

Package ‘editbl’

July 9, 2026

Type Package

Version 1.3.2

Date 2026-07-09

Title 'DT' Extension for CRUD (Create, Read, Update, Delete)
Applications in 'shiny'

Maintainer Jasper Schelfhout <jasper.schelfhout@openanalytics.eu>

Description The core of this package is a function eDT() which enhances DT::datatable() such that it can be used to interactively modify data in 'shiny'. By the use of generic 'dplyr' methods it supports many types of data storage, with relational databases ('dbplyr') being the main use case.

License GPL-3

Copyright Open Analytics NV, 2023

Imports shiny, shinyjs, DT, tibble, dplyr, rlang, uuid, fontawesome
(>= 0.4.0)

Suggests testthat, dtplyr, data.table, vctrs, RSQLite, dbplyr, glue,
DBI, bit64, knitr, dm

URL <https://github.com/openanalytics/editbl>

BugReports <https://github.com/openanalytics/editbl/issues>

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Jasper Schelfhout [aut, cre],
Maxim Nazarov [rev],
Daan Seynaeve [rev],
Lennart Tuijnder [rev],
Saar Junius [aut]

Repository CRAN

Date/Publication 2026-07-09 16:30:02 UTC

Contents

connectDB	2
customButton	3
eDT	4
eDTOutput	10
e_rows_delete	11
e_rows_delete.default	13
e_rows_insert	14
e_rows_insert.default	15
e_rows_insert.dtplyr_step	17
e_rows_insert.tbl_dbi	18
e_rows_update	20
e_rows_update.data.frame	21
e_rows_update.default	23
e_rows_update.dtplyr_step	24
e_rows_update.tbl_dbi	26
foreignTbl	28
get_db_table_name	29
inputServer	30
inputServer.default	31
inputUI	31
inputUI.default	32
joinForeignTbl	33
rows_delete.dtplyr_step	34
runDemoApp	35
selectInputDT_Server	36
selectInputDT_UI	37
Index	39

connectDB	<i>Connect to a database.</i>
-----------	-------------------------------

Description

Connect to a database.

Usage

```
connectDB(  
  dbname = system.file("extdata", "chinook.sqlite", package = utils::packageName()),  
  drv = RSQLite::SQLite(),  
  ...  
)
```

Arguments

dbname	character(0)
drv	database driver
...	arguments passed to DBI::dbConnect

Details

Connects by default to a test SQLite database originally obtained here: [chinook_git](#)

Value

database connection

Examples

```
conn <- connectDB()
DBI::dbDisconnect(conn)
```

customButton	<i>Generate a custom button for eDT</i>
--------------	---

Description

Generate a custom button for [eDT](#)

Usage

```
customButton(id, label, icon = "", disabled = FALSE)
```

Arguments

id	character(1), namespaced id
label	character(1)
icon	shiny::icon
disabled	logical. Whether or not the button should start in a disabled state.

Details

Combines elements of shiny::actionButton and [datatable options](#)

Value

list to be used in eDT(options = list(buttons = xxx))

Author(s)

Jasper Schelfhout

Examples

```
if(interactive()){

  ui <- eDTOutput("data")
  server <- function(input,output,session){
    b <- customButton('print', label = 'print')
    eDT_result <- eDT(id = "data", mtcars, options = list(buttons = list("save", b)))
    observeEvent(input$print,{
      print(eDT_result$state())
    })
  }
  shinyApp(ui,server)
}
```

eDT

Create a modifieable datatable.

Description

Create a modifieable datatable.

Usage

```
eDT(
  data,
  options = list(dom = "Bftrlip", keys = TRUE, ordering = FALSE, autoFill = list(update =
    FALSE, focus = "focus"), buttons = list("add", "undo", "redo", "save")),
  class = "display",
  callback = NULL,
  rownames = FALSE,
  colnames = NULL,
  container,
  caption = NULL,
  filter = c("none", "bottom", "top"),
  escape = TRUE,
  style = "auto",
  width = NULL,
  height = NULL,
  elementId = NULL,
  fillContainer = getOption("DT.fillContainer", NULL),
  autoHideNavigation = getOption("DT.autoHideNavigation", NULL),
  selection = "none",
  extensions = c("KeyTable", "AutoFill", "Buttons"),
```

```

    plugins = NULL,
    editable = list(target = "cell"),
    id,
    keys = NULL,
    in_place = FALSE,
    format = function(x) {
      x
    },
    foreignTbIs = list(),
    columnOrder = c(),
    statusColor = c(insert = "#e6e6e6", update = "#32a6d3", delete = "#e52323"),
    inputUI = edittbl::inputUI,
    defaults = tibble(),
    cloneDefaults = tibble(),
    env = environment(),
    canEditRow = TRUE,
    canDeleteRow = TRUE,
    canCloneRow = TRUE,
    utilityColumns = NULL
  )

