With NoSQLBooster for MongoDB, you can run SQL SELECT Query against MongoDB. SQL support includes functions, expressions, aggregation for collections with nested objects and arrays.  
See [the features and SQL examples](https://mongobooster.com/featuresForSQL) supported by the NoSQLBooster for MongoDB.

Let’s look at how to use the GROUP BY clause with the SUM function in SQL.

Instead of writing the MongoDB query which is represented as a JSON-like structure

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | db.employees.aggregate([  {  $group: {  \_id: "$department",  total: { $sum: "$salary" }  },  } ]) |

You can query MongoDB by using old SQL which you probably already know.

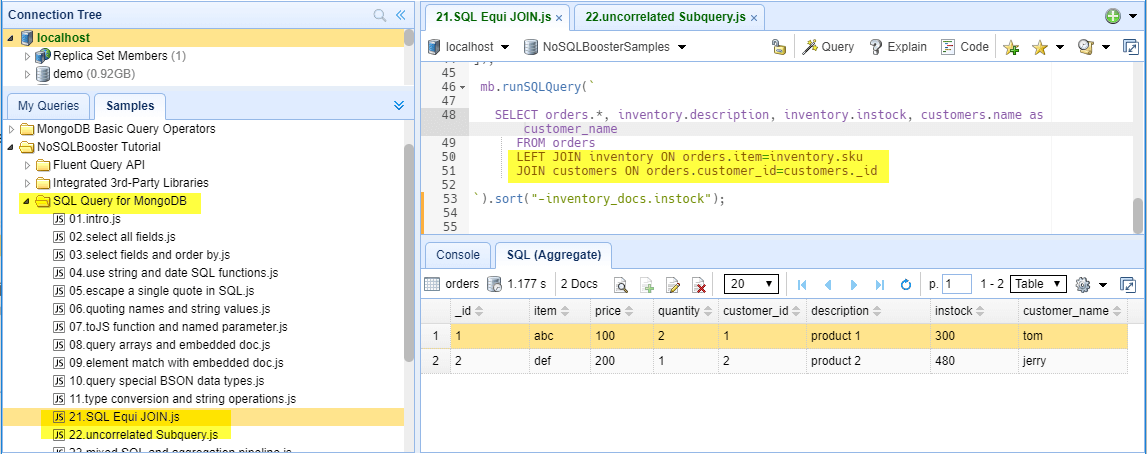
|  |  |
| --- | --- |
| 1 2 3 4 5 | mb.runSQLQuery(`   SELECT department, SUM(salary) AS total FROM employees GROUP BY department  `); |

**Features**

* Access data via SQL including WHERE filters, ORDER BY, GROUP BY, HAVING, DISTINCT, LIMIT
* SQL Functions (COUNT, SUM, MAX, MIN, AVG)
* SQL Functions (Date, String, Conversion)
* SQL Equi JOIN and Uncorrelated SubQuery
* Aggregation Pipeline Operators as SQL Functions (dateToString, toUpper, split, substr…)
* Provide a programming interface (mb.runSQLQuery) that can be integrated into your script
* Autocomplete for keywords, MongoDB collection names, field names, and SQL functions

Please note that MongoDB does not natively support SQL features. The SQL query is validated and translated into a MongoDB query and executed by NoSQLBooster for MongoDB. The Equivalent MongoDB Query can be viewed in the console.log tab.

It should be mentioned that there is a tutorial on NoSQLBooster SQL Query for MongoDB in the lower left “Samples” pane. With this tutorial, you can learn and understand how to use NoSQLBooster SQL Query for MongoDB. Better yet, all SQL Functions provide the appropriate code snippets and mouse hover information and support code completion.

[](https://www.nosqlbooster.com/s3/img/sql-equi-join.png)SQL Query for MongoDB Tutorial

**Getting Started**

For example, the employees’ collection has the following fields including number, first\_name, last\_name, salary, department, and hire\_date.

