

# Package: editbl (via r-universe)

September 12, 2024

**Type** Package

**Version** 1.0.5

**Date** 2024-05-21

**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, 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

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Repository** <https://openanalytics.r-universe.dev>

**RemoteUrl** <https://github.com/openanalytics/editbl>

**RemoteRef** HEAD

**RemoteSha** 74fbc97bdb4b763e2512fd3253f3ef1d24c182a9

## Contents

addButtons	3
beginTransaction	4
castForDisplay	4
castFromTbl	5
castToFactor	5
castToSQLSupportedType	6
castToTbl	6
castToTemplate	7
checkForeignTbls	8
coalesce	8
coerceColumns	9
coerceValue	9
commitTransaction	10
connectDB	10
createButtons	11
createButtonsHTML	11
customButton	12
demoServer_custom	13
demoServer_DB	14
demoServer_mtcars	14
demoUI_custom	15
demoUI_DB	15
demoUI_mtcars	16
devServer	16
devUI	17
disableDoubleClickButtonCss	17
eDT	18
eDTOutput	22
eDT_app	24
eDT_app_server	24
eDT_app_ui	25
e_rows_insert	25
e_rows_insert.default	27
e_rows_insert.dtplyr_step	28
e_rows_insert.tbl_dbi	29
e_rows_update	31
e_rows_update.data.frame	33
e_rows_update.default	34
e_rows_update.dtplyr_step	36
e_rows_update.tbl_dbi	37
fillDeductedColumns	39
fixInteger64	40
foreignTbl	41
getColumnTypeSums	42
getNonNaturalKeyCols	43
get_db_table_name	44

initData . . . . .	44
inputServer . . . . .	45
inputServer.default . . . . .	46
inputUI . . . . .	46
inputUI.default . . . . .	47
joinForeignTbl . . . . .	48
rollbackTransaction . . . . .	49
rowInsert . . . . .	49
rows_delete.dplyr_step . . . . .	50
rowUpdate . . . . .	51
runDemoApp . . . . .	52
runDemoApp_custom . . . . .	53
runDemoApp_DB . . . . .	53
runDemoApp_mtcars . . . . .	54
runDevApp . . . . .	54
selectInputDT_Server . . . . .	55
selectInputDT_UI . . . . .	56
shinyInput . . . . .	57
standardizeArgument_colnames . . . . .	57
standardizeArgument_editable . . . . .	58
whereSQL . . . . .	59

## **Index** **60**

---

addButtons	<i>Add modification buttons as a column</i>
------------	---

---

### **Description**

Add modification buttons as a column

### **Usage**

```
addButtons(df, columnName, ns)
```

### **Arguments**

df	data.frame
columnName	character(1)
ns	namespace function

### **Value**

df with extra column containing buttons

### **Author(s)**

Jasper Schelfhout

`beginTransaction`      *Start a transaction for a tibble*

---

**Description**

Start a transaction for a tibble

**Usage**

```
beginTransaction(tbl)
```

**Arguments**

`tbl`                  `tbl`

**Author(s)**

Jasper Schelfhout

---

`castForDisplay`      *Cast columns in data.frame to editable types in datatable*

---

**Description**

Cast columns in `data.frame` to editable types in `datatable`

**Usage**

```
castForDisplay(data, cols = colnames(data))
```

**Arguments**

`data`                  `data.frame`  
`cols`                  character columns to perform casting on.

**Value**

`data.frame` with some columns cast to another type

**Author(s)**

Jasper Schelfhout

---

castFromTbl	<i>Cast tbl to class of template</i>
-------------	--------------------------------------

---

**Description**

Cast tbl to class of template

**Usage**

```
castFromTbl(tbl, template)
```

**Arguments**

tbl	tbl
template	tabular object like <code>data.frame</code> or <code>data.table</code> or <code>tbl</code> .

