I asked the question how to do a simple
update ...
set x = x + 60

In MongoDB. Here is the answer:

db.jimdb.update({Income: 60.0}, {$inc: {Income: 60.00}})

Naturally we can do this also with negative numbers.

{ "_id" : ObjectId("5bee237745bae62a2e3f3306"),
  "Income" : 120.0
}
db.jimdb.update({Income: 20.0}, {$inc: {Income: -10.00}})

//-------------------------------
Straightforward update WITHOUT $set: does a replace!
Example:
Given:

{ "_id" : ObjectId("5bee237745bae62a2e3f3304"),
  "Bonus" : 50.0
}

Now I do this update:

db.jimdb.update({Bonus: 50.0},{Income: 10.0})

Note that Bonus disappeared:

{ "_id" : ObjectId("5bee237745bae62a2e3f3304"),
  "Income" : 10.0
}

But I want both Income and Bonus:

db.jimdb.update({Income: 10.0}, {$set: {Bonus: 50}})

And here it is:
Simple solution of how to create “the equivalent of a third column”

db.jimdb.aggregate(
    {$project: {Income: "$Income",
                Bonus: "$Bonus",
                Total:{$sum: ["$Income", "$Bonus"]}}}
)

Note that the above does NOT change jimdb, it creates a return value that you see with all documents of jimdb, but now with three key/value pairs instead of just two.

The expression "$Income" returns the VALUE of the key $Income. So normally " " means that I want the string AS IS but here "$ " does NOT mean that.

I can *capture* the aggregated collection in a new collection, using $out:

db.jimdb.aggregate([
    {$project: {Income: "$Income",
                Bonus: "$Bonus",
                Total:{ $sum: ["$Income", "$Bonus"]}}},
    {$out: "jimdb6"}
])

db.getCollection("jimdb6").find({})
Now we want to add two fields for one specific document only, not for the whole collection.

I have to do this in steps.

First, I want an aggregate operation, but applied ONLY to one element:

```javascript
db.jimdb.aggregate([{ $match: { Income: 10.0 } }])
```

This $match works like a where clause and the aggregate starts behaving like a select suddenly.

Combining the last two solutions (without a new $out) gives:

```javascript
db.jimdb.aggregate([{ $match: { Income: 10.0 } },
    { $project: { Income: "$Income",
                 Bonus: "$Bonus",
                 Total: { $sum: ["$Income", "$Bonus"] } } }])
```

Here is a really overly complex solution.

I need to use $add but “hide it from the $aggregate” within a $sum.

```javascript
db.jimdb.aggregate([{ $match: { Income: 10.0 } },
    { $group: { _id: 99, total: { $sum: { $add: ["$Income", "$Bonus"] } } } }])
```

The above returns correctly Income+Bonus

```javascript
{
  "_id": 99.0,
  "total": 60.0
}
```

//  -----------

The $let operation allows you to define local variables and do computation with them. More complicated solutions follow.

“$$low” returns the value of the local variable low define in vars:

```javascript
{
  $let:
```
This returns: false

This will not run by itself. It has to be built into an aggregate command:

```javascript
db.jimdb.aggregate( [

{ $project: {
    finalTotal: {
        $let: {
            vars: { low: 1, high: 5 },
            in: { $sum: [ "low", "high" ] } } } }

] )
```

This returns: 6

//-----------------------------

But we can ignore the local variables and use the Key/Values from the current collection in the program. Here we use the keys $Income and $Bonus.

```javascript
db.jimdb.aggregate( [

{ $project: {
    finalTotal: {
        $let: {
            vars: { low: 1, high: 5 },
            in: { $sum: [ "$Income", "$Bonus" ] } } } }

] )
```

// ------

Below we generate a new collection that contains the total AND the given data but “in the wrong order.” Later we put the Total: at the end.

```javascript
db.jimdb.aggregate( [

{ $project: {
    Total: {
        $let: {
            vars: { low: 1, high: 5 },
            in: { $sum: [ "$Income", "$Bonus" ] } } }

] )
```
We'd like to have the given data first and the total after. Also if we don't use the local variables, we don't need to define them. We still need the {} though.

db.jimdb.aggregate( [
    {
        $project: {
            Income: "$Income",
            Bonus: "$Bonus",
            Total: {
                $let: {
                    vars: {},
                    in: { $sum: [ "$Income", "$Bonus" ] }
                }
            }
        }
    },
    { $out: "jimdb5" }
] )

db.jimdb5.find({})

{ "_id" : ObjectId("5bee237745bae62a2e3f3303"),
  "Income" : 40.0,
  "Bonus" : 35.0,
  "Total" : 75.0 }

{ "_id" : ObjectId("5bee237745bae62a2e3f3305"),
  "Income" : 90.0,
  "Bonus" : 45.0,
  "Total" : 135.0 }

{ "_id" : ObjectId("5bee237745bae62a2e3f3306"),
  "Income" : 120.0,
  "Bonus" : 65.0,
  "Total" : 185.0 }

{ "_id" : ObjectId("5bfdd75645bae62a2e3f3307"),
  "Income" : 10.0,
  "Bonus" : 50.0,}
Yes, MongoDB has cursors. In fact a “find()” returns a cursor. A find().sort() processes this return value.

This cursor can be used in a JavaScript program.

hasNext() returns true if there is a next value for the cursor. next() returns the next value of the cursor.

First we assign the cursor to a variable.

```javascript
var myCursor = db.jimdb.find({});
```

Display the data on the screen (for use of Studio3T).

```javascript
while (myCursor.hasNext()) {
    print(tojson(myCursor.next()));
}
```

A little bit more of JavaScript.

Concatenate some text to the result.

```javascript
while (myCursor.hasNext()) {
    print("test" + tojson(myCursor.next()));
}
```

Create document numbers for the result:

```javascript
var i = 0

while (myCursor.hasNext()) {
    print(i++ + "": " + tojson(myCursor.next()));
}
```

Here is the output.

```javascript
0: {
    "_id": ObjectId("5bee237745bae62a2e3f3303"),
    "Income": 40,
    "Bonus": 35
}
1: {
    "_id": ObjectId("5bee237745bae62a2e3f3305"),
    "Income": 90,
    "Bonus": 45
```
2: {
    "_id" : ObjectId("5bee237745bae62a2e3f3306"),
    "Income" : 120,
    "Bonus" : 65
}

3: {
    "_id" : ObjectId("5bfdd75645bae62a2e3f3307"),
    "Income" : 10,
    "Bonus" : 50
}