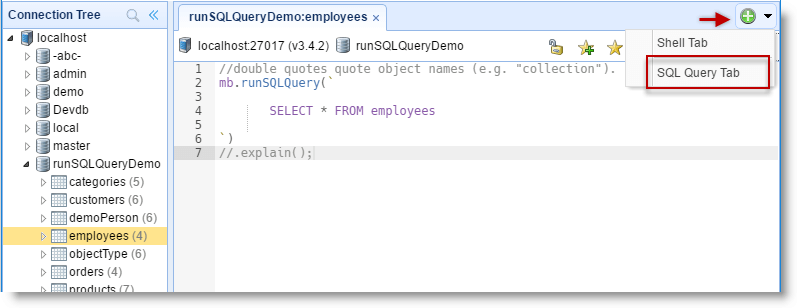
**Prepare Demo Data**

Insert the following demo data to MongoDB. Open a shell tab “Command-T” and execute the following script to get the employees collection.

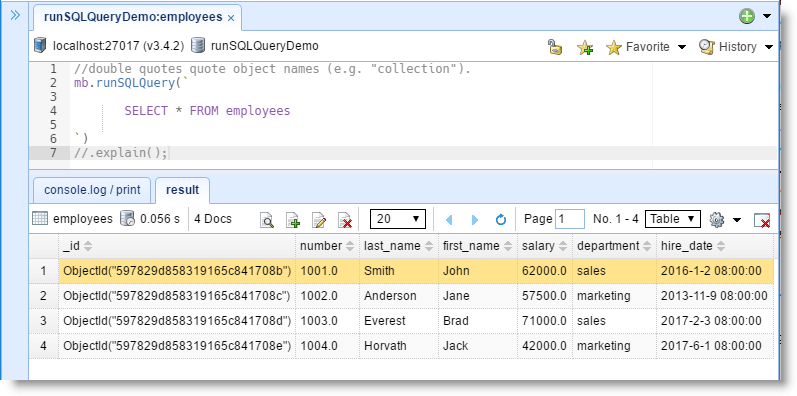
|  |  |
| --- | --- |
| 1 2 3 4 5 6 | db.employees.insert([   {"number":1001,"last\_name":"Smith","first\_name":"John","salary":62000,"department":"sales", hire\_date:ISODate("2016-01-02")},  {"number":1002,"last\_name":"Anderson","first\_name":"Jane","salary":57500,"department":"marketing", hire\_date:ISODate("2013-11-09")},  {"number":1003,"last\_name":"Everest","first\_name":"Brad","salary":71000,"department":"sales", hire\_date:ISODate("2017-02-03")},  {"number":1004,"last\_name":"Horvath","first\_name":"Jack","salary":42000,"department":"marketing", hire\_date:ISODate("2017-06-01")}, ]) |

**Select All Fields**

First, click on employees collection, then click on “SQL Query Tab” in the tab toolbar or use the “Command-Alt-T” keyboard shortcut. A basic “SELECT \* from employees” is automatically generated for us! NoSQLBooster for MongoDB also offers a “runSQLQuery” code snippets. Just type a snippet prefix “run”, and press “tab” to insert this snippet.

[](https://www.nosqlbooster.com/s3/img/sql-query-basic.png)alt text

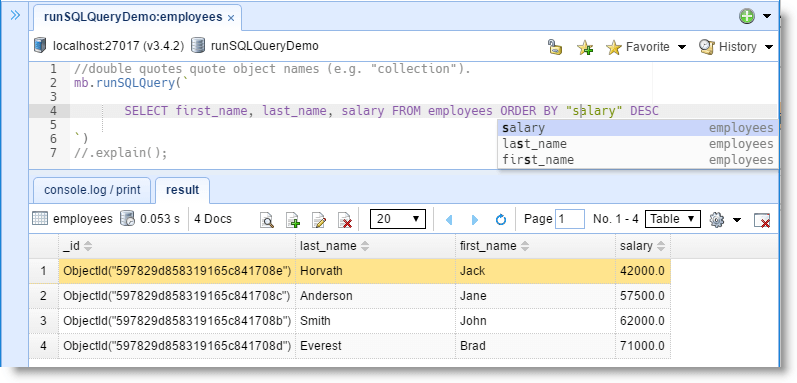
Just execute the query by clicking the execute button or use the “Command-Enter” keyboard shortcut. This would produce the result as shown below.

