

Package: sqlq (via r-universe)

May 15, 2026

Title 'SQL' Query Builder

Version 1.0.1

Maintainer Pierrick Roger <pierrick.roger@cea.fr>

Description Allows to build complex 'SQL' (Structured Query Language) queries dynamically. Classes and/or factory functions are used to produce a syntax tree from which the final character string is generated. Strings and identifiers are automatically quoted using the right quotes, using either ANSI (American National Standards Institute) quoting or the quoting style of an existing database connector. Style can be configured to set uppercase/lowercase for keywords, remove unnecessary spaces, or omit optional keywords.

URL <https://gitlab.com/cnrgh/databases/r-sqlq>

BugReports <https://gitlab.com/cnrgh/databases/r-sqlq/-/issues>

Depends R (>= 4.1)

License AGPL-3

Encoding UTF-8

Suggests roxygen2, testthat, knitr, rmarkdown, covr, cyclocomp, lintr, lgr, RSQLite

Imports R6, chk, DBI

NeedsCompilation no

RoxygenNote 7.3.2

Collate 'Statement.R' 'Expr.R' 'ExprBetween.R' 'ExprComp.R'
'ExprBinOp.R' 'ExprCommOp.R' 'Token.R' 'TokenSymbol.R'
'TokenIdentifier.R' 'ExprField.R' 'ExprFieldDef.R'
'ExprIsNotNull.R' 'ExprIsNull.R' 'ExprList.R'
'ExprListFields.R' 'ExprListValues.R' 'ExprUnaryOp.R' 'utils.R'
'TokenValue.R' 'ExprValue.R' 'Query.R' 'QueryCreate.R'
'QueryDelete.R' 'QueryInsert.R' 'QuerySelect.R' 'StmtSet.R'
'StmtUpdate.R' 'QueryUpdate.R' 'StmtCreate.R' 'StmtDelete.R'
'StmtFrom.R' 'StmtInsert.R' 'StmtJoin.R' 'StmtLimit.R'

'StmtSelect.R' 'StmtSelectAll.R' 'StmtSelectFields.R'
 'StmtValues.R' 'StmtWhere.R' 'TokenEmpty.R' 'TokenKeyword.R'
 'factories.R' 'package.R' 'tokens.R'

VignetteBuilder knitr

Author Pierrick Roger [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8177-4873>)

Repository <https://pkrog.r-universe.dev>

Date/Publication 2025-09-16 07:40:02 UTC

RemoteUrl <https://github.com/cran/sqlq>

RemoteRef HEAD

RemoteSha 937108373858352b70d8170cfffac3a5bb9e26a1a

Contents

sqlq-package	3
apply_case	4
Expr	5
ExprBetween	5
ExprBinOp	6
ExprCommOp	8
ExprComp	9
ExprField	10
ExprFieldDef	12
ExprIsNotNull	13
ExprIsNull	14
ExprList	15
ExprListFields	16
ExprListValues	17
ExprUnaryOp	18
ExprValue	19
make_between	20
make_create_table	21
make_delete	21
make_fields	22
make_insert	23
make_join	23
make_row	24
make_rows	25
make_select	25
make_select_all	26
make_set	27
make_update	27
make_values	28
make_where	29
Query	29
QueryCreate	31

QueryDelete	32
QueryInsert	33
QuerySelect	34
QueryUpdate	35
quote_ids	36
quote_values	37
Statement	37
StmtCreate	38
StmtDelete	39
StmtFrom	40
StmtInsert	41
StmtJoin	42
StmtLimit	44
StmtSelect	45
StmtSelectAll	46
StmtSelectFields	47
StmtSet	48
StmtUpdate	50
StmtValues	51
StmtWhere	52
Token	53
TokenEmpty	54
TokenIdentifier	55
TokenKeyword	56
TokenSymbol	57
TokenValue	58

Index	59
--------------	-----------

sqlq-package	<i>sqlq: 'SQL' Query Builder</i>
--------------	----------------------------------

Description

Allows to build complex 'SQL' (Structured Query Language) queries dynamically. Classes and/or factory functions are used to produce a syntax tree from which the final character string is generated. Strings and identifiers are automatically quoted using the right quotes, using either ANSI (American National Standards Institute) quoting or the quoting style of an existing database connector. Style can be configured to set uppercase/lowercase for keywords, remove unnecessary spaces, or omit optional keywords.

Details

sqlq package.

sqlq simplifies the creation of SQL queries, and ensure identifiers and string values are correctly quoted.

Global options used by *sqlq*:

- `sqlq_always_quote`: If set to TRUE, token identifiers (table and column names) will always be quoted.
- `sqlq_conn`: Set the database connector to use for quoting identifiers and values. Default is `DBI::ANSI()`.
- `sqlq_omit_kwd`: If set to TRUE, optional SQL keywords (like INNER or OUTER) will be omitted.
- `sqlq_spaces`: If set to FALSE, try to avoid non-necessary spaces (e.g.: around operators or after a comma).
- `sqlq_uppercase`: If set to FALSE, SQL keywords and alphabetical operators (e.g.: OR, AND, ...) will be written in lowercase.

Author(s)

Maintainer: Pierrick Roger <pierrick.roger@cea.fr> ([ORCID](#))

See Also

[options](#).

Examples

```
options(sqlq_uppercase = FALSE)
```

`apply_case`

Put string in right case according to global option.

Description

If global option `sqlq_case` is set to "lower", put the string in lowercase, if it is set to "upper", put the string in uppercase. Otherwise the string is not changed.

Usage

```
apply_case(s)
```

Arguments

`s` The string whose case must be changed.

Value

The string in the right case.

Expr *Expression abstract class.*

Description

This abstract class represents an SQL expression.

Super class

`sqlq::Statement` -> Expr

Methods

Public methods:

- `Expr$clone()`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Expr$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# No example provided, as this class is abstract.
```

ExprBetween *This class represents an SQL BETWEEN expression.*

Description

This class represents an SQL BETWEEN expression.

This class represents an SQL BETWEEN expression.

Details

Used to generate SQL expression BETWEEN / AND.

Super classes

`sqlq::Statement` -> `sqlq::Expr` -> ExprBetween

Methods**Public methods:**

- [ExprBetween\\$new\(\)](#)
- [ExprBetween\\$getTokens\(\)](#)
- [ExprBetween\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprBetween$new(field, low, high)
```

Arguments:

`field` An ExprField instance representing the field to check.