```

Arguments

data	tbl. When using non-standard tibble objects, install the required package implementing specific methods. (data.table: dtplyr, spark: sparklyr, generic database: dbplyr,...). While other tabular objects like data.frame are supported, explicitly casting to tbl is recommended.
options	a list of initialization options (see https://datatables.net/reference/option/); the character options wrapped in JS() will be treated as literal JavaScript code instead of normal character strings; you can also set options globally via options(DT.options = list(...)), and global options will be merged into this options argument if set
class	the CSS class(es) of the table; see https://datatables.net/manual/styling/classes
callback	the body of a JavaScript callback function with the argument table to be applied to the DataTables instance (i.e. table)
rownames	TRUE (show row names) or FALSE (hide row names) or a character vector of row names; by default, the row names are displayed in the first column of the table if exist (not NULL)
colnames	if missing, the column names of the data; otherwise it can be an unnamed character vector of names you want to show in the table header instead of the default data column names; alternatively, you can provide a <i>named</i> numeric or character vector of the form 'newName1' = i1, 'newName2' = i2 or c('newName1' = 'oldName1', 'newName2' = 'oldName2', ...), where newName is the new name you want to show in the table, and i or oldName is the index of the current column name

container	a sketch of the HTML table to be filled with data cells; by default, it is generated from <code>htmltools::tags\$table()</code> with a table header consisting of the column names of the data
caption	the table caption; a character vector or a tag object generated from <code>htmltools::tags\$caption()</code>
filter	whether/where to use column filters; none: no filters; bottom/top: put column filters at the bottom/top of the table; range sliders are used to filter numeric/date/time columns, select lists are used for factor columns, and text input boxes are used for character columns; if you want more control over the styles of filters, you can provide a named list to this argument; see Details for more
escape	whether to escape HTML entities in the table: TRUE means to escape the whole table, and FALSE means not to escape it; alternatively, you can specify numeric column indices or column names to indicate which columns to escape, e.g. 1:5 (the first 5 columns), <code>c(1, 3, 4)</code> , or <code>c(-1, -3)</code> (all columns except the first and third), or <code>c('Species', 'Sepal.Length')</code> ; since the row names take the first column to display, you should add the numeric column indices by one when using <code>rownames</code>
style	either 'auto', 'default', 'bootstrap', or 'bootstrap4'. If 'auto', and a <code>**bslib**</code> theme is currently active, then bootstrap styling is used in a way that "just works" for the active theme. Otherwise, DataTables 'default' styling is used. If set explicitly to 'bootstrap' or 'bootstrap4', one must take care to ensure Bootstrap's HTML dependencies (as well as Bootswatch themes, if desired) are included on the page. Note, when set explicitly, it's the user's responsibility to ensure that only one unique 'style' value is used on the same page, if multiple DT tables exist, as different styling resources may conflict with each other.
width, height	Width/Height in pixels (optional, defaults to automatic sizing)
elementId	An id for the widget (a random string by default).
fillContainer	TRUE to configure the table to automatically fill it's containing element. If the table can't fit fully into it's container then vertical and/or horizontal scrolling of the table cells will occur.
autoHideNavigation	TRUE to automatically hide navigational UI (only display the table body) when the number of total records is less than the page size. Note, it only works on the client-side processing mode and the 'pageLength' option should be provided explicitly.
selection	the row/column selection mode (single or multiple selection or disable selection) when a table widget is rendered in a Shiny app; alternatively, you can use a list of the form <code>list(mode = 'multiple', selected = c(1, 3, 8), target = 'row', selectable = c(-2, -3))</code> to pre-select rows and control the selectable range; the element target in the list can be 'column' to enable column selection, or 'row+column' to make it possible to select both rows and columns (click on the footer to select columns), or 'cell' to select cells. See details section for more info.
extensions	a character vector of the names of the DataTables extensions (https://datatables.net/extensions/index)