[](https://www.nosqlbooster.com/s3/img/sql-basic-result.png)alt text

* NoSQLBooster for MongoDB provides in-place editing in result tree view. Double-click on any value or array element to edit. Pressing “Esc” return the previous value and exit the editor.
* If you want the results not to be edited directly, you can enable the “read-only” mode by clicking the lock button in the toolbar.

**Select Individual Fields and Field Name Auto-complete**

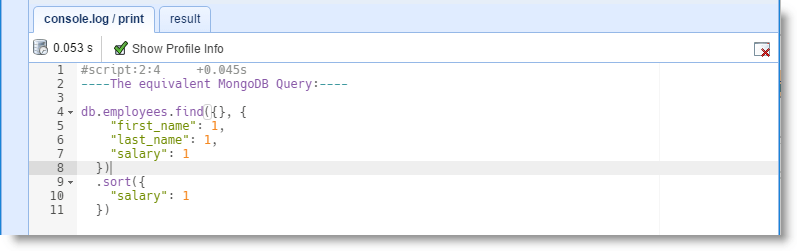
Let’s fetch the first\_name, last\_name and salary fields of the employees available in employees table and sort the result in the descending order by salary.

[](https://www.nosqlbooster.com/s3/img/sql-select-fields-order-by.png)alt text

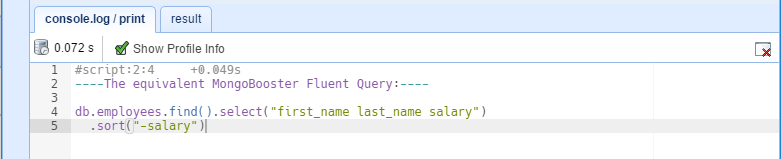
The built-in SQL language service knows all possible completions, SQL functions, keywords, MongoDB collection names, and field names. The IntelliSense suggestions pop up as you type. You can always manually trigger it with “Ctrl-Shift-Space”}}}. Out of the box, “Ctrl-Space”}}}, “Alt-Space”}}} are also acceptable triggers.

**View the Equivalent MongoDB Query**

Just Click on the “console.log/print” tab to show the equivalent MongoDB query:

[](https://www.nosqlbooster.com/s3/img/sql-query-mongodb-json-like.png)alt text

As you know, NoSQLBooster for MongoDB supports [mongoose-like fluent Query API](https://mongobooster.com/FluentQueryAPI), Click Menu-> Options -> Translate SQL to MongoDB Shell Script, click “Translate SQL to NoSQLBooster for MongoDB Fluent API”. Re-execute the script, the equivalent fluent MongoDB query will be shown in the “console.log/print” tab.

[](https://www.nosqlbooster.com/s3/img/sql-query-fluent-api.png)alt text

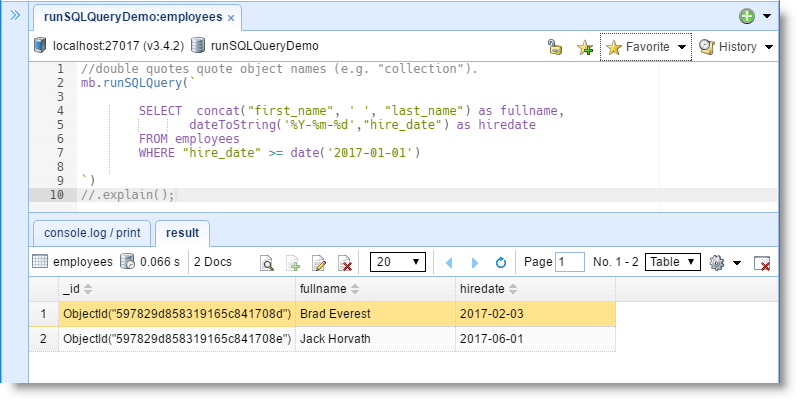
**Use String and Date SQL Functions**

This time, we want to find all employees who are hired this year, and display first\_name and last\_name as full name. Please enter the following SQL statement and click execute button:

|  |  |
| --- | --- |
| 1 2 3 4 | SELECT concat("first\_name", ' ', "last\_name") as fullname,  dateToString('%Y-%m-%d',"hire\_date") as hiredate  FROM employees  WHERE "hire\_date" >= date('2017-01-01') |