`low` An ExprValue instance representing the lower bound.

`high` An ExprValue instance representing the upper bound.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprBetween$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprBetween$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate "i BETWEEN 1 AND 10":
ExprBetween$new(ExprField$new("i"), ExprValue$new(1L), ExprValue$new(10L))
```

ExprBinOp

This class represents an SQL binary operator.

Description

This class represents an SQL binary operator.

This class represents an SQL binary operator.

Details

Used to generate SQL expressions involving a binary operator like in "a / 10".

Super classes

```
sqlq::Statement -> sqlq::Expr -> sqlq::ExprComp -> ExprBinOp
```

Methods**Public methods:**

- `ExprBinOp$new()`
- `ExprBinOp$getTokens()`
- `ExprBinOp$clone()`

Method `new()`: Initializer.

Usage:

```
ExprBinOp$new(lexpr, op, rexpr, ...)
```

Arguments:

`lexpr` An Expr instance for the left part.

`op` The binary operator, as a string.

`rexpr` An Expr instance for the right part.

`...` Arguments to pass to parent class.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprBinOp$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprBinOp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate "a / 10":  
ExprBinOp$new(ExprField$new("a"), "/", ExprValue$new(10))
```

 ExprCommOp

This class represents an SQL logical operator.

Description

This class represents an SQL logical operator.

This class represents an SQL logical operator.

Details

Used to generate SQL expressions involving a commutative binary operator like in "a + 10 + b".

Super classes

`sqlq::Statement` -> `sqlq::Expr` -> `sqlq::ExprComp` -> `ExprCommOp`

Methods

Public methods:

- `ExprCommOp$new()`
- `ExprCommOp$add()`
- `ExprCommOp$nb_expr()`
- `ExprCommOp$getTokens()`
- `ExprCommOp$clone()`

Method `new()`: Initializer.

Usage:

`ExprCommOp$new(op, expr = NULL)`

Arguments:

`op` The logical operator, as a string.

`expr` A list of logical expressions.

Returns: Nothing.

Method `add()`: Add an SQL expression to the logical operator.

Usage:

`ExprCommOp$add(expr)`

Arguments:

`expr` A `Expr` instance.

Returns: Nothing.

Method `nb_expr()`: Returns the number of expressions.

Usage:

`ExprCommOp$nb_expr()`

Returns: The number of expressions in this logical operator.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprCommOp$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprCommOp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate "a + 10 + b":
ExprCommOp$new("+", list(ExprField$new("a"), ExprValue$new(10),
                        ExprField$new("b")))
```

ExprComp

Composed Expression class.

Description

Composed Expression class.

Composed Expression class.

Details

This abstract class is used as a parent class for ExprBinOp and ExprCommOp.

Super classes

```
sqlq::Statement -> sqlq::Expr -> ExprComp
```

Methods

Public methods:

- [ExprComp\\$new\(\)](#)
- [ExprComp\\$enableParenthesis\(\)](#)
- [ExprComp\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprComp$new(paren = TRUE)
```

Arguments:

paren Set to TRUE to enable parenthesis around the expression.

Returns: Nothing.

Method enableParenthesis(): Disable parenthesis around expression.

Usage:

```
ExprComp$enableParenthesis(enabled)
```

Arguments:

enabled Set to TRUE to enable parenthesis and FALSE to disable them.

Returns: Nothing.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ExprComp$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# No example provided, as this class is abstract.
```

ExprField

This class represents an SQL field.

Description

This class represents an SQL field.

This class represents an SQL field.

Details

Used to define a field to be used inside a SELECT or UPDATE statement.

Super classes

```
sqlq::Statement -> sqlq::Expr -> ExprField
```

Methods

Public methods:

- [ExprField\\$new\(\)](#)
- [ExprField\\$getTable\(\)](#)
- [ExprField\\$getTokens\(\)](#)
- [ExprField\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprField$new(name, tabl = NULL)
```

Arguments:

`name` The field name.

`tabl` The table name.

Returns: Nothing.

Method `getTable()`: Return the associated table.

Usage:

```
ExprField$getTable()
```

Returns: The associated table, as a character value, NA if no table is defined.

Method `getTokens()`: Generate the list of tokens representing this statement.

Usage:

```
ExprField$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprField$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate the reference to field "title" in table "books":  
ExprField$new("title", tabl="books")
```

ExprFieldDef	<i>Table field definition.</i>
--------------	--------------------------------

Description

Table field definition.

Table field definition.

Details

Used to define a field when creating a table.

Super classes

`sqlq::Statement` -> `sqlq::Expr` -> ExprFieldDef

Methods

Public methods:

- `ExprFieldDef$new()`
- `ExprFieldDef$getTokens()`
- `ExprFieldDef$clone()`

Method `new()`: Initializer.

Usage:

```
ExprFieldDef$new(name, type, primary = FALSE, nullable = TRUE)
```

Arguments:

`name` The field name.

`type` The field's type (integer, date, varchar(...), ...).

`primary` Set to TRUE if the field is a PRIMARY KEY.

`nullable` Set to FALSE if the field does not accept NULL values.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprFieldDef$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprFieldDef$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate the definition of a field named "title":
ExprFieldDef$new("title", "TEXT", nullable = FALSE)
```

ExprIsNull

This class represents the IS NOT NULL test.

Description

This class represents the IS NOT NULL test.

This class represents the IS NOT NULL test.

Details

Used to test if a field is NOT NULL inside a WHERE clause.

Super classes

```
sqlq::Statement -> sqlq::Expr -> sqlq::ExprComp -> ExprIsNull
```

Methods**Public methods:**

- [ExprIsNull\\$new\(\)](#)
- [ExprIsNull\\$getTokens\(\)](#)
- [ExprIsNull\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprIsNull$new(expr, ...)
```

Arguments:

`expr` The Expr instance to test.