plugins	a character vector of the names of DataTables plug-ins (https://rstudio.github.io/DT/plugins.html). Note that only those plugins supported by the DT package can be used here. You can see the available plugins by calling <code>DT::available_plugins()</code>
editable	FALSE to disable the table editor, or TRUE (or "cell") to enable editing a single cell. Alternatively, you can set it to "row" to be able to edit a row, or "column" to edit a column, or "all" to edit all cells on the current page of the table. In all modes, start editing by doubleclicking on a cell. This argument can also be a list of the form <code>list(target = TARGET, disable = list(columns = INDICES))</code> , where TARGET can be "cell", "row", "column", or "all", and INDICES is an integer vector of column indices. Use the list form if you want to disable editing certain columns. You can also restrict the editing to accept only numbers by setting this argument to a list of the form <code>list(target = TARGET, numeric = INDICES)</code> where INDICES can be the vector of the indices of the columns for which you want to restrict the editing to numbers or "all" to restrict the editing to numbers for all columns. If you don't set numeric, then the editing is restricted to numbers for all numeric columns; set numeric = "none" to disable this behavior. It is also possible to edit the cells in text areas, which are useful for large contents. For that, set the editable argument to a list of the form <code>list(target = TARGET, area = INDICES)</code> where INDICES can be the vector of the indices of the columns for which you want the text areas, or "all" if you want the text areas for all columns. Of course, you can request the numeric editing for some columns and the text areas for some other columns by setting editable to a list of the form <code>list(target = TARGET, numeric = INDICES1, area = INDICES2)</code> . Finally, you can edit date cells with a calendar with <code>list(target = TARGET, date = INDICES)</code> ; the target columns must have the Date type. If you don't set date in the editable list, the editing with the calendar is automatically set for all Date columns.
id	character(1) module id
keys	character. Defaults to all columns under the assumption that at least every row is unique.
in_place	logical. Whether to modify the data object in place or to return a modified copy.
format	function accepting and returning a datatable
foreignTbIs	list. List of objects created by foreignTbIs
columnOrder	vector. Order of columns, original data and foreignTbIs combined. Defaults to no order specified.
statusColor	named character. Colors to indicate status of the row.
inputUI	function. UI function of a shiny module with at least arguments id data and ... #' elements with inputIds identical to one of the column names are used to update the data.
defaults	expression that evaluates to a tibble with (a subset of) columns of the data. It will be evaluated for each new row in the environment defined by 'env'. This allows for defaults like <code>Sys.time()</code> or <code>uuid::UUIDgenerate()</code> as well as dynamic inputs.

<code>cloneDefaults</code>	expression that evaluates to a tibble with (a subset of) columns of the data. It will be evaluated for each cloned row in the environment defined by <code>'env'</code> .
<code>env</code>	environment in which the server function is running. Should normally not be modified.
<code>canEditRow</code>	can be either of the following: • logical, e.g. TRUE or FALSE. In case of FALSE, buttons will not be visible except for new rows. • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.
<code>canDeleteRow</code>	can be either of the following: • logical, e.g. TRUE or FALSE. In case of FALSE, buttons will not be visible except for new rows. • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.
<code>canCloneRow</code>	can be either of the following: • logical, e.g. TRUE or FALSE. In case of FALSE, buttons will not be visible except for new rows. • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.
<code>utilityColumns</code>	named character vector. Defines names for (hidden) utility columns used by eDT to keep track of modifications. Should normally only be adjusted in rare case of name clashes with data. <pre>c(status = '_editbl_status', buttons = '_editbl_buttons', identity = '_editbl_identity', deleted = '_editbl_deleted')</pre>

Details

Works the same as [datatable](#). This function is however a shiny module and comes with additional arguments and different defaults. Instead of having `output$id = renderDT(DT::datatable(iris))`, `eDT(id = 'id', data = iris)` should be used on the server side. On the UI side [eDTOutput](#) should be used instead of [DTOutput](#).

Can also be used as standalone app when not ran in reactive context.

All arguments except `'id'` and `'env'` can be normal objects or reactive objects.

Value

list

- result reactive modified version of data (saved)
- state reactive current state of the data (unsaved)
- selected reactive selected rows of the data (unsaved)

- edited reactive edited rows (saved)
- inserted reactive inserted rows (saved)
- deleted reactive deleted rows (saved)

Author(s)

Jasper Schelfhout

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  # tibble support
  modifiedData <- editbl::eDT(tibble::as_tibble(mtcars))

  # data.table support
  modifiedData <- editbl::eDT(dplyr::lazy_dt(data.table::data.table(mtcars)))

  # database support
  tmpFile <- tempfile(fileext = ".sqlite")
  file.copy(system.file("extdata", "chinook.sqlite", package = 'editbl'), tmpFile)

  conn <- editbl::connectDB(dbname = tmpFile)
  modifiedData <- editbl::eDT(dplyr::tbl(conn, "Artist"), in_place = TRUE)
  DBI::dbDisconnect(conn)

  unlink(tmpFile)

  # Within shiny
  library(shiny)
  library(editbl)
  shinyApp(
    ui = fluidPage(fluidRow(column(12, eDTOutput('tbl')))),
    server = function(input, output) {
      eDT('tbl', iris, )
    }
  )