Clicking on the “console.log/print” tab to show the equivalent MongoDB query:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 | db.employees.aggregate(  [{  "$match": {  "hire\_date": {  "$gte": ISODate("2017-01-01T08:00:00.000+08:00")  }  }  },  {  "$project": {  "fullname": {  "$concat": [  "$first\_name",  " ",  "$last\_name"  ]  },  "hiredate": {  "$dateToString": {  "format": "%Y-%m-%d",  "date": "$hire\_date"  }  }  }  }  ]) |

[](https://www.nosqlbooster.com/s3/img/sql-query-concat-datestring.png)alt text

Let us look at concat function **concat(“first\_name”, ‘ ‘, “last\_name”)**. The concat function is a MongoDB string aggregation operators. Through mapping SQL functions to MongoDB operators, NoSQLBooster for MongoDB allows you to use all MongoDB aggregation operators as SQL functions in you SQL statement.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | // instead of writing { $concat: [ "$first\_name", " ", "last\_name" ] }  // we write,  concat("first\_name", ' ', "last\_name") //Double quotes quote field name, Single quotes are for strings  // or  concat(first\_name, ' ', last\_name) //allow to omit double quotes |

1. No $ prefix with MongoDB operator and collection field name.
2. **Double quotes quote field name. Single quotes are for strings.**
3. All function names are case-sensitive except for COUNT, SUM, MAX, MIN, AVG.
4. We can use the standard SQL comparison operators: =, !=, <>, <, <=, >=, or >.

The date function converts a string to a MongoDB Date type. NoSQLBooster for MongoDB uses Moment.js to parse date string. When creating a date from a string, Moment.js first check if the string matches the known ISO 8601 formats, Moment.js then check if the string matches the RFC 2822 Date time format before dropping to the fall back of new Date(string) if a known format is not found.

Please refer to [Moment.js parse string document](https://momentjs.com/docs/#/parsing/string/) to view all supported string formats.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | # An ISO 8601 string requires a date part.  2013-02-08 # A calendar date part  #A time part can also be included, separated from the date part by a space or an uppercase T. 2013-02-08 09:30 # An hour and minute time part 2013-02-08 09:30:26 # An hour, minute, and second time part  #If a time part is included, an offset from UTC can also be included as +-HH:mm, +-HHmm, +-HH or Z.  2017-01-01T08:00:00.000+08:00 2013-02-08 09+07:00 # +-HH:mm 2013-02-08 09:30:26.123+07:00 # +-HH:mm |

The dateToString is another MongoDB date operator to convert a date object to a string according to a user-specified format. The [$dateToString](https://docs.mongodb.com/manual/reference/operator/aggregation/dateToString/#exp._S_dateToString) expression has the following syntax:

|  |  |
| --- | --- |
| 1 | { $dateToString: { format: <formatString>, date: <dateExpression> } } |

As SQL functions doesn’t support JSON object parameter, NoSQLBooster for MongoDB converts the object param as plain parameter list.

|  |  |
| --- | --- |
| 1 | dateToString('%Y-%m-%d',"hire\_date") as hiredate |

The first parameter is formatString, single quotes, the second parameter is “Date Field”, double quotes.

Please follow [this link](https://docs.mongodb.com/manual/reference/operator/aggregation-date/) to learn more about MongoDB date aggregation operators.

**Quoting Names and String Values**

In NoSQLBooster, we follow ANSI SQL standard. Single quotes delimit a string constant or a date/time constant. Double quotes delimit identifiers e.g., collection names or column names. This is generally only necessary when your identifier doesn’t fit the rules for simple identifiers.