`...` Arguments to pass to parent class.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprIsNull$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprIsNull$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate a NOT NULL test:
ExprIsNull$new(ExprField$new("title"))
```

ExprIsNull

This class represents the IS NULL test.

Description

This class represents the IS NULL test.

This class represents the IS NULL test.

Details

Used to test if a field is NULL inside a WHERE clause.

Super classes

```
sqlq::Statement -> sqlq::Expr -> sqlq::ExprComp -> ExprIsNull
```

Methods**Public methods:**

- [ExprIsNull\\$new\(\)](#)
- [ExprIsNull\\$getTokens\(\)](#)
- [ExprIsNull\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprIsNull$new(expr, ...)
```

Arguments:

`expr` The Expr instance to test.

`...` Arguments to pass to parent class.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprIsNull$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprIsNull$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate a NULL test:  
ExprIsNull$new(ExprField$new("title"))
```

ExprList	<i>This class represents an SQL list.</i>
----------	---

Description

This class represents an SQL list.

This class represents an SQL list.

Details

An abstract class to represent a list. Used by ExprListValues and ExprListFields.

Super classes

```
sqlq::Statement -> sqlq::Expr -> ExprList
```

Methods

Public methods:

- [ExprList\\$new\(\)](#)
- [ExprList\\$getTokens\(\)](#)
- [ExprList\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprList$new(expr)
```

Arguments:

expr A list of Expr instances.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprList$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprList$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# No example provided, as this class is abstract.
```

ExprListFields	<i>This class represents a list of fields.</i>
----------------	--

Description

This class represents a list of fields.

This class represents a list of fields.

Details

Used to define a list of ExprField instances for the INSERT query.

Super classes

```
sqlq::Statement -> sqlq::Expr -> sqlq::ExprList -> ExprListFields
```

Methods**Public methods:**

- [ExprListFields\\$new\(\)](#)
- [ExprListFields\\$clone\(\)](#)

Method new(): Initializer.

Usage:

```
ExprListFields$new(fields)
```

Arguments:

fields A list of ExprField instances.

Returns: Nothing.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ExprListFields$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# To generate the list of fields "id", "title", "year":
ExprListFields$new(list(ExprField$new("id"),
                       ExprField$new("title"),
                       ExprField$new("year")))
```

ExprListValues *This class represents a list of values.*

Description

This class represents a list of values.

This class represents a list of values.

Details

Used to define a list of ExprValue instances for the INSERT query.

Super classes

`sqlq::Statement -> sqlq::Expr -> sqlq::ExprList -> ExprListValues`

Methods

Public methods:

- `ExprListValues$new()`
- `ExprListValues$clone()`

Method `new()`: Initializer.

Usage:

```
ExprListValues$new(values)
```

Arguments:

values A list of ExprValue instances.

Returns: Nothing.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprListValues$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# To generate the list of values 1234, "The River", "1965":
ExprListValues$new(list(ExprValue$new(1234),
                       ExprValue$new("The River"),
                       ExprValue$new(1965)))
```

ExprUnaryOp

This class represents an SQL unary operator.

Description

This class represents an SQL unary operator.

This class represents an SQL unary operator.

Details

Used to generate SQL expressions involving an unary operator like in "NOT flag".

Super classes

`sqlq::Statement` -> `sqlq::Expr` -> `sqlq::ExprComp` -> ExprUnaryOp

Methods

Public methods:

- `ExprUnaryOp$new()`
- `ExprUnaryOp$getTokens()`
- `ExprUnaryOp$clone()`

Method `new()`: Initializer.

Usage:

`ExprUnaryOp$new(op, expr, ...)`

Arguments:

`op` The unary operator, as a string.

`expr` An Expr instance.

`...` Arguments to pass to parent class.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

`ExprUnaryOp$getTokens()`

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`ExprUnaryOp$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# To generate "NOT flag":  
ExprUnaryOp$new("not", ExprField$new("flag"))
```

ExprValue	<i>This class represents an SQL value.</i>
-----------	--

Description

This class represents an SQL value.

This class represents an SQL value.

Details

Used to represent an SQL value.

Super classes

```
sqlq::Statement -> sqlq::Expr -> ExprValue
```

Methods

Public methods:

- [ExprValue\\$new\(\)](#)
- [ExprValue\\$getTokens\(\)](#)
- [ExprValue\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
ExprValue$new(value)
```

Arguments:

value The value.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
ExprValue$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ExprValue$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# To generate the integer value 30:  
ExprValue$new(30L)  
  
# To generate the string value "abcd":  
ExprValue$new("abcd")
```

make_between	Create a <i>BETWEEN</i> expression.
--------------	-------------------------------------

Description

Create an ExprBetween instance.

Usage

```
make_between(field, low, high)
```

Arguments

field	A character value or an ExprField instance representing the field to check.
low	An atomic single value or an ExprValue instance representing the lower bound.
high	An atomic single value of an ExprValue instance representing the upper bound.

Value

An instance of ExprBetween.

Examples

```
# To generate a BETWEEN expression checking if the "year" field is between  
# 1990 and 2000:  
between <- make_between("year", 1990, 2000)
```

make_create_table *Create an SQL CREATE TABLE query.*

Description

Create a QueryCreate instance.

Usage

```
make_create_table(tabl, fields_def)
```

Arguments

tabl	Name of the new table
fields_def	An list of ExprFieldDef instances.

Value

An instance of QueryCreate.

Examples

```
# To generate the CREATE query for creating a simple table for listing books:
fields_def <- list(ExprFieldDef$new('id', 'integer', primary=TRUE),
                  ExprFieldDef$new('title', 'varchar(200)', nullable=FALSE),
                  ExprFieldDef$new('author', 'varchar(80)', nullable=FALSE))
create <- make_create_table(tabl = 'books', fields_def = fields_def)
```

make_delete *Create an SQL DELETE FROM query.*

Description

Create a QueryDelete instance.

Usage

```
make_delete(tabl, where = NULL)
```

Arguments

tabl	Name of the new table
where	Set a StmtWhere instance to add a where clause.

Value

An instance of QueryDelete.

Examples

```
# Create a simple DELETE query for deleting some old books:
where <- StmtWhere$new(ExprBinOp$new(
  ExprField$new("year"), "<",
  ExprValue$new(2015)
))
delete <- make_delete(tabl = "books", where = where)
```

make_fields

Create a list of table fields.

Description

Create an ExprListFields instance.

Usage

```
make_fields(fields)
```

Arguments

fields A character vector containing field names.