**Value**

tbl cast to the type of template

**Author(s)**

Jasper Schelfhout

---

castToFactor	<i>Cast all columns that exist in a foreignTbl to factor</i>
--------------	--

---

**Description**

Cast all columns that exist in a foreignTbl to factor

**Usage**

```
castToFactor(data, foreignTbIs)
```

**Arguments**

data	<code>data.frame</code>
foreignTbIs	list of foreign tbIs as created by <a href="#">foreignTbl</a>

**Details**

Can be used to fixate possible options when editing.

**Value**

data.frame

**Author(s)**

Jasper Schelfhout

---

castToSQLSupportedType

*Cast the data type to something supported by SQL.*

---

**Description**

Cast the data type to something supported by SQL.

**Usage**

castToSQLSupportedType(x)

**Arguments**

x                    single value or vector of values

**Value**

x, possibly cast to different type

**Author(s)**

Jasper Schelfhout

---

castToTbl

*Cast data to tbl*

---

**Description**

Cast data to tbl

**Usage**

castToTbl(data)

**Arguments**

data                object

**Value**

tbl

**Author(s)**

Jasper Schelfhout

---

castToTemplate      *Cast tbl or data.frame x to the types of the template*

---

**Description**

Cast tbl or data.frame x to the types of the template

**Usage**

```
castToTemplate(x, template)
```

**Arguments**

x	data.frame, tbl or data.table
template	data.frame, tbl or data.table

**Details**

If template is a tbl with database source, convert to an in-memory tibble with same data types instead.

Rownames might differ or get lost.

**Value**

object containing data of x in the class and structure of the template.

**Author(s)**

Jasper Schelfhout

---

checkForeignTbls	<i>Check if all rows in tbl fulfill foreignTbl constraints</i>
------------------	--

---

**Description**

Check if all rows in tbl fulfill foreignTbl constraints

**Usage**

```
checkForeignTbls(tbl, foreignTbls)
```

**Arguments**

tbl	tbl
foreignTbls	list of foreign tbls as created by <a href="#">foreignTbl</a>

**Value**

logical stating if tbl fulfills all constraints imposed by all foreign tbls.

**Author(s)**

Jasper Schelfhout

---

coalesce	<i>Return first non NULL argument</i>
----------	---------------------------------------

---

**Description**

Return first non NULL argument

**Usage**

```
coalesce(...)
```

**Arguments**

...	set of arguments
-----	------------------

**Author(s)**

Jasper Schelfhout



---

coerceColumns	<i>Cast columns to the type of the template</i>
---------------	---

---

**Description**

Cast columns to the type of the template

**Usage**

```
coerceColumns(template, x)
```

**Arguments**

template	data.frame
x	data.frame

**Details**

only affects columns in both the template and x

---

coerceValue	<i>DT::coerceValue with better POSIXct support</i>
-------------	--

---

**Description**

DT::coerceValue with better POSIXct support

**Usage**

```
coerceValue(val, old)
```

**Arguments**

val	A character string.
old	An old value, whose type is the target type of val.

**Details**

Will assume UTC in case no timezone is specified.

**Author(s)**

Jasper Schelfhout

commitTransaction      *Start a transaction for a tibble*

---

**Description**

Start a transaction for a tibble

**Usage**

```
commitTransaction(tbl)
```

**Arguments**

tbl                      tbl

**Author(s)**

Jasper Schelfhout

---

connectDB                *Connect to a database.*

---

**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)
```

---

createButtons	<i>Create buttons to modify the row. See <a href="#">createButtonsHTML</a></i>
---------------	--

---

**Description**

Create buttons to modify the row. See [createButtonsHTML](#)

**Usage**

```
createButtons(suffix, ns)
```

**Arguments**

suffix	character(1)
ns	character(1) namespace

**Details**

buttons used per row in the app.

**Value**

character HTML

---

createButtonsHTML	<i>Helper function to write HTML</i>
-------------------	--------------------------------------

---

**Description**

Helper function to write HTML

**Usage**