The following SQL statement selects all the customers from the department “sales,” in the “employees” collection:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM employees WHERE department='sales'; |

The equivalent MongoDB query

|  |  |
| --- | --- |
| 1 2 3 | db.employees.find({  "department": "sales" }) |

Single quotes are for strings. If you double quotes “sales,” NoSQLBooster for MongoDB treat it as a column “sales,” not a string ‘sales’.

|  |  |
| --- | --- |
| 1 | SELECT \* FROM employees WHERE department="sales"; |

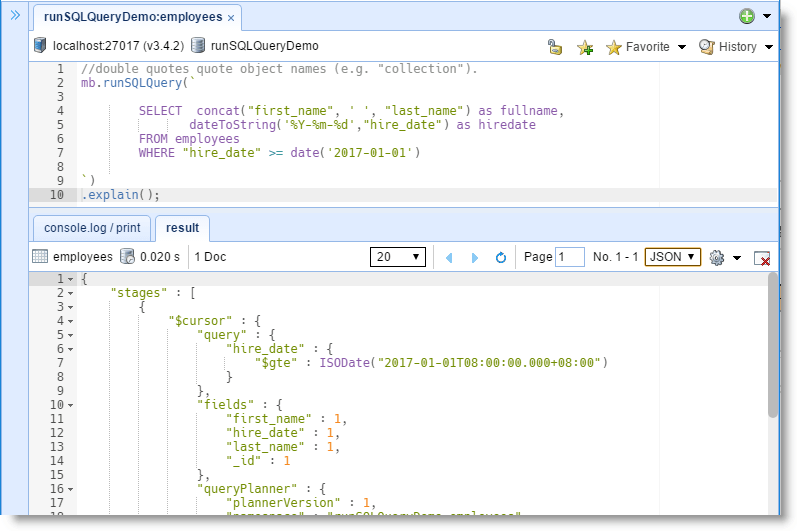
The SQL generated the following MongoDB query that you may not want

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | db.employees.aggregate(  [{  "$addFields": {  "\_\_tmp\_cond\_1": {  "$eq": [  "$department",  "$sales"  ]  }  }  },  {  "$match": {  "\_\_tmp\_cond\_1": true  }  },  {  "$project": {  "\_\_tmp\_cond\_1": 0  }  }  ]) |

Stick to using single quotes.

**Explain Data for the Query**

Just add a new line “.explain()” to the end of mb.runSQLQuery and click execute button again. It returns the information on the processing of the pipeline.

[](https://www.nosqlbooster.com/s3/img/sql-query-explain.png)alt text

**Querying Special BSON Data Types, UUID, BinData, DBRef…**

To query values of these particular BSON Data types, write the values as you would in the MongoDB shell. All MongoDB build-in Data Types functions are available.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 | --date SELECT \* FROM collection WHERE date\_field >= date("2018-02-09T00:00:00+08:00") SELECT \* FROM collection WHERE date\_field >= ISODate("2018-02-09")  --number SELECT \* FROM collection WHERE int64\_field >= NumberLong("3223123123122132992")  SELECT \* FROM collection WHERE decimal\_field = NumberDecimal("8989922322323232.12")  --Regular Expression SELECT \* FROM collection WHERE string\_field = RegExp('query','i')  --binary SELECT \* FROM collection WHERE objectId\_field = ObjectId("56034dae9b835b3ee6a52cb7") SELECT \* FROM collection WHERE binary\_field = BinData(0,"X96v3g==") SELECT \* FROM collection WHERE md5\_field = MD5("f65485ac0686409aabfa006f0c771fbb") SELECT \* FROM collection WHERE hex\_field = HexData(0,"00112233445566778899aabbccddeeff")  --uuid SELECT \* FROM collection WHERE uuid\_field = UUID("4ae5bfce-1dba-4776-80eb-17678822b94e") SELECT \* FROM collection WHERE luuid\_field = LUUID("8c425c91-6a72-c25c-1c9d-3cfe237e7c92") SELECT \* FROM collection WHERE luuid\_field = JSUUID("8c425c91-6a72-c25c-1c9d-3cfe237e7c92") SELECT \* FROM collection WHERE luuid\_field = CSUUID("6a72c25c-5c91-8c42-927c-7e23fe3c9d1c") SELECT \* FROM collection WHERE luuid\_field = PYUUID("5cc2726a-915c-428c-927c-7e23fe3c9d1c")  --timstamp SELECT \* FROM collection WHERE timestamp\_field = Timestamp(1443057070, 1) --symbol SELECT \* FROM collection WHERE symbol\_field = Symbol('I am a symbol') --dbref SELECT \* FROM collection WHERE dbref\_field = DBRef("unicorns", ObjectId("55f23233edad44cb25b0d51a")) --minkey maxkey SELECT \* FROM collection WHERE minkey\_field = MinKey and maxkey\_field = MaxKey  --array, array\_field is [1, 2, '3'] SELECT \* FROM collection WHERE array\_field = [1,2,'3']  --object, object\_field is { a : 1, b : {b1 : 2, b2 : "b2"} SELECT \* FROM collection WHERE object\_field = toJS(a=1, b=toJS(b1=2, b2='b2')) |