Value

An instance of ExprListFields.

Examples

```
# To generate a list of fields:
fields <- make_fields(c('author', 'title', 'year'))
```

make_insert	<i>Create an SQL INSERT INTO query.</i>
-------------	---

Description

Create a QueryInsert instance.

Usage

```
make_insert(tabl, fields, values)
```

Arguments

tabl	A table name.
fields	A character vector containing field names.
values	A list of lists/vectors of values, each representing a row to insert.

Value

An instance of QueryInsert.

Examples

```
# To generate a simple INSERT query:
values <- list(list('John Smith', 'Memories', 1999),
              list('Barbara', 'My Life', 2010))
insert <- make_insert(tabl = 'books', fields = c('author', 'title', 'year'),
                     values = values)
```

make_join	<i>Create a SQL JOIN statement.</i>
-----------	-------------------------------------

Description

Create a StmtJoin instance.

Usage

```
make_join(
  field1,
  table1,
  field2,
  table2 = NULL,
  type = c("inner", "left", "right", "full")
)
```

Arguments

field1	The first field on which to join.
table1	The table name of the first field.
field2	The second field on which to join.
table2	The table name of the second field (optional).
type	The type of join to perform. One of "inner", "left", "right", or "full". Defaults to "inner".

Value

An instance of StmtJoin.

Examples

```
# To generate a JOIN statement joining the "author_id" field of the "books"
# table with the "id" field of the "authors" table:
join <- make_join("author_id", "books", "id", "authors")
```

make_row

Create a list of SQL values.

Description

Create an ExprListValues instance using a list. Useful when building an SQL list of values of mixed types, to use for instance with INSERT statement to define the row of values to insert.

Usage

```
make_row(values)
```

Arguments

values	A list/vector containing values.
--------	----------------------------------

Value

An instance of ExprListValues.

Examples

```
# To generate a list of values:
row <- make_row(list('John Smith', 'Memories', 1999))
```

make_rows	<i>Create a list of rows of values</i>
-----------	--

Description

Create a StmtValues instance.

Usage

```
make_rows(values)
```

Arguments

values A list of lists/vectors of values, each representing a row.

Value

An instance of StmtValues.

Examples

```
# To generate a VALUES statement with two rows:
rows <- make_rows(list(list('John Smith', 'Memories', 1999),
                       list('Barbara', 'My Life', 2010)))
```

make_select	<i>Create an SQL SELECT query.</i>
-------------	------------------------------------

Description

Create a QuerySelectFields instance to select a set of fields. The table name and the list of fields are the only required parameters.

Usage

```
make_select(
  tabl,
  fields,
  distinct = FALSE,
  limit = NULL,
  where = NULL,
  join = NULL
)
```

Arguments

tabl	A table name.
fields	A character vector containing field names or a list of ExprField objects.
distinct	If set to TRUE, add the distinct keyword.
limit	Add a limit (integer value) to the number of records returned.
where	Set a StmtWhere instance to add a where clause.
join	Set a StmtJoin instance to add a join clause.

Value

A SelectQuery instance.

Examples

```
# Here is a simple SELECT query:
make_select("books", fields = c("title", "author"))
```

make_select_all	<i>Create an SQL SELECT query for all fields.</i>
-----------------	---

Description

Create a QuerySelectAll instance (i.e.: select *) to retrieve all fields of a table.

Usage

```
make_select_all(
  tabl,
  distinct = FALSE,
  limit = NULL,
  where = NULL,
  join = NULL
)
```

Arguments

tabl	A table name.
distinct	If set to TRUE, add the distinct keyword.
limit	Add a limit (integer value) to the number of records returned.
where	Set a StmtWhere instance to add a where clause.
join	Set a StmtJoin instance to add a join clause.

Value

A instance of QuerySelect.

Examples

```
# Here is a simple SELECT * query:  
make_select_all("books")
```

make_set	<i>Create an SQL SET statement.</i>
----------	-------------------------------------

Description

Create a StmtSet instance.

Usage

```
make_set(...)
```

Arguments

... Named arguments, each representing a field name and its value.

Value

An instance of StmtSet.

Examples

```
# To generate a SET statement for setting the "price" and "old" fields:  
set <- make_set(price = 9.50, old = TRUE)
```

make_update	<i>Create an SQL UPDATE query.</i>
-------------	------------------------------------

Description

Create a QueryUpdate instance.

Usage

```
make_update(tbl, set, where = NULL)
```

Arguments

tbl	A table name.
set	A StmtSet instance containing the fields to update.
where	A StmtWhere instance to add a where clause (optional).

Value

An instance of QueryUpdate.

Examples

```
# Generate a simple update query:
where <- StmtWhere$new(ExprBinOp$new(
  ExprField$new("year"), "<",
  ExprValue$new(2010)
))
set <- make_set(price = 9.50, old = TRUE)
update <- make_update('books', set = set, where = where)$toString()
```

make_values

Create a list of SQL values.

Description

Create an ExprListValues instance using a vector. Useful when building an SQL list of values of identical type, to use with the IN operator.

Usage

```
make_values(values)
```

Arguments

values	A list/vector containing values.
--------	----------------------------------

Value

An instance of ExprListValues.

Examples

```
# To generate a list of values from a vector:
values <- make_values(c(1999, 2012, 2014))
```

make_where	<i>Create a WHERE clause.</i>
------------	-------------------------------

Description

Create a StmtWhere instance.

Usage

```
make_where(cond)
```

Arguments

cond An Expr instance representing the condition for the WHERE clause.

Value

An instance of StmtWhere.

Examples

```
# To generate a WHERE clause checking if the "year" field is greater than
# 2000:
where <- make_where(ExprBinOp$new(ExprField$new("year"), ">",
                                ExprValue$new(2000)))
```

Query	<i>This class handles an SQL Query.</i>
-------	---

Description

This class handles an SQL Query.

This class handles an SQL Query.

Details

This class represents an SQL query.