```
createButtonsHTML(suffix = "%1$s", ns = "%2$s")
```

**Arguments**

suffix	character(1) sprintf placeholer for suffix
ns	character(1) sprintf placeholder for ns

**Details**

generate HTML as character once and reuse. Since buttons have to be generated a lot, this otherwise slows down the app.

**Value**

character(1) HTML to be filled in with sprintf

---

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)  
}
```

---

demoServer_custom	<i>Server of the mtcars demo app</i>
-------------------	--------------------------------------

---

**Description**

Server of the mtcars demo app

**Usage**

```
demoServer_custom(id, x)
```

**Arguments**

id	character(1)
x	tbl

**Value**

NULL, just executes the module server.

**Author(s)**

Jasper Schelfhout

demoServer\_DB      *Server of the DB demo app*

---

**Description**

Server of the DB demo app

**Usage**

```
demoServer_DB(id, conn)
```

**Arguments**

id	character(1)
conn	database connection object as given by <a href="#">dbConnect</a> .

**Value**

NULL, just executes the module server.

**Author(s)**

Jasper Schelfhout

---

demoServer\_mtcars      *Server of the mtcars demo app*

---

**Description**

Server of the mtcars demo app

**Usage**

```
demoServer_mtcars(id)
```

**Arguments**

id	character(1)
----	--------------

**Value**

NULL, just executes the module server.

**Author(s)**

Jasper Schelfhout

---

demoUI\_custom      *UI of the demo mtcars app*

---

**Description**

UI of the demo mtcars app

**Usage**

```
demoUI_custom(id)
```

**Arguments**

id                    character(1)

**Value**

HTML

**Author(s)**

Jasper Schelfhout

---

demoUI\_DB            *UI of the DB demo app*

---

**Description**

UI of the DB demo app

**Usage**

```
demoUI_DB(id, conn)
```

**Arguments**

id                    character(1)  
conn                  database connection object as given by [dbConnect](#).

**Value**

HTML

**Author(s)**

Jasper Schelfhout

demoUI\_mtcars      *UI of the demo mtcars app*

---

**Description**

UI of the demo mtcars app

**Usage**

```
demoUI_mtcars(id)
```

**Arguments**

id                    character(1)

**Value**

HTML

**Author(s)**

Jasper Schelfhout

---

devServer            *Server of the development app*

---

**Description**

Server of the development app

**Usage**

```
devServer(id, conn)
```

**Arguments**

id                    character(1)  
conn                  database connection object as given by [dbConnect](#).

**Value**

NULL, just executes the module server.

**Author(s)**

Jasper Schelfhout



---

devUI                      *UI of the development app*

---

**Description**

UI of the development app

**Usage**

```
devUI(id, conn)
```

**Arguments**

id	character(1)
conn	database connection object as given by <a href="#">dbConnect</a> .

**Value**

HTML

**Author(s)**

Jasper Schelfhout

---

disableDoubleClickButtonCss  
*Function to generate CSS to disable clicking events on a column*

---

**Description**

Function to generate CSS to disable clicking events on a column

**Usage**

```
disableDoubleClickButtonCss(id)
```

**Arguments**

id	character(1) namespaced id of the datatable
----	---

**Details**

<https://stackoverflow.com/questions/60406027/how-to-disable-double-click-reactivity-for-specific-c>  
<https://stackoverflow.com/questions/75406546/apply-css-styling-to-a-single-dt-datatable>

**Value**

character CSS

eDT

*Create a modifiable datatable.***Description**

Create a modifiable datatable.