  # Custom inputUI
  editbl::eDT(mtcars, inputUI = function(id, data){
    ns <- NS(id)
    textInput(
      ns("mpg"),
      label = "mpg",
      value = data$mpg))

  # Do not allow delete
  editbl::eDT(mtcars, canDeleteRow = FALSE)
}
```

eDTOutput

UI part of eDT

Description

UI part of [eDT](#)

Usage

```
eDTOutput(id, ...)
```

Arguments

id	character(1)
...	arguments passed to DTOutput

Details

Works exactly like [DTOutput](#) apart from the fact that instead of the `outputId` argument, `id` is requested. Reason being that this function is a UI to a shiny module. This means that the datatable can be found under the id `'{namespace}-{id}-DT'` instead of `'{namespace}-{outputId}'`.

Also some minor CSS and javascript is executed for functional puposes.

Value

HTML

Author(s)

Jasper Schelfhout

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  # tibble support
  modifiedData <- editbl::eDT(tibble::as_tibble(mtcars))

  # data.table support
  modifiedData <- editbl::eDT(dplyr::lazy_dt(data.table::data.table(mtcars)))

  # database support
  tmpFile <- tempfile(fileext = ".sqlite")
  file.copy(system.file("extdata", "chinook.sqlite", package = 'editbl'), tmpFile)

  conn <- editbl::connectDB(dbname = tmpFile)
  modifiedData <- editbl::eDT(dplyr::tbl(conn, "Artist"), in_place = TRUE)
  DBI::dbDisconnect(conn)
```

```

unlink(tmpFile)

# Within shiny
library(shiny)
library(editbl)
shinyApp(
  ui = fluidPage(fluidRow(column(12, eDTOutput('tbl')))),
  server = function(input, output) {
    eDT('tbl',iris,)
  }
)

# Custom inputUI
editbl::eDT(mtcars, inputUI = function(id, data){
  ns <- NS(id)
  textInput(
    ns("mpg"),
    label = "mpg",
    value = data$mpg))})

# Do not allow delete
editbl::eDT(mtcars, canDeleteRow = FALSE)
}

```

e_rows_delete

Delete rows from a tibble

Description

Delete rows from a tibble

Usage

```

e_rows_delete(
  x,
  y,
  by = NULL,
  ...,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)

```

Arguments

x, y A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.

by	An unnamed character vector giving the key columns. The key columns must exist in both <code>x</code> and <code>y</code>. Keys typically uniquely identify each row, but this is only enforced for the key values of <code>y</code> when <code>rows_update()</code>, <code>rows_patch()</code>, or <code>rows_upsert()</code> are used. By default, we use the first column in <code>y</code>, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
unmatched	For <code>rows_update()</code>, <code>rows_patch()</code>, and <code>rows_delete()</code>, how should keys in <code>y</code> that are unmatched by the keys in <code>x</code> be handled? One of: • "error", the default, will error if there are any keys in <code>y</code> that are unmatched by the keys in <code>x</code>. • "ignore" will ignore rows in <code>y</code> with keys that are unmatched by the keys in <code>x</code>.
copy	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same <code>src</code> as <code>x</code> . This allows you to join tables across <code>srcs</code> , but it is a potentially expensive operation so you must opt into it.
in_place	Should <code>x</code> be modified in place? This argument is only relevant for mutable backends (e.g. databases, <code>data.tables</code>). When <code>TRUE</code>, a modified version of <code>x</code> is returned invisibly; when <code>FALSE</code>, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_delete`. Allows for specific implementations should the behavior differ from what's needed by `edittbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

e_rows_delete.default *Delete rows from a tibble*

Description

Delete rows from a tibble

Usage

```
## Default S3 method:
e_rows_delete(
  x,
  y,
  by = NULL,
  ...,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_delete`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

e_rows_insert	<i>Insert rows into a tibble</i>
---------------	----------------------------------

Description

Insert rows into a tibble

Usage

```
e_rows_insert(
  x,
  y,
  by = NULL,
  ...,
  conflict = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

<code>x, y</code>	A pair of data frames or data frame extensions (e.g. a tibble). <code>y</code> must have the same columns of <code>x</code> or a subset.
<code>by</code>	An unnamed character vector giving the key columns. The key columns must exist in both <code>x</code> and <code>y</code> . Keys typically uniquely identify each row, but this is only enforced for the key values of <code>y</code> when <code>rows_update()</code> , <code>rows_patch()</code> , or <code>rows_upsert()</code> are used. By default, we use the first column in <code>y</code> , since the first column is a reasonable place to put an identifier variable.