Methods

Public methods:

- [Query\\$new\(\)](#)
- [Query\\$add\(\)](#)
- [Query\\$toString\(\)](#)
- [Query\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
Query$new(stmts)
```

Arguments:

`stmts` A character vector of statement class names. It describes the accepted statements and their order, using wildcards to indicate if a statement is optional, or if it is allowed to occur multiple times. Example: `c("Select", "From", "Join*", "Where?", "Limit?")`

Returns: Nothing.

Method `add()`: Add a statement.

Usage:

```
Query$add(stmt)
```

Arguments:

`stmt` The statement to add.

Returns: Nothing.

Method `toString()`: Generates the string representation of this query.

Usage:

```
Query$toString()
```

Returns: A string containing the full SQL query.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Query$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# No example provided, as this class is abstract.
```

QueryCreate	<i>Create query.</i>
-------------	----------------------

Description

Create query.

Create query.

Details

This class represents an SQL CREATE TABLE query. See the function `make_create_table()` to create more easily a QueryCreate object.

Super class

[sqlq::Query](#) -> QueryCreate

Methods

Public methods:

- [QueryCreate\\$new\(\)](#)
- [QueryCreate\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
QueryCreate$new(create)
```

Arguments:

`create` A StmtCreate instance.

Returns: Nothing.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
QueryCreate$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[make_create_table](#)

Examples

```
# To generate the CREATE query for creating a simple table for listing books:
fields_def <- list(ExprFieldDef$new('id', 'integer', primary=TRUE),
                  ExprFieldDef$new('title', 'varchar(200)', nullable=FALSE),
                  ExprFieldDef$new('author', 'varchar(80)', nullable=FALSE))
create <- QueryCreate$new(StmtCreate$new(tabl = 'books',
                                         fields_def = fields_def))
```

QueryDelete

Delete query.

Description

Delete query.

Delete query.

Details

This class represents an SQL SELECT query. See the function `make_delete()` to create more easily a QueryDelete object.

Super class

`sqlq::Query` -> QueryDelete

Methods**Public methods:**

- `QueryDelete$new()`
- `QueryDelete$clone()`

Method `new()`: Initializer.

Usage:

```
QueryDelete$new(delete)
```

Arguments:

`delete` A StmtDelete instance.

Returns: Nothing.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
QueryDelete$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also[make_delete](#)**Examples**

```
# Create a simple DELETE query for deleting some old books:
where <- StmtWhere$new(ExprBinOp$new(
  ExprField$new("year"), "<",
  ExprValue$new(2015)
))
delete <- QueryDelete$new(StmtDelete$new('books'))
delete$add(where)
```

QueryInsert

Insert query.

Description

Insert query.

Insert query.

Details

This class represents an SQL SELECT query. See the `make_insert()` factory function to create more easily an INSERT query object.

Super class

`sqlq::Query` -> QueryInsert

Methods**Public methods:**

- `QueryInsert$new()`
- `QueryInsert$clone()`

Method `new()`: Initializer.

Usage:

```
QueryInsert$new(insert, values)
```

Arguments:

`insert` A StmtInsert instance.

`values` A StmtValues instance.

Returns: Nothing.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
QueryInsert$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

[make_insert](#)

Examples

```
# To generate a simple INSERT query:
fields <- c('author', 'title', 'year')
insert <- StmtInsert$new(tabl = 'books', fields = make_fields(fields))
values <- make_rows(list(list('John Smith', 'Memories', 1999),
                          list('Barbara', 'My Life', 2010)))
insert <- QueryInsert$new(insert = insert, values = values)
```

QuerySelect

Class for the SELECT query.

Description

Class for the SELECT query.

Class for the SELECT query.

Details

This class represents an SQL SELECT query. See `make_select()` and `make_select_all()` factory functions to create more easily a SELECT query.

Super class

`sqlq::Query` -> QuerySelect

Methods**Public methods:**

- `QuerySelect$new()`
- `QuerySelect$clone()`

Method `new()`: Initializer.

Usage:

```
QuerySelect$new(select, from)
```

Arguments:

select A StmtSelect instance.
 from A StmtFrom instance.

Returns: Nothing.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
QuerySelect$new(clone(deep = FALSE))
```

Arguments:

deep Whether to make a deep clone.

See Also

[make_select](#), [make_select_all](#)

Examples

```
# Here is a simple SELECT * query:
select <- QuerySelect$new(select = StmtSelectAll$new(),
                          from = StmtFrom$new("books"))
```

QueryUpdate

Update Query.

Description

Update Query.

Update Query.

Details

This class represents an SQL UPDATE query. See the `make_update()` factory function to create more easily an UPDATE query object.

Super class

`sqlq::Query` -> QueryUpdate

Methods

Public methods:

- [QueryUpdate\\$new\(\)](#)
- [QueryUpdate\\$clone\(\)](#)

Method new(): Initializer.

Usage:

```
QueryUpdate$new(up, set)
```

Arguments:

up A StmtUpdate instance.

set A StmtSet instance.

Returns: Nothing.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
QueryUpdate$new(clone(deep = FALSE))
```

Arguments:

deep Whether to make a deep clone.

See Also

[make_update](#)

Examples

```
# To generate a simple UPDATE query:
where <- StmtWhere$new(ExprBinOp$new(
  ExprField$new("year"), "<",
  ExprValue$new(2010)
))
set <- make_set(price = 9.50, old = TRUE)
update <- QueryUpdate$new(StmtUpdate$new('books'), set = set)
update$add(where)
```

quote_ids

Quote identifiers (e.g.: table names or field names) for SQL queries.

Description

Identifiers are quoted only if it contains at least one non-alphanumeric character.

Usage

```
quote_ids(ids)
```

Arguments

ids Character vector of identifiers to quote.

Value

A character vector containing the same identifiers, quoted if necessary.

quote_values	<i>Quote character values for SQL queries.</i>
--------------	--

Description

Quote character values inside a vector or list. If other values are found inside the list or vector, they are converted to character values.

Usage

```
quote_values(values)
```

Arguments

values	Vector or list of values.
--------	---------------------------

Value

A character vector containing the same values, converted. All character values are quoted.

Statement	<i>Abstract class that represents an SQL statement.</i>
-----------	---

Description

Abstract class that represents an SQL statement.

Abstract class that represents an SQL statement.

Details

This abstract class represents an SQL statement (FROM, SELECT, WHERE, ...). Note that expressions (Expr class) are a particular type of Statement in sqlq.

Methods**Public methods:**

- [Statement\\$new\(\)](#)
- [Statement\\$getTokens\(\)](#)
- [Statement\\$string\(\)](#)
- [Statement\\$clone\(\)](#)

Method new(): Initializer

Usage:

```
Statement$new()
```

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

`Statement$getTokens()`

Returns: A list of Token objects.

Method `toString()`: Generates the string representation of this statement.

Usage:

`Statement$toString()`

Returns: A string containing the SQL expression.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Statement$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# No example provided, as this class is abstract.
```

StmtCreate	<i>CREATE TABLE statement.</i>
------------	--------------------------------

Description

CREATE TABLE statement.