**Usage**

```
eDT(
  data,
  options = list(dom = "BfrtIip", 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(),
  statusColor = c(insert = "#e6e6e6", update = "#32a6d3", delete = "#e52323"),
  inputUI = editbl::inputUI,
  defaults = tibble(),
  env = environment()
)
```

**Arguments**

`data` `tbl`. The function will automatically cast to `tbl` if needed.

options	a list of initialization options (see <a href="https://datatables.net/reference/option/">https://datatables.net/reference/option/</a> ); the character options wrapped in <code>JS()</code> will be treated as literal JavaScript code instead of normal character strings; you can also set options globally via <code>options(DT.options = list(...))</code> , and global options will be merged into this options argument if set
class	the CSS class(es) of the table; see <a href="https://datatables.net/manual/styling/classes">https://datatables.net/manual/styling/classes</a>
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 <code>'newName1' = i1, 'newName2' = i2</code> or <code>c('newName1' = 'oldName1', 'newName2' = 'oldName2', ...)</code> , where <code>newName</code> is the new name you want to show in the table, and <code>i</code> or <code>oldName</code> 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 list to this argument of the form <code>list(position = 'top', clear = TRUE, plain = FALSE)</code> , where <code>clear</code> indicates whether you want the clear buttons in the input boxes, and <code>plain</code> means if you want to use Bootstrap form styles or plain text input styles for the text input boxes
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. <code>1:5</code> (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 rownames
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, <b>DataTables 'default' styling</b> is used. If set explicitly to 'bootstrap' or 'bootstrap4', one must take care to ensure Bootstrap's HTML dependencies (as well as Bootstrap 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 ( <a href="https://datatables.net/extensions/index">https://datatables.net/extensions/index</a> )
plugins	a character vector of the names of DataTables plug-ins ( <a href="https://rstudio.github.io/DT/plugins.html">https://rstudio.github.io/DT/plugins.html</a> ). 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 <code>numeric = "none"</code> 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 <a href="#">datatable</a>
foreignTbls	list. List of objects created by <a href="#">foreignTbl</a>
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 Sys.time() or uuid::UUIDgenerate() as well as dynamic inputs.
env	environment in which the server function is running. Should normally not be modified.

## 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)

## 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(dtplyr::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)})
}
```

---

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)})
  }

```

---

eDT\_app

*Open interactive app to explore and modify data*


---

### Description

Open interactive app to explore and modify data

### Usage

```
eDT_app(...)
```

### Arguments

... arguments past to [eDT](#)

### Details

When [eDT](#) is not used within the server of a shiny app, it will call this function to start up a shiny app itself. Just as `DT::datatable()` displays a table in the browser when called upon interactively.

### Value

data (or a modified version thereof) once you click 'close'

---

eDT\_app\_server

*Server of eDT\_app*


---

### Description

Server of eDT\_app

### Usage

```
eDT_app_server(moduleId = "nevergonnagiveyouup", ...)
```

### Arguments

moduleId character(1) id to connect with eDT\_app\_server  
 ... arguments passed to [eDT](#)



**Value**

moduleServer which on application stop returns version of x with made changes

**Author(s)**

Jasper Schelfhout

---

eDT_app_ui	<i>UI of eDT_app</i>
------------	----------------------

---

**Description**

UI of eDT\_app

**Usage**

```
eDT_app_ui(moduleId = "nevergonnagiveyouup", eDTId = "nevergonnaletyoudown")
```

**Arguments**

moduleId	character(1) id to connect with eDT_app_server
eDTId	character(1) id to connect <a href="#">eDTOutput</a> to <a href="#">eDT</a> within the module.

**Value**

HTML

**Author(s)**

Jasper Schelfhout

---

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.
<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: <ul style="list-style-type: none"> <li>• "error", the default, will error if there are any keys in <code>y</code> that conflict with keys in <code>x</code>.</li> <li>• "ignore" will ignore rows in <code>y</code> with keys that conflict with keys in <code>x</code>.</li> </ul>
<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, data.tables). 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 `edit_tbl`. 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

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.
conflict	For rows_insert(), how should keys in y that conflict with keys in x be handled? A conflict arises if there is a key in y that already exists in x. One of: <ul style="list-style-type: none"><li>• "error", the default, will error if there are any keys in y that conflict with keys in x.</li><li>• "ignore" will ignore rows in y with keys that conflict with keys in x.</li></ul>
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.

