

# QBAggregate

## Property Summary

QBFunction	<a href="#">abs</a> Create abs(column) expression
QBSort	<a href="#">asc</a> Create an ascending sort expression
QBAggregate	<a href="#">avg</a> Create an aggregate expression.
QBFunction	<a href="#">bit_length</a> Create bit_length(column) expression
QBFunction	<a href="#">ceil</a> Create ceil(column) expression
QBAggregate	<a href="#">count</a> Create an aggregate expression.
QBFunction	<a href="#">day</a> Extract day from date
QBSort	<a href="#">desc</a> Create an descending sort expression
QBFunction	<a href="#">floor</a> Create floor(column) expression
QBFunction	<a href="#">hour</a> Extract hour from date
QBCondition	<a href="#">isNull</a> Compare column with null.
QBFunction	<a href="#">len</a> Create length(column) expression
QBFunction	<a href="#">lower</a> Create lower(column) expression
QBAggregate	<a href="#">max</a> Create an aggregate expression.
QBAggregate	<a href="#">min</a> Create an aggregate expression.
QBFunction	<a href="#">minute</a> Extract minute from date
QBFunction	<a href="#">month</a> Extract month from date
QBColumn	<a href="#">not</a> Create a negated condition.
QBTableClause	<a href="#">parent</a> Get query builder parent table clause, this may be a query or a join clause.
QBSelect	<a href="#">root</a> Get query builder parent.
QBFunction	<a href="#">round</a> Create round(column) expression
QBFunction	<a href="#">second</a> Extract second from date
QBFunction	<a href="#">sqrt</a> Create sqrt(column) expression
QBAggregate	<a href="#">sum</a> Create an aggregate expression.
QBFunction	<a href="#">trim</a> Create trim(column) expression
QBFunction	<a href="#">upper</a> Create upper(column) expression
QBFunction	<a href="#">year</a> Extract year from date

## Method Summary

QBCondition	<a href="#">between</a> (value1, value2) Compare column to a range of 2 values or other columns.
QBFunction	<a href="#">cast</a> (type) Create cast(column, type) expression
QBFunction	<a href="#">concat</a> (arg) Concatenate with value

QBFunction	<a href="#">divide(arg)</a> Divide by value
QBCondition	<a href="#">eq(value)</a> Compare column with a value or another column.
QBCondition	<a href="#">ge(value)</a> Compare column with a value or another column.
QBCondition	<a href="#">gt(value)</a> Compare column with a value or another column.
QBCondition	<a href="#">isin(query)</a> Compare column with subquery result.
QBCondition	<a href="#">isin(values)</a> Compare column with values.
QBCondition	<a href="#">le(value)</a> Compare column with a value or another column.
QBCondition	<a href="#">like(pattern)</a> Compare column with a value or another column.
QBCondition	<a href="#">like(pattern, escape)</a> Compare column with a value or another column.
QBFunction	<a href="#">locate(arg)</a> Create locate(arg) expression
QBFunction	<a href="#">locate(arg, start)</a> Create locate(arg, start) expression
QBCondition	<a href="#">lt(value)</a> Compare column with a value or another column.
QBFunction	<a href="#">minus(arg)</a> Subtract value
QBFunction	<a href="#">mod(arg)</a> Create mod(arg) expression
QBFunction	<a href="#">multiply(arg)</a> Multiply with value
QBFunction	<a href="#">nullif(arg)</a> Create nullif(arg) expression
QBFunction	<a href="#">plus(arg)</a> Add up value
QBFunction	<a href="#">substring(pos)</a> Create substring(pos) expression
QBFunction	<a href="#">substring(pos, len)</a> Create substring(pos, len) expression

## Property Details

### abs

Create abs(column) expression

### Returns

[QBFunction](#)

### Sample

```
query.result.add(query.columns.custname.abs)
```

### asc

Create an ascending sort expression

### Returns

[QBSort](#)

### Sample

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.sort
.add(query.joins.orders_to_order_details.columns.quantity.asc)
.add(query.columns.companyid)
foundset.loadRecords(query)
```

### avg

Create an aggregate expression.

---

**Returns**[QBAggregate](#)**Sample**

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.groupBy.addPk() // have to group by on pk when using having-conditions in (foundset) pk queries
.root.having.add(query.joins.orders_to_order_details.columns.quantity.avg.eq(1))
foundset.loadRecords(query)
```

**bit\_length**

Create bit\_length(column) expression

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.custname.bit_length)
```

**ceil**

Create ceil(column) expression

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.mycol.ceil)
```

**count**

Create an aggregate expression.