CREATE TABLE statement.

Super class

[sqlq::Statement](#) -> StmtCreate

Methods

Public methods:

- [StmtCreate\\$new\(\)](#)
- [StmtCreate\\$getTokens\(\)](#)
- [StmtCreate\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
StmtCreate$new(tabl, fields_def)
```

Arguments:

tabl A table name.

fields_def An instance of ExprListFields

Returns: Nothing.

Method getTokens(): Generates the list of tokens representing this statement.

Usage:

```
StmtCreate$getTokens()
```

Returns: A list of Token objects.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
StmtCreate$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# To generate a simple CREATE TABLE statement:
fields_def <- list(ExprFieldDef$new('id', 'integer', primary=TRUE),
                  ExprFieldDef$new('title', 'varchar(200)', nullable=FALSE),
                  ExprFieldDef$new('author', 'varchar(80)', nullable=FALSE))
StmtCreate$new(tabl = 'books', fields_def = fields_def)
```

StmtDelete

DELETE FROM statement.

Description

DELETE FROM statement.

DELETE FROM statement.

Super class

[sqlq::Statement](#) -> StmtDelete

Methods**Public methods:**

- [StmtDelete\\$new\(\)](#)
- [StmtDelete\\$getTokens\(\)](#)
- [StmtDelete\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

`StmtDelete$new(tabl)`

Arguments:

`tabl` A table name.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

`StmtDelete$getTokens()`

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`StmtDelete$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Simple DELETE statement:
StmtDelete$new('books')
```

StmtFrom

SQL From statement.

Description

SQL From statement.

SQL From statement.

Super class

[sqlq::Statement](#) -> StmtFrom

Methods

Public methods:

- [StmtFrom\\$new\(\)](#)
- [StmtFrom\\$getTokens\(\)](#)
- [StmtFrom\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
StmtFrom$new(tabl)
```

Arguments:

tabl A table name.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtFrom$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtFrom$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Example a FROM statement:  
StmtFrom$new('books')
```

StmtInsert

INSERT INTO statement.

Description

INSERT INTO statement.

INSERT INTO statement.

Super class

[sqlq::Statement](#) -> StmtInsert

Methods

Public methods:

- [StmtInsert\\$new\(\)](#)
- [StmtInsert\\$getTokens\(\)](#)
- [StmtInsert\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
StmtInsert$new(tabl, fields)
```

Arguments:

`tabl` A table name.

`fields` An instance of `ExprListFields`

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtInsert$getTokens()
```

Returns: A list of `Token` objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtInsert$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Simple INSERT statement:
fields <- c('author', 'title', 'year')
insert <- StmtInsert$new(tabl = 'books', fields = make_fields(fields))
```

StmtJoin

SQL JOIN statement.

Description

SQL JOIN statement.

SQL JOIN statement.

Details

This class represents a `SQL JOIN` statement. It requires two fields on which to join, and the type of join to perform (inner, left, right, or full). The table on which to join is determined by looking at the two fields in order and using the first table name available.

Super class

`sqlq::Statement` -> StmtJoin

Methods**Public methods:**

- `StmtJoin$new()`
- `StmtJoin$getTokens()`
- `StmtJoin$clone()`

Method `new()`: Initializer. To determine the table on which to join, we look at the both fields in order and use the first table name available.

Usage:

```
StmtJoin$new(field1, field2, type = c("inner", "left", "right", "full"))
```

Arguments:

field1 The first field on which to join.

field2 The second field on which to join.

type The type of join to perform. One of "inner", "left", "right", or "full". Defaults to "inner".

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtJoin$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtJoin$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create an inner join (default join type) between table 'foo' and table
# 'bar':
join <- StmtJoin$new(ExprField$new("id", "foo"),
                    ExprField$new("foo_id", "bar"))

# Create a left join between table 'foo' and table 'bar':
join <- StmtJoin$new(ExprField$new("id", "foo"),
                    ExprField$new("foo_id", "bar"),
                    type = "left")
```

StmtLimit

LIMIT statement.

Description

LIMIT statement.

LIMIT statement.

Details

This class represents a SQL LIMIT statement. It requires a single integer limit value.

Super class

[sqlq::Statement](#) -> StmtLimit

Methods

Public methods:

- [StmtLimit\\$new\(\)](#)
- [StmtLimit\\$getTokens\(\)](#)
- [StmtLimit\\$clone\(\)](#)

Method `new()`: Initializer

Usage:

`StmtLimit$new(limit)`

Arguments:

`limit` The integer limit.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

`StmtLimit$getTokens()`

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`StmtLimit$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a LIMIT statement with a limit of 10:
limit <- StmtLimit$new(10L)

# Use the created LIMIT statement inside a SELECT query:
query <- QuerySelect$new(StmtSelectAll$new(),
                        from = StmtFrom$new("books"))
query$add(limit)
```

StmtSelect

Abstract SELECT statement.

Description

Abstract SELECT statement.

Abstract SELECT statement.

Details

This is an abstract class representing a SQL SELECT statement. It is inherited by concrete classes StmtSelectAll and StmtSelectFields.

Super class

`sqlq::Statement` -> StmtSelect

Methods

Public methods:

- `StmtSelect$new()`
- `StmtSelect$clone()`

Method `new()`: Initializer

Usage:

```
StmtSelect$new(distinct = FALSE)
```

Arguments:

`distinct` Set to TRUE enable distinct keyword and remove duplicate results.

Returns: Nothing.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtSelect$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[StmtSelectAll](#), [StmtSelectFields](#)

Examples

```
# No example provided, as this class is abstract.
```

StmtSelectAll	<i>SELECT * statement.</i>
---------------	----------------------------

Description

SELECT * statement.