---

`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**

<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.

`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 <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_insert`. Allows for specific implementations should the behavior differ from what's needed by `edit.tbl`. 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: <ul style="list-style-type: none"> <li>• "error", the default, will error if there are any keys in y that are unmatched by the keys in x.</li> <li>• "ignore" will ignore rows in y with keys that are unmatched by the keys in x.</li> </ul>
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

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 `edit.tbl`. 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**

- |                   |   |
|-------------------|---|
| <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.<br>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.
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: <ul style="list-style-type: none"> <li>• "error", the default, will error if there are any keys in y that are unmatched by the keys in x.</li> <li>• "ignore" will ignore rows in y with keys that are unmatched by the keys in x.</li> </ul>
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 `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\_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 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

---

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**

- |                   |   |
|-------------------|---|
| <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.<br>By default, we use the first column in <code>y</code> , 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 `edit.tbl`. 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)

```

---

fillDeductedColumns    *Fill data columns based on foreignTbls*

---

### Description

Fill data columns based on foreignTbls

### Usage

```
fillDeductedColumns(tbl, foreignTbls)
```

### Arguments

tbl	tbl
foreignTbls	list of foreign tbls as created by <a href="#">foreignTbl</a>

### Details

When a combination of columns is not found in the foreignTbl, fill the deductedColumns with NA. on foreignTbls suggesting conflicting data, an arbitrary choice is made. It is best to afterwards check with checkForeignTbls to see if a valid result is obtained.

**Value**

tbl

**Author(s)**

Jasper Schelfhout

---

fixInteger64	<i>Replace instances of integer64 with actual NA values instead of weird default 9218868437227407266</i>
--------------	--

---

**Description**

Replace instances of integer64 with actual NA values instead of weird default 9218868437227407266

**Usage**

```
fixInteger64(x)
```

**Arguments**

x	data.frame
---	------------

**Details**

[github issue](#)

**Value**

x with integer64 columns set to bit64::as.integer64(NA)

**Author(s)**

Jasper Schelfhout



---

foreignTbl	<i>Create a foreign tibble</i>
------------	--------------------------------

---

## 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

x	tbl. The referencing table.
y	tbl. The referenced table.
by	character. Column names to match on. Note that you should rename and/or typecast the columns in y should they not exactly match the columns in x.
naturalKey	character. The columns that form the natural key in y. 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.
allowNew	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")))
  )
}
```

---

getColumnTypeSums      *Get types of columns in a tbl*

---

**Description**

Get types of columns in a `tbl`

**Usage**

```
getColumnTypeSums(tbl)
```

**Arguments**

tbl                   tbl

**Value**

named list with types of the columns

**Author(s)**

Jasper Schelfhout

---

`getNonNaturalKeyCols`   *Get all columns that are not natural keys*

---

**Description**

Get all columns that are not natural keys

**Usage**

```
getNonNaturalKeyCols(foreignTbls)
```

**Arguments**

foreignTbls    list of foreign tbls as created by `foreignTbl`

**Value**

character

**Author(s)**

Jasper Schelfhout

---

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

---

initData	<i>Add some extra columns to data to allow for / keep track of modifications</i>
----------	--

---

**Description**

Add some extra columns to data to allow for / keep track of modifications

**Usage**

```
initData(
  data,
  ns,
  buttonCol = "buttons",
  statusCol = "status",
  deleteCol = "deleted",
  iCol = "i"
)
```

**Arguments**

data	data.frame
ns	namespace function
buttonCol	character(1) name of column with buttons
statusCol	character(1) name of column with general status (e.g. modified or not).
deleteCol	character(1) name of the column with deletion status.
iCol	character(1) name of column containing a unique identifier.

**Value**

data with extra columns buttons, status, i.