...	Other parameters passed onto methods.
conflict	For <code>rows_insert()</code>, how should keys in <code>y</code> that conflict with keys in <code>x</code> be handled? A conflict arises if there is a key in <code>y</code> that already exists in <code>x</code>. One of: • "error", the default, will error if there are any keys in <code>y</code> that conflict with keys in <code>x</code>. • "ignore" will ignore rows in <code>y</code> with keys that conflict with keys in <code>x</code>.
copy	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same src as <code>x</code> . This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should <code>x</code> be modified in place? This argument is only relevant for mutable backends (e.g. databases, <code>data.tables</code>). When <code>TRUE</code>, a modified version of <code>x</code> is returned invisibly; when <code>FALSE</code>, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_insert`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

`e_rows_insert.default` *Insert rows into a tibble*

Description

Insert rows into a tibble

Usage

```
## Default S3 method:
e_rows_insert(
  x,
  y,
  by = NULL,
  ...,
  conflict = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

<code>x, y</code>	A pair of data frames or data frame extensions (e.g. a tibble). <code>y</code> must have the same columns of <code>x</code> or a subset.
<code>by</code>	An unnamed character vector giving the key columns. The key columns must exist in both <code>x</code> and <code>y</code> . Keys typically uniquely identify each row, but this is only enforced for the key values of <code>y</code> when <code>rows_update()</code> , <code>rows_patch()</code> , or <code>rows_upsert()</code> are used. By default, we use the first column in <code>y</code> , since the first column is a reasonable place to put an identifier variable.
<code>...</code>	Other parameters passed onto methods.
<code>conflict</code>	For <code>rows_insert()</code> , how should keys in <code>y</code> that conflict with keys in <code>x</code> be handled? A conflict arises if there is a key in <code>y</code> that already exists in <code>x</code> . One of: • "error", the default, will error if there are any keys in <code>y</code> that conflict with keys in <code>x</code>. • "ignore" will ignore rows in <code>y</code> with keys that conflict with keys in <code>x</code>.
<code>copy</code>	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is TRUE, then <code>y</code> will be copied into the same src as <code>x</code> . This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
<code>in_place</code>	Should <code>x</code> be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of <code>x</code> is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_insert`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

e_rows_insert.dtplyr_step

rows_insert implementation for data.table backends.

Description

rows_insert implementation for data.table backends.

Usage

```
## S3 method for class 'dtplyr_step'
e_rows_insert(x, y, by = NULL, ..., copy = FALSE, in_place = FALSE)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around [rows_insert](#). Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

`e_rows_insert.tbl_dbi` *rows_insert* implementation for DBI backends.

Description

`rows_insert` implementation for DBI backends.

Usage

```
## S3 method for class 'tbl_dbi'
e_rows_insert(x, y, by = NULL, ..., copy = FALSE, in_place = FALSE)
```

Arguments

<code>x, y</code>	A pair of data frames or data frame extensions (e.g. a tibble). <code>y</code> must have the same columns of <code>x</code> or a subset.
<code>by</code>	An unnamed character vector giving the key columns. The key columns must exist in both <code>x</code> and <code>y</code> . Keys typically uniquely identify each row, but this is only enforced for the key values of <code>y</code> when <code>rows_update()</code> , <code>rows_patch()</code> , or <code>rows_upsert()</code> are used. By default, we use the first column in <code>y</code> , since the first column is a reasonable place to put an identifier variable.
<code>...</code>	Other parameters passed onto methods.
<code>copy</code>	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same src as <code>x</code> . This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
<code>in_place</code>	Should <code>x</code> be modified in place? This argument is only relevant for mutable backends (e.g. databases, <code>data.tables</code>). When <code>TRUE</code> , a modified version of <code>x</code> is returned invisibly; when <code>FALSE</code> , a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_insert`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

Examples

```
library(dplyr)

# Set up a test table
conn <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
artists_df <- data.frame(
  ArtistId = c(1,2),
  Name = c("AC/DC", "The Offspring")
)
DBI::dbWriteTable(conn, "Artist", artists_df)

# Insert new row
artists <- tbl(conn, "Artist")
DBI::dbBegin(conn)
e_rows_insert(artists,
  data.frame(ArtistId = 999, Name = "testArtist"),
  in_place = TRUE)

DBI::dbRollback(conn)
DBI::dbDisconnect(conn)
```

e_rows_update	<i>Update rows of a tibble</i>
---------------	--------------------------------

