the array

```
query = {"booked_weeks.bookingId": "B75421" };  
projection = {"booked_weeks.$": 1};
```

```
db.villas.find(query, projection);
```

```
{  
  _id: "P1234",  
  booked_weeks: [  
    {  
      from: 7,  
      to: 8,  
      bookingId: "B75421"  
    }  
  ]  
}
```



Arrays 101

You can also use **\$ in an update** to modify the matched element

```
query = { "booked_weeks.bookingId": "B75421" };  
update = { $set: { "booked_weeks.$.to": 10} };
```

```
db.villas.updateOne(query, update);
```

```
{  
  _id: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    {  
      from: 2,  
      to: 3,  
      bookingId: "B66462"  
    },  
    {  
      from: 3,  
      to: 4,  
      bookingId: "B76251"  
    },  
    {  
      from: 7,  
      to: 10, <- Was 8  
      bookingId: "B75421"  
    }  
  ]  
}
```



Test time!

Test time!

How many properties are booked before week 2?

```
query = {"booked_weeks.from": { $lt: 2} }
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 6 },  
    { from: 8, to: 9 }  
  ]  
}
```



Test time!

How many properties are booked before week 2?

```
query = {"booked_weeks.from": { $lt: 2} }
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 6 },  
    { from: 8, to: 9 }  
  ]  
}
```



Test time!

How many properties are booked after week 7?

```
query = { "booked_weeks.to": { $gt: 7} }
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 6 },  
    { from: 8, to: 9 }  
  ]  
}
```



Test time!

How many properties are booked after week 7?

```
query = { "booked_weeks.to": { $gt: 7} }
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 6 },  
    { from: 8, to: 9 }  
  ]  
}
```



Test time!

How many properties are booked in week 6?

```
query = { "booked_weeks.from": 6 }
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 7 },  
    { from: 8, to: 9 }  
  ]  
}
```



Test time!

How many properties are booked in week 6?

```
query = {  
  "booked_weeks.from": { $lte: 6 },  
  "booked_weeks.to":   { $gt: 6 }  
}
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 7 },  
    { from: 8, to: 9 }  
  ]  
}
```



Test time!

How many properties are booked in week 6?

```
query = {  
  "booked_weeks.from": { $lte: 6 },  
  "booked_weeks.to":   { $gt: 6 }  
}
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 7 },  
    { from: 8, to: 9 }  
  ]  
}
```



\$elemMatch

How many properties are booked in week 6?

```
elQuery = {  
  from: { $lte: 6 },  
  to: { $gt: 6 }  
}
```

```
query = {  
  booked_weeks: { $elemMatch:  
    elQuery }  
}
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Berry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 2, to: 6 },  
    { from: 8, to: 9 },  
    { from: 9, to: 10 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Cherry Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 5, to: 7 },  
    { from: 8, to: 9 }  
  ]  
}
```





Adding to arrays



\$push

Use **\$push** to add something to an array without reading and writing the whole array

```
query = { property: "P1234" }
```

```
newbooking = { from: 5, to: 6 }
```

```
update = { $push: { booked_weeks:  
           newbooking}}
```

```
properties.updateOne(query, update)
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 }  
  ]  
}
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  booked_weeks: [  
    { from: 1, to: 3 },  
    { from: 3, to: 4 },  
    { from: 7, to: 9 },  
    { from: 5, to: 6 }  
  ]  
}
```



\$each (reviews)

Use **\$push with \$each, \$position, \$slice** for a subset or topN pattern

```
query = { property: "P1234" }  
review = { score: 5,  
           comment: "Very quiet" }
```

```
update = { $push: {  
            latest_reviews: {  
                $each: [review],  
                $position: 0,  
                $slice: 3  
            } } }
```

```
properties.updateOne(query, update)
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  latest_reviews: [  
    {score: 2,comment: "Too dirty"},  
    {score: 4,comment: "Nice view"},  
    {score: 5,comment: "Cheap"}  
  ]  
}
```

```
{ property: "P1234",  
  name: "Apple Villa",  
  sleeps: 5,  
  latest_reviews: [  
    {score: 5,comment: "Very quiet"}  
    {score: 2,comment: "Too dirty"},  
    {score: 4,comment: "Nice view"},  
    {score: 5,comment: "Cheap"}  
  ]  
}
```