**Author(s)**

Jasper Schelfhout

---

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    *An input server for a data.frame*

---

### 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 <a href="#">foreignTbl</a>
...	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    *An input UI for a data.frame*

---

### 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                    *UI part for modal with input fields for editing*

---

**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 <a href="#">foreignTbl</a>
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

---

rollbackTransaction     *Start a transaction for a tibble*

---

**Description**

Start a transaction for a tibble

**Usage**

```
rollbackTransaction(tbl)
```

**Arguments**

tbl                    tbl

**Author(s)**

Jasper Schelfhout

---

rowInsert                    *Add a row to a table in the database.*

---

**Description**

Add a row to a table in the database.

**Usage**

```
rowInsert(conn, table, values)
```

**Arguments**

conn                    database connection object as given by [dbConnect](#).  
table                    character  
values                    named list, row to add. Names are database column names. Unspecified columns will get database defaults.

**Value**

integer number of affected rows.

---

```
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: <ul style="list-style-type: none"> <li>"error", the default, will error if there are any keys in y that are unmatched by the keys in x.</li> <li>"ignore" will ignore rows in y with keys that are unmatched by the keys in x.</li> </ul>
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

---

rowUpdate	<i>Update rows in the database.</i>
-----------	-------------------------------------

---

### Description

Update rows in the database.

### Usage

```
rowUpdate(conn, table, values, where)
```

### Arguments

<code>conn</code>	database connection object as given by <a href="#">dbConnect</a> .
<code>table</code>	character
<code>values</code>	named list, values to be set. Names are database column names.
<code>where</code>	named list, values to filter on. Names are database column names. If <code>NULL</code> no filter is applied.

### Value

integer number of affected rows.

---

runDemoApp	<i>Run a demo app</i>
------------	-----------------------

---

**Description**

Run a demo app

**Usage**

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

**Arguments**

app	demoApp to run. Options: database / mtcars / custom
...	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))
}
```

---

runDemoApp_custom	<i>Run a custom demo app</i>
-------------------	------------------------------

---

**Description**

Run a custom demo app

**Usage**

runDemoApp\_custom(x)

**Arguments**

x	tbl
---	-----

**Value**

An object that represents the app. Printing the object or passing it to [runApp\(\)](#) will run the app.

---

runDemoApp_DB	<i>Run a demo app</i>
---------------	-----------------------

---

**Description**

Run a demo app

**Usage**

runDemoApp\_DB()

**Value**

An object that represents the app. Printing the object or passing it to [runApp\(\)](#) will run the app.

---

runDemoApp_mtcars	<i>Run a demo app</i>
-------------------	-----------------------

---

**Description**

Run a demo app

**Usage**

```
runDemoApp_mtcars()
```

**Value**

An object that represents the app. Printing the object or passing it to [runApp\(\)](#) will run the app.

---

runDevApp	<i>Run a development app</i>
-----------	------------------------------

---

**Description**

Run a development app

**Usage**

```
runDevApp()
```

**Details**

This app prints some of the server objects and has a button to interactively browse the code. This is useful for debugging and experimenting with new features.

**Value**

An object that represents the app. Printing the object or passing it to [runApp\(\)](#) will run the app.

---

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 <a href="#">selectInputDT_UI</a>
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      *UI part of a DT select input*

---

### 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)  
}
```



---

shinyInput	<i>Get a shiny input for a column of a tbl</i>
------------	--

---

**Description**

Get a shiny input for a column of a tbl

**Usage**

```
shinyInput(x, inputId, label, selected)
```

**Arguments**

x	column
inputId	shiny input Id
label	character(1)
selected	object of class of x

**Value**

shiny input

**Author(s)**

Jasper Schelfhout

---

standardizeArgument_colnames	<i>Standardize colnames argument to the format of named character vector</i>
------------------------------	--

---

**Description**

Standardize colnames argument to the format of named character vector

**Usage**