Description

Update rows of a tibble

Usage

```
e_rows_update(
  x,
  y,
  by = NULL,
  ...,
  match,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

`in_place` Should `x` be modified in place? This argument is only relevant for mutable backends (e.g. databases, `data.tables`).
When `TRUE`, a modified version of `x` is returned invisibly; when `FALSE`, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_update`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

`e_rows_update.data.frame`

rows_update implementation for data.frame backends.

Description

`rows_update` implementation for `data.frame` backends.

Usage

```
## S3 method for class 'data.frame'
e_rows_update(
  x,
  y,
  by = NULL,
  match = NULL,
  ...,
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

<code>x, y</code>	A pair of data frames or data frame extensions (e.g. a tibble). <code>y</code> must have the same columns of <code>x</code> or a subset.
<code>by</code>	An unnamed character vector giving the key columns. The key columns must exist in both <code>x</code> and <code>y</code> . Keys typically uniquely identify each row, but this is only enforced for the key values of <code>y</code> when <code>rows_update()</code> , <code>rows_patch()</code> , or <code>rows_upsert()</code> are used. By default, we use the first column in <code>y</code> , since the first column is a reasonable place to put an identifier variable.
<code>match</code>	named list consisting out of two equal length <code>data.frame</code> 's with columns defined in <code>by</code> . This allows for updates of columns defined in <code>by</code> .
<code>...</code>	Other parameters passed onto methods.
<code>copy</code>	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same <code>src</code> as <code>x</code> . This allows you to join tables across <code>srcs</code> , but it is a potentially expensive operation so you must opt into it.
<code>in_place</code>	Should <code>x</code> be modified in place? This argument is only relevant for mutable backends (e.g. databases, <code>data.tables</code>). When <code>TRUE</code> , a modified version of <code>x</code> is returned invisibly; when <code>FALSE</code> , a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_update`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

e_rows_update.default *Update rows of a tibble*

Description

Update rows of a tibble

Usage

```
## Default S3 method:
e_rows_update(
  x,
  y,
  by = NULL,
  ...,
  match = match,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

`in_place` Should `x` be modified in place? This argument is only relevant for mutable backends (e.g. databases, `data.tables`).
When `TRUE`, a modified version of `x` is returned invisibly; when `FALSE`, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_update`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

`e_rows_update.dtplyr_step`

rows_update implementation for data.table backends.

Description

`rows_update` implementation for `data.table` backends.

Usage

```
## S3 method for class 'dtplyr_step'
e_rows_update(
  x,
  y,
  by = NULL,
  match = NULL,
  ...,
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when <code>rows_update()</code>, <code>rows_patch()</code>, or <code>rows_upsert()</code> are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_update`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

e_rows_update.tbl_dbi *rows_update implementation for DBI backends.*

Description

rows_update implementation for DBI backends.

Usage

```
## S3 method for class 'tbl_dbi'
e_rows_update(
  x,
  y,
  by = NULL,
  match = NULL,
  ...,
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around [rows_update](#). Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

Examples

```
library(dplyr)

# Set up a test table
conn <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
artists_df <- data.frame(
  ArtistId = c(1,2),
  Name = c("AC/DC", "The Offspring")
)
DBI::dbWriteTable(conn, "Artist", artists_df)

# Update rows without changing the key.
artists <- tbl(conn, "Artist")
DBI::dbBegin(conn)
y <- data.frame(ArtistId = 1, Name = "DC/AC")
e_rows_update(
  x = artists,
  y = y,
  by = "ArtistId",
  in_place = TRUE)
DBI::dbRollback(conn)

# Update key values of rows.
DBI::dbBegin(conn)
y <- data.frame(ArtistId = 999, Name = "DC/AC")
match <- list(
  x = data.frame("ArtistId" = 1),
  y = data.frame("ArtistId" = 999)
)
e_rows_update(
  x = artists,
  y = y,
  match = match,
```

```

    by = "ArtistId",
    in_place = TRUE)
DBI::dbRollback(conn)
DBI::dbDisconnect(conn)

```

foreignTbl

*Create a foreign tibble***Description**

Create a foreign tibble

Usage