**Accessing Arrays and Embedded Documents**

Nested documents (sub-documents) and arrays including filters and expressions are supported. You can access such fields using a dotted name.

Given the following documents in the survey collection:

|  |  |
| --- | --- |
| 1 2 3 4 5 | db.survey.insert([   { \_id: 1, results: [ { product: "abc", score: 10 }, { product: "xyz", score: 5 } ]},  { \_id: 2, results: [ { product: "abc", score: 8 }, { product: "xyz", score: 7 } ]},  { \_id: 3, results: [ { product: "abc", score: 7 }, { product: "xyz", score: 8 } ]} ]) |

The “product” and “score” would be referenced as results.product and results.score respectively:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM survey WHERE results.product = 'xyz' AND results.score >= 8; |

or

|  |  |
| --- | --- |
| 1 | SELECT \* FROM survey WHERE "results.product" = 'xyz' AND "results.score" >= 8; |

***Element Match with Embedded Documents***

 The elemMatch query criteria ***(score >=8)*** will be translated as ***“score”: { “$gte”: 8 }***. This syntax is more concise and expressive.

|  |  |
| --- | --- |
| 1 2 | -- Enter "elemMatch [Tab]", to trigger auto-complete SELECT \* FROM survey WHERE "results" =elemMatch(product='xyz', score >=8) |

**SQL Equi JOIN**

NoSQLBooster supports SQL Equi JOIN which performs a JOIN against equality or matching column(s) values of the associated tables.  
An equal sign (=) is used as comparison operator in the where clause to refer equality.

NoSQLBooster supports INNER JOIN and LEFT JOIN, OUTER JOIN is not supported.

1. (INNER) JOIN: Returns records that have matching values in both tables
2. LEFT JOIN: Return all records from the left table, and the matched records from the right table

Equi JOIN will be translated as MongoDB $lookup stage. (MongoDB 3.2+) refer to: <https://docs.mongodb.com/manual/reference/operator/aggregation/lookup>

|  |  |
| --- | --- |
| 1 | SELECT \* FROM orders JOIN inventory ON orders.item=inventory.sku |

**The “toJS” SQL Function and named parameter**

The “toJS” helper function transforms the named parameters and arithmetic operator into a JSON object, also transforms an ordinary parameter list into an array.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | toJS(k='v'); //result {k:'v'} toJS(k="v"); //result {k:'$v'}, Double quotes quote object names toJS(k=v); //result {k:'$v'}, without quote, v is a object name toJS(k>5, k<=10); //result { "k": { "$gt": 5, "$lte": 10} } toJS(a=1, b=toJS(b1=2, b2='b2')); //result {a : 1, b : {b1 : 2, b2 : "b2"} toJS(1, 2,'3'); // result [1,2,'3']; |