SELECT * statement.

Details

This class represents a SQL SELECT * statement. It can be used to select all fields from a table, with optional distinct keyword to remove duplicate results.

Super classes

[sqlq::Statement](#) -> [sqlq::StmtSelect](#) -> StmtSelectAll

Methods**Public methods:**

- [StmtSelectAll\\$new\(\)](#)
- [StmtSelectAll\\$getTokens\(\)](#)
- [StmtSelectAll\\$clone\(\)](#)

Method new(): Initializer

Usage:

```
StmtSelectAll$new(distinct = FALSE)
```

Arguments:

distinct Set to TRUE enable distinct keyword and remove duplicate results.

Returns: Nothing.

Method getTokens(): Generates the list of tokens representing this statement.

Usage:

```
StmtSelectAll$getTokens()
```

Returns: A list of Token objects.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
StmtSelectAll$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create a SELECT * statement:
select_all <- StmtSelectAll$new()

# Use the created SELECT * statement inside a SELECT query:
query <- QuerySelect$new(select = select_all,
                        from = StmtFrom$new("books"))

# Create a SELECT DISTINCT * statement:
select_distinct_all <- StmtSelectAll$new(distinct = TRUE)
```

StmtSelectFields	<i>SELECT fields statement.</i>
------------------	---------------------------------

Description

SELECT fields statement.

SELECT fields statement.

Details

This class represents a SQL SELECT statement with specific fields. It requires a list of ExprField instances representing the fields to select, with optional distinct keyword to remove duplicate results.

Super classes

```
sqlq::Statement -> sqlq::StmtSelect -> StmtSelectFields
```

Methods**Public methods:**

- [StmtSelectFields\\$new\(\)](#)
- [StmtSelectFields\\$getTokens\(\)](#)
- [StmtSelectFields\\$clone\(\)](#)

Method `new()`: Initializer

Usage:

```
StmtSelectFields$new(fields, distinct = FALSE)
```

Arguments:

`fields` A list of ExprField instances.

`distinct` Set to TRUE enable distinct keyword and remove duplicate results.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtSelectFields$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtSelectFields$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a SELECT statement with specific fields:
field1 <- ExprField$new("title", "books")
field2 <- ExprField$new("name", "authors")
select_fields <- StmtSelectFields$new(fields = list(field1, field2))

# Use the created SELECT statement inside a SELECT query:
query <- QuerySelect$new(select = select_fields,
                        from = StmtFrom$new("books"))
```

StmtSet

SET statement.

Description

SET statement.

SET statement.

Details

This class represents a SQL SET statement, used in UPDATE queries to set field values. It can hold one or more field/value pairs. The factory function `make_set()` can be used to create a SET statement more easily.

Super class

`sqlq::Statement` -> StmtSet

Methods

Public methods:

- [StmtSet\\$new\(\)](#)
- [StmtSet\\$add_field\(\)](#)
- [StmtSet\\$getTokens\(\)](#)
- [StmtSet\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
StmtSet$new()
```

Returns: Nothing.

Method `add_field()`: Add a field/value pair.

Usage:

```
StmtSet$add_field(field, value)
```

Arguments:

`field` The field, as an `ExprField` instance.

`value` The value to set, as an `Expr` instance.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtSet$getTokens()
```

Returns: A list of `Token` objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[make_set\(\)](#)

Examples

```
# Create a SET statement with a single field/value pair:
set_stmt <- StmtSet$new()
set_stmt$add_field(ExprField$new("price"), ExprValue$new(9.50))

# Use the created SET statement inside an UPDATE query:
query <- QueryUpdate$new(StmtUpdate$new("books"), set = set_stmt)
```

StmtUpdate	<i>UPDATE statement.</i>
------------	--------------------------

Description

UPDATE statement.

UPDATE statement.

Details

This class represents a SQL UPDATE statement. It requires a table name.

Super class

`sqlq::Statement` -> StmtUpdate

Methods

Public methods:

- `StmtUpdate$new()`
- `StmtUpdate$getTokens()`
- `StmtUpdate$clone()`

Method `new()`: Initializer.

Usage:

```
StmtUpdate$new(tabl)
```

Arguments:

tabl A table name.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtUpdate$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtUpdate$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create an UPDATE statement for table 'books':  
update <- StmtUpdate$new("books")
```

StmtValues	VALUES statement.
------------	-------------------

Description

VALUES statement.

VALUES statement.

Details

This class represents a SQL VALUES statement, used when inserting multiple rows.

Super class

[sqlq::Statement](#) -> StmtValues

Methods

Public methods:

- [StmtValues\\$new\(\)](#)
- [StmtValues\\$getTokens\(\)](#)
- [StmtValues\\$clone\(\)](#)

Method `new()`: Initializer.

Usage:

```
StmtValues$new(values)
```

Arguments:

values An instance of ExprListValues

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

```
StmtValues$getTokens()
```

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
StmtValues$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create a VALUES statement with two rows:
row1 <- ExprListValues$new(list(ExprValue$new("abc"), ExprValue$new(123)))
row2 <- ExprListValues$new(list(ExprValue$new("def"), ExprValue$new(456)))
values <- StmtValues$new(list(row1, row2))
```

 StmtWhere

SQL WHERE statement.

Description

SQL WHERE statement.

SQL WHERE statement.

Details

This class represents a SQL WHERE statement, used to filter results in SELECT, UPDATE, and DELETE statements.

Super class

`sqlq::Statement` -> StmtWhere

Methods**Public methods:**

- `StmtWhere$new()`
- `StmtWhere$getTokens()`
- `StmtWhere$clone()`

Method `new()`: Initializer.

Usage:

`StmtWhere$new(expr)`

Arguments:

`expr` The expression to evaluate.

Returns: Nothing.

Method `getTokens()`: Generates the list of tokens representing this statement.