```

foreignTbl(
  x,
  y,
  by = intersect(dplyr::tbl_vars(x), dplyr::tbl_vars(y)),
  naturalKey = dplyr::tbl_vars(y),
  allowNew = FALSE
)

```

Arguments

<code>x</code>	tbl. The referencing table.
<code>y</code>	tbl. The referenced table.
<code>by</code>	character. Column names to match on. Note that you should rename and/or typecast the columns in <code>y</code> should they not exactly match the columns in <code>x</code> .
<code>naturalKey</code>	character. The columns that form the natural key in <code>y</code> . These are the only ones that can actually get modified in eDT when changing cells in the table. Reasoning being that these columns should be sufficient to uniquely identify a row in the referenced table. All other columns will be automatically fetched and filled in.
<code>allowNew</code>	logical. Whether or not new values are allowed. If TRUE, the rows in the foreignTbl will only be used as suggestions, not restrictions.

Details

This is a tibble that can be passed onto [eDT](#) as a referenced table.

It is the equivalent of a database table to which the data tbl of eDT has a foreign key.

It will be merged with the tbl passed onto the data argument allowing to provide restrictions for certain columns.

Note that row uniqueness for the columns used in `by` and `naturalKey` is assumed. This assumption will however not be checked since it is an expensive operation on big datasets. However, if violated, it might give errors or unexpected results during usage of the eDT module.

Value

List with unmodified arguments. However, they have now been checked for validity.

- y, see argument y.
- by, see argument by.
- naturalKey, see argument naturalKey.
- allowNew, see argument allowNew

Author(s)

Jasper Schelfhout

Examples

```
a <- tibble::tibble(
  first_name = c("Albert", "Donald", "Mickey"),
  last_name_id = c(1,2,2)
)

b <- foreignTbl(
  a,
  tibble::tibble(
    last_name = c("Einstein", "Duck", "Mouse"),
    last_name_id = c(1,2,3)
  ),
  by = "last_name_id",
  naturalKey = "last_name"
)

## Only run this in interactive R sessions
if(interactive()){
  eDT(a,
    foreignTbIs = list(b),
    options = list(columnDefs = list(list(visible=FALSE, targets="last_name_id")))
  )
}
```

get_db_table_name	<i>Get name of the tbl in the database</i>
-------------------	--

Description

Get name of the tbl in the database

Usage

```
get_db_table_name(x)
```

Arguments

x tbl_dbi

Value

SQL, the table name as used in the database

inputServer	<i>An input server for a data.frame</i>
-------------	---

Description

An input server for a data.frame

Usage

```
inputServer(id, data, ...)
```

Arguments

id	character(1) module id
data	single row data.frame
...	further arguments for methods

Details

A new method for this can be added if you wish to alter the default behavior of the pop-up modals in [eDT](#).

Value

modified version of data

Author(s)

Jasper Schelfhout

Examples

```
if(interactive()){
  library(shiny)
  ui <- inputUI('id')
  server <- function(input,output,session){
    input <- inputServer("id", mtcars[1,])
    observe({print(input())})
  }
  shinyApp(ui, server)
}
```

inputServer.default	<i>An input server for a data.frame</i>
---------------------	---

Description

An input server for a data.frame

Usage

```
## Default S3 method:  
inputServer(id, data, colnames, notEditable, foreignTbls, ...)
```

Arguments

id	character(1) module id
data	single row data.frame
colnames	named character
notEditable	character columns that should not be edited
foreignTbls	list of foreignTbls. See foreignTbl
...	for compatibility with other methods

Details

Reads all inputs ids that are identical to column names of the data and updates the data.

Value

reactive modified version of data

Author(s)

Jasper Schelfhout

inputUI	<i>An input UI for a data.frame</i>
---------	-------------------------------------

Description

An input UI for a data.frame

Usage

```
inputUI(id, ...)
```

Arguments

id character(1) module id
... arguments passed onto methods

Details

A new method for this can be added if you wish to alter the default behavior of the pop-up modals in [eDT](#).

Value

HTML. A set of input fields corresponding to the given row.

Author(s)

Jasper Schelfhout

Examples

```
if(interactive()){
  library(shiny)
  ui <- inputUI('id')
  server <- function(input,output,session){
    input <- inputServer("id", mtcars[1,])
    observe({print(input())})
  }
  shinyApp(ui, server)
}
```

inputUI.default	<i>UI part for modal with input fields for editing</i>
-----------------	--

Description

UI part for modal with input fields for editing

Usage

```
## Default S3 method:
inputUI(id, ...)
```

Arguments

id character module id
... for compatibility with method

Details

The UI elements that have an id identical to a column name are used for updating the data.

Value

HTML. A set of input fields corresponding to the given row.

Author(s)

Jasper Schelfhout

joinForeignTbl	<i>Merge a tbl with it a foreignTbl</i>
----------------	---

Description

Merge a tbl with it a foreignTbl

Usage