With named parameter and “toJS” helper function, you can query complex objects or pass JSON-object parameter to a SQL function.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | --elemMatch, named parameter and Arithmetic operators --cool stuff, (score>8, score<=10) will be translated as {"score": { "$gt": 8, "$lte": 10 }} SELECT \* FROM survey WHERE results =elemMatch(item='abc', score>8, score<=10)  --date timezone, named parameter SELECT year(date="date", timezone='America/Chicago') as year FROM sales  --query object type value, object\_field: { a : 1, b : {b1 : 2, b2 : "b2"} SELECT \* FROM collection WHERE object\_field = toJS(a=1, b=toJS(b1=2, b2='b2'))  -- text search with full-text search options SELECT \* FROM article WHERE   $text = toJS($search='cake', $language='en', $caseSensitive=false, $diacriticSensitive=false)   SELECT literal(toJS(k>5, k<=10)) FROM collection |

**Mixed use of SQL and Chainable Aggregation Pipeline**

NoSQLBooster for MongoDB translates SQL to MongoDB find/aggregate method which returns a AggregateCursor. All MongoDB [cursor methods](https://docs.mongodb.com/manual/reference/method/js-cursor/) and NoSQLBooster for MongoDB’s [extension methods](https://mongobooster.com/FluentQueryAPI#CursorExtensionMethods) can be called. This also allows NoSQLBooster Intellisense to know all AggregateCursor’s chainable stage methods. (sort, limit, match, project, unwind…)

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | mb.runSQLQuery(`   SELECT \* FROM "survey" where type='ABC' and year(date) = 2018 ORDER BY "results.score" DESC   `) .unwind('$tags') .project("-\_id") //alias select .limit(1000) |

The equivalent MongoDB Query is a bit longer.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 | db.survey.aggregate( [  {  "$addFields" : {  "year(date)" : {  "$year" : "$date"  }  }  },  {  "$match" : {  "type" : "ABC",  "year(date)" : 2018  }  },  {  "$project" : {  "year(date)" : 0  }  },  {  "$sort" : {  "results.score" : -1  }  },  {  "$unwind" : "$tags"  },  {  "$project" : {  "\_id" : 0  }  },  {  "$limit" : 1000  } ]) |

**Save SQL Query as MongoDB Read-only View**

You can use the extension method “saveAsView” to save SQL Query result as a MongoDB read-only view.

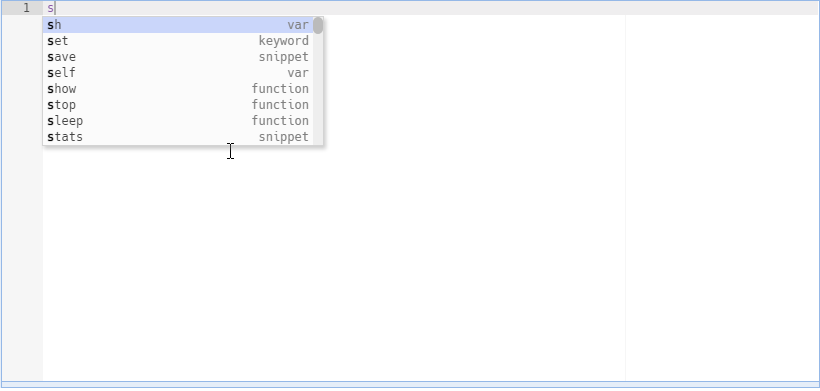
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | //double quotes quote object names (e.g. "collection"). Single quotes are for strings 'string'. mb.runSQLQuery("  SELECT concat(first\_name, ' ', last\_name) as fullname,  dateToString('%Y-%m-%d',hire\_date) as hiredate   FROM employees   WHERE hire\_date >= date('2017-01-01')    ").saveAsView("Employers\_hired\_after\_2017", {dropIfExists:true}) //drop view if it exists. |

You can also use the forEach method to apply a javascript method to each document.

|  |  |
| --- | --- |
| 1 2 3 4 | mb.runSQLQuery("SELECT \* FROM employees WHERE hire\_date >= date('2017-01-01')")  .forEach(it=>{  //sendToMail(it)  }); |

**SQL Snippets**

NoSQLBooster includes a lot of SQL-specific code snippets to save you time, Date Range, Text Search, Query and Array, Existence Check, Type Check and so on. You can always manually trigger it with “Ctrl-Shift-Space”}}}. Out of the box, “Ctrl-Space”}}}, “Alt-Space”}}} are acceptable triggers.

