# QBFunction

## Property Summary

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

## Method Summary

| | |
|---|---|
| QBCondition | between(value1, value2)<br>Compare column to a range of 2 values or other columns. |
| QBFunction | cast(type)<br>Create cast(column, type) expression |
| QBFunction | concat(arg)<br>Concatename with value |

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

Concatename with value

**Parameters**

{Object} arg - valeu 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: greaterThanOrEqual

**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))
```