```
joinForeignTbl(
  tbl,
  foreignTbl,
  keepNA = TRUE,
  by = foreignTbl$by,
  copy = TRUE,
  type = c("inner", "left")[1]
)
```

Arguments

tbl	tbl
foreignTbl	list as created by foreignTbl
keepNA	logical keep rows from tbl with NA keys.
by	named character, columns to join on.
copy	logical, whether or not to copy the foreignTbl to the source of argument tbl for joining.
type	character(1), type of joint to perform. Can be 'inner' or 'left'.

Details

see also dplyr join functions, for example `dplyr::left_join`.

Value

tbl, containing both columns from argument tbl and argument foreignTbl.

Author(s)

Jasper Schelfhout

rows_delete.dtplyr_step*rows_delete implementation for data.table backends.*

Description

rows_delete implementation for data.table backends.

Usage

```
## S3 method for class 'dtplyr_step'
rows_delete(x, y, by = NULL, ..., unmatched, copy = FALSE, in_place = FALSE)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

runDemoApp

Run a demo app

Description

Run a demo app

Usage

```
runDemoApp(app = "database", ...)
```

Arguments

<code>app</code>	demoApp to run. Options: database / mtcars / custom
<code>...</code>	arguments passed onto the demoApp

Details

These apps are for illustrative purposes.

Value

An object that represents the app. Printing the object or passing it to `runApp()` will run the app.

Examples

```
## Only run this example in interactive R sessions
if(interactive()){

  # Database
  tmpFile <- tempfile(fileext = ".sqlite")
  file.copy(system.file("extdata", "chinook.sqlite", package = 'editbl'), tmpFile)

  conn <- connectDB(dbname = tmpFile)

  runDemoApp(app = "database", conn = conn)
  DBI::dbDisconnect(conn)

  unlink(tmpFile)

  # mtcars
  runDemoApp(app = "mtcars")

  # Any tibble of your liking
  runDemoApp(app = "custom", dplyr::tibble(iris))
}
```

selectInputDT_Server *Server part to use a [datatable](#) as select input*

Description

Server part to use a [datatable](#) as select input

Usage

```
selectInputDT_Server(
  id,
  label = "",
  choices,
  selected = NULL,
  multiple = FALSE
)
```

Arguments

id	character(1) same one as used in selectInputDT_UI
label	character(1)
choices	data.frame
selected	data.frame with rows available in choices.
multiple	logical. Whether or not multiple row selection is allowed

Value

A selection of rows from the `data.frame` provided under choices.

Author(s)

Jasper Schelfhout

See Also

`shiny::selectInput`. This function can be more convenient for selecting rows with multiple columns.

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  ui <- selectInputDT_UI('id')
  data <- data.frame(id = 1:3, name = letters[1:3])
  server <- function(input,output, session){
    selected = selectInputDT_Server('id', choices = data, selected = data[1,] )
    observe({print(selected())})
  }
  shiny::shinyApp(ui, server)
}
```

selectInputDT_UI	<i>UI part of a DT select input</i>
------------------	-------------------------------------

Description

UI part of a DT select input

Usage

```
selectInputDT_UI(id)
```

Arguments

`id` `character(1)` same one as used in [selectInputDT_Server](#)

Value

HTML

Author(s)

Jasper Schelfhout

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  ui <- selectInputDT_UI('id')
  data <- data.frame(id = 1:3, name = letters[1:3])
  server <- function(input,output, session){
    selected = selectInputDT_Server('id', choices = data, selected = data[1,] )
    observe({print(selected())})
  }
  shiny::shinyApp(ui, server)
}
```

Index

connectDB, [2](#)
customButton, [3](#)

datatable, [7](#), [8](#), [36](#)
DTOutput, [8](#), [10](#)

e_rows_delete, [11](#)
e_rows_delete.default, [13](#)
e_rows_insert, [14](#)
e_rows_insert.default, [15](#)
e_rows_insert.dtplyr_step, [17](#)
e_rows_insert.tbl_dbi, [18](#)
e_rows_update, [20](#)
e_rows_update.data.frame, [21](#)
e_rows_update.default, [23](#)
e_rows_update.dtplyr_step, [24](#)
e_rows_update.tbl_dbi, [26](#)
eDT, [3](#), [4](#), [10](#), [28](#), [30](#), [32](#)
eDTOutput, [8](#), [10](#)

foreignTbl, [7](#), [28](#), [31](#), [33](#)

get_db_table_name, [29](#)

inputServer, [30](#)
inputServer.default, [31](#)
inputUI, [31](#)
inputUI.default, [32](#)

joinForeignTbl, [33](#)
JS, [5](#)

options, [5](#)

rows_delete, [12](#), [14](#)
rows_delete.dtplyr_step, [34](#)
rows_insert, [15–17](#), [19](#)
rows_update, [21](#), [22](#), [24–26](#)
runApp(), [35](#)
runDemoApp, [35](#)

selectInputDT_Server, [36](#), [37](#)
selectInputDT_UI, [36](#), [37](#)