**Returns**[QBAggregate](#)**Sample**

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.groupBy.addPk() // have to group by on pk when using having-conditions in (foundset) pk queries
.root.having.add(query.joins.orders_to_order_details.columns.quantity.count.eq(0))
foundset.loadRecords(query)
```

**day**

Extract day from date

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.mydatecol.day)
```

**desc**

Create an descending sort expression

**Returns**[QBSort](#)

---

**Sample**

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.sort
.add(query.joins.orders_to_order_details.columns.quantity.desc)
.add(query.columns.companyid)
foundset.loadRecords(query)
```

**floor**

Create floor(column) expression

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.mycol.floor)
```

**hour**

Extract hour from date

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.mydatecol.hour)
```

**isNull**

Compare column with null.

**Returns**

[QBCondition](#)

**Sample**

```
query.where.add(query.columns.flag.isNull)
```

**len**

Create length(column) expression

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.custname.len)
```

**lower**

Create lower(column) expression

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.custname.lower)
```

**max**

Create an aggregate expression.

**Returns**[QBAggregate](#)**Sample**

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.groupBy.addPk() // have to group by on pk when using having-conditions in (foundset) pk queries
.root.having.add(query.joins.orders_to_order_details.columns.quantity.count.max(10))
foundset.loadRecords(query)
```

**min**

Create an aggregate expression.

**Returns**[QBAggregate](#)**Sample**

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.groupBy.addPk() // have to group by on pk when using having-conditions in (foundset) pk queries
.root.having.add(query.joins.orders_to_order_details.columns.quantity.count.min(10))
foundset.loadRecords(query)
```

**minute**

Extract minute from date

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.mydatecol.minute)
```

**month**

Extract month from date

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.mydatecol.month)
```

**not**

Create a negated condition.

**Returns**[QBColumn](#)**Sample**

```
query.where.add(query.columns.flag.not.eq(1))
```

**parent**

Get query builder parent table clause, this may be a query or a join clause.

**Returns**[QBTableClause](#)

**Sample**

```
/** @type {QBSelect<db:/example_data/person>} */
var query = databaseManager.createSelect('db:/example_data/person')
query.where.add(query.joins.person_to_parent.joins.person_to_parent.columns.name.eq('john'))
foundset.loadRecords(query)
```

**root**

Get query builder parent.

**Returns**

[QBSelect](#)

**Sample**

```
/** @type {QBSelect<db:/example_data/order_details>} */
var subquery = databaseManager.createSelect('db:/example_data/order_details')

/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.where.add(query
    .or
        .add(query.columns.order_id.not.isin([1, 2, 3]))
        .add(query.exists(
            subquery.where.add(subquery.columns.orderid.eq(query.columns.
order_id)).root
        )))
foundset.loadRecords(query)
```

**round**

Create round(column) expression

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.mycol.round)
```

**second**

Extract second from date

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.mydatecol.second)
```

**sqrt**

Create sqrt(column) expression

**Returns**

[QBFunction](#)

**Sample**

```
query.result.add(query.columns.custname.sqrt)
```

**sum**

Create an aggregate expression.

**Returns**[QBAggregate](#)**Sample**

```
/** @type {QBSelect<db:/example_data/orders>} */
var query = databaseManager.createSelect('db:/example_data/orders')
query.groupBy.addPk() // have to group by on pk when using having-conditions in (foundset) pk queries
.root.having.add(query.joins.orders_to_order_details.columns.quantity.count.sum(10))
foundset.loadRecords(query)
```

**trim**

Create trim(column) expression

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.custname.trim)
```

**upper**

Create upper(column) expression

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.custname.upper)
```

**year**

Extract year from date

**Returns**[QBFunction](#)**Sample**

```
query.result.add(query.columns.mydatecol.year)
```

**Method Details****between**[QBCondition](#) **between** (value1, value2)

Compare column to a range of 2 values or other columns.

**Parameters**

```
{Object} value1
{Object} value2
```

**Returns**[QBCondition](#)**Sample**

```
query.where.add(query.columns.flag.between(0, 5))
```

**cast**[QBFunction](#) **cast** (type)

Create cast(column, type) expression

---

**Parameters**

{String} type - string type, see QUERY\_COLUMN\_TYPES

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.cast(QUERY_COLUMN_TYPES.TYPE_INTEGER))
```

**concat**

QBFunction **concat** (arg)

Concatenate with value

**Parameters**

{Object} arg - value to concatenate with

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.firstname.concat(' ').concat(query.columns.lastname))
```

**divide**

QBFunction **divide** (arg)

Divide by value

**Parameters**

{Object} arg - nr to divide by

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.divide(2))
```

**eq**

QBCondition **eq** (value)

Compare column with a value or another column.