Usage:

`StmtWhere$getTokens()`

Returns: A list of Token objects.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`StmtWhere$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a WHERE statement with a simple expression:
expr <- ExprBinOp$new(ExprValue$new("age"), ">=", ExprValue$new(18))
where <- StmtWhere$new(expr)

# Use the created WHERE statement inside a SELECT query:
query <- QuerySelect$new(StmtSelectAll$new(),
                        from = StmtFrom$new("users"))
query$add(where)
```

Token

*Abstract Token class.***Description**

Abstract Token class.

Abstract Token class.

Details

This is an abstract class representing a SQL token. It is inherited by concrete token classes such as `TokenValue` and `TokenIdentifier`.

Methods**Public methods:**

- [Token\\$string\(\)](#)
- [Token\\$clone\(\)](#)

Method `toString()`: Convert this object into a string.

Usage:

```
Token$string()
```

Returns: A character value.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Token$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# No example provided, as this class is abstract.
```

TokenEmpty

Empty token class.

Description

Empty token class.

Empty token class.

Details

This class represents an empty SQL token. It is used in situations where a token is required by the structure of the code, but no actual SQL code needs to be generated.

Super class

`sqlq::Token` -> TokenEmpty

Methods**Public methods:**

- `TokenEmpty$new()`
- `TokenEmpty$string()`
- `TokenEmpty$clone()`

Method `new()`: Initializer.

Usage:

`TokenEmpty$new()`

Returns: Nothing.

Method `toString()`: Converts into a string.

Usage:

`TokenEmpty$string()`

Returns: A string containing the SQL expression.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`TokenEmpty$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# No example since this class is not exported.
```

TokenIdentifier	<i>TokenIdentifier class.</i>
-----------------	-------------------------------

Description

TokenIdentifier class.

TokenIdentifier class.

Details

This class represents a SQL identifier token, such as a table or column name.

Super class

`sqlq:Token` -> TokenIdentifier

Methods

Public methods:

- `TokenIdentifier$new()`
- `TokenIdentifier$string()`
- `TokenIdentifier$clone()`

Method new(): Initializer.

Usage:

`TokenIdentifier$new(id)`

Arguments:

id The identifier.

Returns: Nothing.

Method toString(): Converts into a string.

Usage:

`TokenIdentifier$string()`

Returns: A string containing the SQL expression.

Method clone(): The objects of this class are cloneable with this method.

Usage:

`TokenIdentifier$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

Examples

```
# No example since this class is not exported.
```

TokenKeyword *TokenKeyword class.*

Description

TokenKeyword class.

TokenKeyword class.

Details

Represents an SQL keyword such as SELECT, FROM, WHERE, etc.

Super class

[sqlq:Token](#) -> TokenKeyword

Methods

Public methods:

- [TokenKeyword\\$new\(\)](#)
- [TokenKeyword\\$string\(\)](#)
- [TokenKeyword\\$clone\(\)](#)

Method new(): Initializer.

Usage:

TokenKeyword\$new(kwd)

Arguments:

kwd The keyword.

Returns: Nothing.

Method toString(): Converts into a string.

Usage:

TokenKeyword\$string()

Returns: A string containing the SQL expression.

Method clone(): The objects of this class are cloneable with this method.

Usage:

TokenKeyword\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Examples

```
# No example since this class is not exported.
```

TokenSymbol	<i>TokenSymbol class.</i>
-------------	---------------------------

Description

TokenSymbol class.

TokenSymbol class.

Details

Represents a SQL symbol such as *, +, -, /, =, <, >, etc.

Super class

[sqlq:Token](#) -> TokenSymbol

Methods

Public methods:

- [TokenSymbol\\$new\(\)](#)
- [TokenSymbol\\$string\(\)](#)
- [TokenSymbol\\$clone\(\)](#)

Method new(): Initializer.

Usage:

`TokenSymbol$new(symbol)`

Arguments:

symbol The symbol.

Returns: Nothing.

Method toString(): Converts into a string.

Usage:

`TokenSymbol$string()`

Returns: A string containing the SQL expression.

Method clone(): The objects of this class are cloneable with this method.

Usage:

`TokenSymbol$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

Examples

```
# No example since this class is not exported.
```

TokenValue	<i>Token value class.</i>
------------	---------------------------

Description

Token value class.

Token value class.

Details

Represents a SQL value such as a number or a string.

Super class

`sqlq::Token` -> TokenValue

Methods

Public methods:

- `TokenValue$new()`
- `TokenValue$string()`
- `TokenValue$clone()`

Method `new()`: Initializer.

Usage:

`TokenValue$new(value)`

Arguments:

value The value.

Returns: Nothing.

Method `toString()`: Converts into a string.

Usage:

`TokenValue$string()`

Returns: A string containing the SQL expression.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`TokenValue$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

Examples

No example since this class is not exported.

Index

apply_case, 4

Expr, 5
ExprBetween, 5
ExprBinOp, 6
ExprCommOp, 8
ExprComp, 9
ExprField, 10
ExprFieldDef, 12
ExprIsNotNull, 13
ExprIsNull, 14
ExprList, 15
ExprListFields, 16
ExprListValues, 17
ExprUnaryOp, 18
ExprValue, 19

make_between, 20
make_create_table, 21, 31
make_delete, 21, 33
make_fields, 22
make_insert, 23, 34
make_join, 23
make_row, 24
make_rows, 25
make_select, 25, 35
make_select_all, 26, 35
make_set, 27, 48, 49
make_update, 27, 36
make_values, 28
make_where, 29

options, 4

Query, 29
QueryCreate, 31
QueryDelete, 32
QueryInsert, 33
QuerySelect, 34
QueryUpdate, 35

quote_ids, 36
quote_values, 37

sqlq (sqlq-package), 3
sqlq-package, 3
sqlq::Expr, 5, 7–10, 12–19
sqlq::ExprComp, 7, 8, 13, 14, 18
sqlq::ExprList, 16, 17
sqlq::Query, 31–35
sqlq::Statement, 5, 7–10, 12–19, 38–41, 43–48, 50–52
sqlq::StmtSelect, 46, 47
sqlq::Token, 54–58
Statement, 37
StmtCreate, 38
StmtDelete, 39
StmtFrom, 40
StmtInsert, 41
StmtJoin, 42
StmtLimit, 44
StmtSelect, 45
StmtSelectAll, 46, 46
StmtSelectFields, 46, 47
StmtSet, 48
StmtUpdate, 50
StmtValues, 51
StmtWhere, 52

Token, 53
TokenEmpty, 54
TokenIdentifier, 55
TokenKeyword, 56
TokenSymbol, 57
TokenValue, 58