Database expressions





\$abs	\$bitNot	\$dateDiff	\$filter	\$isoWeekYear
\$accumulator	\$bitOr	\$dateFromParts	\$first	\$last
\$acos	\$bitXor	\$dateFromString	\$firstN	\$lastN
\$acosh	\$bottom	\$dateSubtract	\$floor	\$let
\$add	\$bottomN	\$dateToParts	\$function	\$linearFill
\$addToSet	\$bsonSize	\$dateToString	\$getField	\$literal
\$allElementsTrue	\$ceil	\$dateTrunc	\$gt	\$ln
\$and	\$cmp	\$dayOfMonth	\$gte	\$locf
\$anyElementTrue	\$concat	\$dayOfWeek	\$hour	\$log
\$arrayElemAt	\$concatArrays	\$dayOfYear	\$ifNull	\$log10
\$arrayToObject	\$cond	\$degreesToRadians	\$in	< Many Removed >
\$asin	\$convert	\$denseRank	\$indexOfArray	
\$asinh	\$cos	\$derivative	\$indexOfBytes	\$trim
\$atan	\$cosh	\$divide	\$indexOfCP	\$trunc
\$atan2	\$count	\$documentNumber	\$integral	\$type
\$atanh	\$covariancePop	\$eq	\$isArray	\$unsetField
\$avg	\$covarianceSamp	\$exp	\$isNumber	\$week
\$binarySize	\$dateAdd	\$expMovingAvg	\$isoDayOfWeek	\$year
\$bitAnd			\$isoWeek	\$zip

Simple query

How can I find all bookings in "London"

```
// SELECT * from bookings  
// WHERE location = "London"
```

```
query = { location: "London" }  
bookings.find(query)
```

```
{  
  bookingId: "B65143",  
  property: "P1234",  
  name: "Apple Villa",  
  location: "London",  
  from: 2,  
  to: 3  
}
```

```
{  
  bookingId: "B66462",  
  property: "P1234",  
  name: "Apple Villa",  
  location: "London",  
  from: 3,  
  to: 8  
}
```

```
{  
  bookingId: "B76251",  
  property: "P1234",  
  name: "Cherry Villa",  
  location: "London",  
  from: 4,  
  to: 6  
}
```



Expressive query

How can I find all bookings in London longer than 3 weeks when **there is no duration field** to query?

```
// SELECT * from bookings  
// WHERE to-from > 3 AND  
// location = "London"
```

```
duration = { $subtract: ["$to", "$from"]}
```

```
query = {  
  location: "London",  
  $expr: { $gt: [duration, 3] }  
}
```

```
{  
  bookingId: "B65143",  
  property: "P1234",  
  name: "Apple Villa",  
  location: "London",  
  from: 2,  
  to: 3  
}
```

```
{  
  bookingId: "B66462",  
  property: "P1234",  
  name: "Apple Villa",  
  location: "London",  
  from: 3,  
  to: 8  
}
```

```
{  
  bookingId: "B76251",  
  property: "P1234",  
  name: "Cherry Villa",  
  location: "London",  
  from: 4,  
  to: 6  
}
```





Expressive projection

How can we return the computed values?

```
// SELECT *, to-from AS duration  
// from bookings WHERE duration > 3 AND  
// location = "London"
```

```
duration = { $subtract: ["$to", "$from"]}
```

```
query = { location: "London",  
          $expr: { $gt: [duration, 3]}}
```

```
projection = { bookingId: 1, from: 1,  
               to: 1, property: 1,  
               name: 1,  
               duration: duration }
```

```
bookings.find(query, projection)
```

```
{  
  bookingId: "B66462",  
  property: "P1234",  
  name: "Apple Villa",  
  from: 3,  
  to: 8,  
  duration: 5  
}
```



Test: What performance issue will we have if we are querying on duration frequently?



Expressive update

Update all the documents to add the computed field

```
// UPDATE bookings
// SET duration = to-from

newDuration = {$subtract:["$to", "$from"]}

query = {} // Match everything

update= [{$set:{ duration: newDuration }}]

bookings.updateMany(query, update)
```

Before

```
{
  bookingId: "B66462",
  property: "P1234",
  name: "Apple Villa",
  from: 3,
  to: 8
}
```

After

```
{
  bookingId: "B66462",
  property: "P1234",
  name: "Apple Villa",
  from: 3,
  to: 8,
  duration: 5
}
```



Expressive Update

When updating **to** or **from**, recompute the field based on the new value and the one in the DB

```
// UPDATE bookings
// SET to=10, duration=10-from
// WHERE bookingId = "B66462"
```

```
query = { bookingId: "B66462" }
```

```
newToDate = 10;
```

```
newDuration = {$subtract: [newToDate, "$from"]}
```

```
update = [{ $set: {to: newToDate,
                  duration: newDuration }} ]
```

```
bookings.updateOne(query, update)
```

Before

```
{
  bookingId: "B66462",
  property: "P1234",
  name: "Apple Villa",
  from: 3,
  to: 8,
  duration: 5
}
```

After

```
{
  bookingId: "B66462",
  property: "P1234",
  name: "Apple Villa",
  from: 3,
  to: 10,
  duration: 7
}
```



And how could
that possibly go
wrong?