Operator: equals

**Parameters**

{Object} value

**Returns**

QBCondition

**Sample**

```
query.where.add(query.columns.flag.eq(1))
```

**ge**

QBCondition **ge** (value)

Compare column with a value or another column.

Operator: greaterThanOrEqualTo

**Parameters**

{Object} value

**Returns**

QBCondition

---

**Sample**

```
query.where.add(query.columns.flag.ge(2))
```

**gt**

**QBCondition** **gt** (value)

Compare column with a value or another column.  
Operator: greaterThan

**Parameters**

{Object} value

**Returns**

**QBCondition**

**Sample**

```
query.where.add(query.columns.flag.gt(0))
```

**isin**

**QBCondition** **isin** (query)

Compare column with subquery result.

**Parameters**

{QBPart} query - subquery

**Returns**

**QBCondition**

**Sample**

```
query.where.add(query.columns.flag.isin(query2))
```

**isin**

**QBCondition** **isin** (values)

Compare column with values.

**Parameters**

{Object[]} values - array of values

**Returns**

**QBCondition**

**Sample**

```
query.where.add(query.columns.flag.isin([1, 5, 99]))
```

**le**

**QBCondition** **le** (value)

Compare column with a value or another column.  
Operator: lessThanOrEqual

**Parameters**

{Object} value

**Returns**

**QBCondition**

**Sample**

```
query.where.add(query.columns.flag.le(2))
```

**like**

**QBCondition** **like** (pattern)

---

Compare column with a value or another column.  
Operator: like

**Parameters**

{String} pattern - the string value of the pattern

**Returns**

QBCondition

**Sample**

```
query.where.add(query.columns.companyname.like('Serv%'))
```

**like**

QBCondition **like** (pattern, escape)

Compare column with a value or another column.  
Operator: like, with escape character

**Parameters**

{String} pattern - the string value of the pattern

{String} escape - the escape char

**Returns**

QBCondition

**Sample**

```
query.where.add(query.columns.companyname.like('X_%', '_'))
```

**locate**

QBFunction **locate** (arg)

Create locate(arg) expression

**Parameters**

{Object} arg - string to locate

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.locate('sample'))
```

**locate**

QBFunction **locate** (arg, start)

Create locate(arg, start) expression

**Parameters**

{Object} arg - string to locate

{Number} start - start pos

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.locate('sample', 5))
```

**lt**

QBCondition **lt** (value)

Compare column with a value or another column.  
Operator: lessThan

**Parameters**

{Object} value

**Returns**

QBCondition

---

**Sample**

```
query.where.add(query.columns.flag.lt(99))
```

**minus****QBFunction** **minus** (arg)

Subtract value

**Parameters****{Object}** arg - nr to subtract**Returns****QBFunction****Sample**

```
query.result.add(query.columns.mycol.minus(2))
```

**mod****QBFunction** **mod** (arg)

Create mod(arg) expression

**Parameters****{Object}** arg - mod arg**Returns****QBFunction****Sample**

```
query.result.add(query.columns.mycol.mod(2))
```

**multiply****QBFunction** **multiply** (arg)

Multiply with value

**Parameters****{Object}** arg - nr to multiply with**Returns****QBFunction****Sample**

```
query.result.add(query.columns.mycol.multiply(2))
```

**nullif****QBFunction** **nullif** (arg)

Create nullif(arg) expression

**Parameters****{Object}** arg - object to compare**Returns****QBFunction****Sample**

```
query.result.add(query.columns.mycol.nullif('none'))
```

**plus****QBFunction** **plus** (arg)

Add up value

---

**Parameters**

{Object} arg - nr to add

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.plus(2))
```

**substring**

QBFunction **substring** (pos)

Create substring(pos) expression

**Parameters**

{Number} pos

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.substring(3))
```

**substring**

QBFunction **substring** (pos, len)

Create substring(pos, len) expression

**Parameters**

{Number} pos

{Number} len

**Returns**

QBFunction

**Sample**

```
query.result.add(query.columns.mycol.substring(3, 2))
```