```
standardizeArgument_colnames(colnames, data)
```

**Arguments**

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
data	tbl. The function will automatically cast to tbl if needed.

**Value**

named character vector

**Author(s)**

Jasper Schelfhout

---

standardizeArgument\_editable

*Standardized editable argument to be in the form of a list*

---

**Description**

Standardized editable argument to be in the form of a list

**Usage**

```
standardizeArgument_editable(editable, data)
```

**Arguments**

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 `list(target = TARGET, disable = list(columns = INDICES))`, 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 `list(target = TARGET, numeric = INDICES)` 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 `list(target = TARGET, area = INDICES)` 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 `list(target = TARGET, numeric = INDICES1, area = INDICES2)`. Finally, you can edit date cells with a calendar with `list(target = TARGET, date = INDICES)`; 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.

`data` `tbl`. The function will automatically cast to `tbl` if needed.

### Value

list of the form `list(target = foo, ...)`

### Author(s)

Jasper Schelfhout

---

whereSQL

*Generate where sql*

---

### Description

Generate where sql

### Usage

```
whereSQL(conn, table, column, operator = "in", values = NULL)
```

### Arguments

<code>conn</code>	database connection object as given by <a href="#">dbConnect</a> .
<code>table</code>	character table name (or alias used in query)
<code>column</code>	character column of table
<code>operator</code>	character
<code>values</code>	character vector of values

### Value

character sql

### Author(s)

Jasper Schelfhout

# Index

addButtons, 3

beginTransaction, 4

castForDisplay, 4

castFromTbl, 5

castToFactor, 5

castToSQLSupportedType, 6

castToTbl, 6

castToTemplate, 7

checkForeignTbls, 8

coalesce, 8

coerceColumns, 9

coerceValue, 9

commitTransaction, 10

connectDB, 10

createButtons, 11

createButtonsHTML, 11, 11

customButton, 12

datatable, 21, 55

dbConnect, 14–17, 49, 51, 59

demoServer\_custom, 13

demoServer\_DB, 14

demoServer\_mtcars, 14

demoUI\_custom, 15

demoUI\_DB, 15

demoUI\_mtcars, 16

devServer, 16

devUI, 17

disableDoubleClickButtonCss, 17

DTOutput, 21, 23

e\_rows\_insert, 25

e\_rows\_insert.default, 27

e\_rows\_insert.dtplyr\_step, 28

e\_rows\_insert.tbl\_dbi, 29

e\_rows\_update, 31

e\_rows\_update.data.frame, 33

e\_rows\_update.default, 34

e\_rows\_update.dtplyr\_step, 36

e\_rows\_update.tbl\_dbi, 37

eDT, 12, 18, 22, 24, 25, 41, 45, 47

eDT\_app, 24

eDT\_app\_server, 24

eDT\_app\_ui, 25

eDTOutput, 21, 22, 25

fillDeductedColumns, 39

fixInteger64, 40

foreignTbl, 5, 8, 21, 39, 41, 43, 46, 48

get\_db\_table\_name, 44

getColumnTypeSums, 42

getNonNaturalKeyCols, 43

initData, 44

inputServer, 45

inputServer.default, 46

inputUI, 46

inputUI.default, 47

joinForeignTbl, 48

JS, 19

options, 19

rollbackTransaction, 49

rowInsert, 49

rows\_delete.dtplyr\_step, 50

rows\_insert, 26, 28–30

rows\_update, 32, 33, 35, 36, 38

rowUpdate, 51

runApp(), 52–54

runDemoApp, 52

runDemoApp\_custom, 53

runDemoApp\_DB, 53

runDemoApp\_mtcars, 54

runDevApp, 54

selectInputDT\_Server, 55, 56

`selectInputDT_UI`, [55](#), [56](#)  
`shinyInput`, [57](#)  
`standardizeArgument_colnames`, [57](#)  
`standardizeArgument_editable`, [58](#)  
  
`whereSQL`, [59](#)