Schema validation





CREATE/ALTER TABLE in MongoDB

You can Create and Alter the schema for a collection in MongoDB - you can also use it to fetch the schema

```
CREATE TABLE BOOKINGS (  
  bookingId STRING NOT NULL,  
  propertyId STRING NOT NULL,  
  name STRING not NULL,  
  from Int32 NOT NULL,  
  to Int32 NOT NULL,  
  CHECK ( bookingId LIKE "B%" ),  
  CHECK ( propertyId LIKE "P%" ),  
  CHECK ( from >=1 AND from <=52),  
  CHECK ( to >=1 AND to <=52)  
}
```

```
{  
  bookingId: "B66462",  
  property: null,  
  name: "Apple Villa",  
  from: 3,  
  to: 5  
}
```



\$jsonSchema validation

You can enforce many simple rules by applying a \$jsonSchema validation

```
schema = {}  
schema.$jsonSchema = {  
  bsonType: "object",  
  required: ["bookingId", "property", "name", "from",  
              "to"],  
  properties: {  
    bookingId: {bsonType: "string", "pattern": "^B" },  
    propertyId: {bsonType: "string", "pattern": "^P" },  
    name: {bsonType: "string"},  
    from: {bsonType: "int", minimum: 1, maximum: 52},  
    To: {bsonType: "int", minimum: 1, maximum: 52}  
  }  
}
```

```
db.createCollection("bookings", {validator: schema})
```

```
{  
  bookingId: "B66462",  
  property: "P12345",  
  name: "Apple Villa",  
  from: 3  
}
```

```
Exception: Document failed validation  
Additional information: {  
  details: {  
    operatorName: '$jsonSchema',  
    schemaRulesNotSatisfied: [{  
      operatorName: 'required',  
      specifiedAs: {  
        required: [ "bookingId",  
                  "property", "name", "from", "to" ]},  
        missingProperties: [ "to" ]  
      }  
    ]  
  }  
}
```



CREATE TABLE

```
CREATE TABLE BOOKINGS (  
  bookingId STRING NOT NULL,  
  propertyId STRING NOT NULL,  
  name STRING not null,  
  from Int32 NOT NULL,  
  to Int32 NOT NULL,  
  duration Int32 NOT NULL,  
  CHECK ( bookingId LIKE "B%" ),  
  CHECK ( propertyId LIKE "P%"),  
  CHECK ( from >=1 AND from <=52),  
  CHECK ( to >=1 AND to <= 52),  
  CHECK (duration >=1 AND duration=to - from);
```

```
{  
  bookingId: "B66462",  
  property: null,  
  name: "Apple Villa",  
  from: 3,  
  to: 5,  
  duration: 2  
}
```



Expressive validation

You can also enforce any calculated fields

```
schema = { $jsonSchema: {  
  bsonType: "object",  
  required: ["bookingId", "property", "name", "from",  
            "to", "duration"],  
  properties: {  
    bookingId: {bsonType: "string", "pattern": "^B" },  
    propertyId: {bsonType: "string", "pattern": "^P" },  
    name: {bsonType: "string"},  
    from: {bsonType: "int", minimum: 1, maximum: 52},  
    to: {bsonType: "int", minimum: 1, maximum: 52},  
    duration: {bsonType: "int", minimum: 1, maximum: 52},  
  }  
}}
```

```
schema.$expr = { $eq: ["$duration",  
                       {$subtract: ["$to", "$from"]} ] }
```

```
db.createCollection("bookings", {validator: schema})
```

```
{  
  bookingId: "B66462",  
  property: "P12345",  
  name: "Apple Villa",  
  from: 3,  
  to: 4,  
  duration: 3  
}
```

```
MongoServerError: Document failed  
validation  
Additional information: {  
  details: {  
    operatorName: '$expr',  
    specifiedAs: {  
      $expr: { $eq: ["$duration", {  
$subtract: [ "$to", "$from" ] } ] } },  
      reason: 'expression did not  
match',  
      expressionResult: false  
    }  
  }  
}
```



Conclusion

It pays to know more than most



Learn how to interact with arrays in query and update

Master expressions - they are key to pushing work to the database and avoiding latency

Validate your schemas