[](https://www.nosqlbooster.com/s3/img/sql-snippet-daterange.gif)SQL Snippet

**SQL Date Range Snippets**

|  |  |
| --- | --- |
| 1 2 3 | -- Enter "daterange [Tab]," then..., today, yesterday, lastNDays SELECT \* FROM collection WHERE  "|" >= date("2018-02-09T00:00:00+08:00") and "|" < date("2018-02-10T00:00:00+08:00") |

**Text Search Snippets**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | -- Enter "text [Tab]", then... SELECT \* FROM collection WHERE $text = toJS($search='|')  -- Enter "textopt [Tab]", then... -- with full-text search options SELECT \* FROM collection WHERE   $text = toJS($search='|', $language='en', $caseSensitive=false, $diacriticSensitive=false) |

The equivalent MongoDB Text Search

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | db.collection.find({  "$text": {  "$search": "|",  "$language": "en",  "$caseSensitive": false,  "$diacriticSensitive": false  } }) |

**Query an Array ($all and $elemMatch) Snippets**

The $elemMatch operator matches documents that contain an array field with at least one element that matches all the specified query criteria.

* Element Match with Embedded Documents

 The elemMatch query criteria ***(quantity>2, quantity<=10)*** will be translated as ***“quantity”: { “$gt”: 2, “$lte”: 10 }***. This syntax is more concise and expressive.

|  |  |
| --- | --- |
| 1 2 3 | -- Enter "elemem [Tab]", then... --  SELECT \* FROM survey WHERE "|" =elemMatch(product='abc', quantity>2, quantity<=10) |

The equivalent MongoDB Query

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | db.survey.find({  "|": {  "$elemMatch": {  "product": "abc",  "quantity": {  "$gt": 2,  "$lte": 10  }  }  } }) |

* Element Match

|  |  |
| --- | --- |
| 1 2 | -- Enter "elem [Tab]", then... SELECT \* FROM survey WHERE "score" =elemMatch($gte=80, $lte=85) |

The equivalent MongoDB Query

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | db.survey.find({  "score": {  "$elemMatch": {  "$gte": 80,  "$lte": 85  }  } }) |

The $all array operator selects the documents where the value of a field is an array that contains all the specified elements.

|  |  |
| --- | --- |
| 1 2 | -- Enter "all [Tab]", then... SELECT \* FROM survey WHERE "|" = toJS($all=['', '']) |

**Existence Check and Type Check Snippets**

* Existence Check ($exists)

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | -- Enter "exists [Tab]", then... SELECT \* FROM collection WHERE "|" =$exists(TRUE)  -- Enter "nonExist [Tab]", then... SELECT \* FROM collection WHERE "|" = $exists(FALSE)  -- Enter "existAndIsNull [Tab]", then... SELECT \* FROM collection WHERE "|" = $exists(TRUE) and "|" IS NULL  -- Enter "existAndIsNotNull [Tab]", then... SELECT \* FROM collection WHERE "|" = $exists(TRUE) and "|" IS NOT NULL |

* Querying by Multiple Data Type ($type)

|  |  |
| --- | --- |
| 1 2 | -- Enter "typeSearch [Tab]", then... SELECT \* FROM collection WHERE "|" = toJS($type=['double','int','string','bool','date','object','array','null']) |

**Complete List of SQL Support**

To check out the complete list of SQL Support, visit [this link](https://mongobooster.com/blog/query-mongodb-with-sql/featuresForSQL).

**Thank you!**

Please visit our [feedback page](https://mongobooster.useresponse.com/) or click the “Feedback” button in the app. Feel free to suggest improvements to our product or service. Users can discuss your suggestion and vote for and against it. We’ll look at it too.