

Package: whence (via r-universe)

October 25, 2026

Type Package

Title Explain Where a Number Came from in 'shiny' Apps

Version 0.1.0.9000

Description Lets the viewer of a 'shiny' dashboard click a number, a row of a summary table or a bar of a 'ggplot2' plot, and see where it came from. The pipeline that makes the result is split into its steps. Each step is run in turn, and the number of rows after each step is recorded. The result is an attrition waterfall that shows how many rows each step removed, and what share of the rows that was, together with code that reproduces the result. The steps are found from the expression alone, so the pipeline does not need to be rewritten. The rows that contribute to a result, and the rows that a step removed, are found only when the viewer asks for them.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports htmltools, rlang, shiny (>= 1.7.0), utils

Suggests arrow, DBI, dbplyr, dplyr, dtplyr, duckplyr, ggplot2, reactable, RSQLite, shinytest2, testthat (>= 3.0.0), tidyr, withr

URL <https://github.com/tenmeh/whence>, <https://tenmeh.github.io/whence/>

BugReports <https://github.com/tenmeh/whence/issues>

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

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

Date/Publication 2026-10-25 09:07:11 UTC

RemoteUrl <https://github.com/tenmeh/whence>
RemoteRef HEAD
RemoteSha e25c7f0e6616896c906ce008268796dbfab77342

Contents

whence_group	2
whence_opened	3
whence_rows	4
whence_table	5
whence_trace	6
whence_value	8
whenceOutput	9
whencePlotOutput	11
whenceTableOutput	13
Index	15

whence_group	<i>Explain one row of a summary table</i>
--------------	---

Description

whence_group() takes a table from [whence_table\(\)](#) and the number of a row. It gives a trace of that row: for each step, how many rows of its group were left, and how many the step removed. Use [whence_rows\(\)](#) on it for the rows of the group, or for the rows of the group that one step removed.

Usage

whence_group(table, ..., row = NULL)

Arguments

- | | |
|-------|--|
| table | A whence_table from whence_table() . |
| ... | The values of the group columns of the row, by name, such as region = "APAC". They must find exactly one row. A single number without a name is the number of the row. |
| row | The number of a row of the table. Give row, or the group values in ... |

Details

The counts come from a replay of the steps with a hidden id column. After a step that breaks the link between a row and the source, such as distinct(), the counts of the group are not known, and the trace says so. The reproduce code is the code of the table with a last filter() that keeps this row, so it gives the row as the table has it.

Value

A whence_group object, which is also a whence_trace. Its value is the row, as a data frame with one row.

Examples

```
library(dplyr, warn.conflicts = FALSE)

by_cyl <- whence_table(
  mtcars |>
  filter(hp > 90) |>
  count(cyl)
)
six <- whence_group(by_cyl, 2)
six
whence_rows(six)

# The same row, by its group
whence_group(by_cyl, cyl = 6)
```

whence_opened

The panel that the viewer has open

Description

whence_opened() lets the app react when the viewer opens the panel of an output: for example, to filter other outputs to the row of a table that the viewer is looking at. Call it in a server function or a module server. It gives a reactive expression.

Usage

```
whence_opened(outputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

outputId	The id of a whence output, as in the UI. In a module, give the id without the namespace, as for output\$id.
session	The session. The default is the current one.

Details

The reactive gives NULL while no panel of that output is open. While one is open, it gives a list:

- row: the number of the row of the table, for [whenceTableOutput\(\)](#) and [whencePlotOutput\(\)](#). NULL for [whenceOutput\(\)](#), and for the panel of a whole table.
- data: that row of the table, as a data frame with one row. For [whenceOutput\(\)](#), the value. NULL for the panel of a whole table.

Only one panel is open at a time.

Value

A reactive expression.

Examples

```
if (interactive() && requireNamespace("dplyr", quietly = TRUE)) {
  library(shiny)
  library(dplyr, warn.conflicts = FALSE)

  ui <- fluidPage(
    whenceTableOutput("by_cyl", label = "Cars by cylinders"),
    textOutput("looking_at")
  )
  server <- function(input, output, session) {
    output$by_cyl <- renderWhenceTable(
      mtcars |> group_by(cyl) |> summarise(cars = n())
    )
    opened <- whence_opened("by_cyl")
    output$looking_at <- renderText({
      req(opened())
      paste("The viewer is looking at", opened()$data$cyl, "cylinders.")
    })
  }
  shinyApp(ui, server)
}
```

whence_rows

The rows that make the value, or the rows that a step removed

Description

whence_rows() finds rows of the source data. With no step, it gives the rows that contribute to the value: for `sales |> filter(...) |> summarise(total = sum(x))` these are the rows of sales that the filters keep. With `step = k`, it gives the rows that step `k` removed. The steps are numbered as in the `step` column of `as.data.frame(trace)`, where the source is step 0.

Usage

```
whence_rows(trace, n = NULL, step = NULL)
```

Arguments

trace	A whence_trace from <code>whence_trace()</code> or <code>whence_table()</code> , or a group from <code>whence_group()</code> .
n	The largest number of rows to give, or NULL for all of them.
step	NULL for the rows that make the value, or the number of a step for the rows that it removed.

Details

For a group from `whence_group()`, both are limited to the rows of that group. After a join, contributing rows are rows of the main source (x) only. Repeated matches return each source row once. Rows from y are not shown.

With `lazy = "count"`, a lazy source has no row attribution until a materialising step such as `collect()` returns a data frame. The identity pass starts from that data frame and follows only later steps. Returned rows are rows of that collected data frame, not rows of the remote source. Requests for earlier steps (or the materialising step itself) give NULL with a `whence_rows_unavailable` warning. Without a materialising step, rows are unavailable, even though the final value or table is collected.

The function replays the pipeline with a hidden id column on a copy of the source. `whence_trace()` never does this, so the waterfall costs nothing extra. The data that you give is never changed.

Some steps destroy the link between a row and the source: `distinct()`, `pivot_longer()`, `pivot_wider()`, and each step that is not a known verb. After such a step the function gives NULL with a warning that names the step. The rows that an earlier step removed can still be found. The waterfall of the trace is always complete, because it uses counts only.

Value

Rows of the source data, in their original order, with the columns of the source. NULL, with a warning of class `whence_rows_unavailable`, when the rows cannot be found.

Examples

```
library(dplyr, warn.conflicts = FALSE)

trace <- whence_trace(
  mtcars |>
  filter(cyl == 4) |>
  filter(mpg > 30) |>
  summarise(mean_hp = mean(hp))
)
whence_rows(trace)

# The cars that the second filter removed
whence_rows(trace, step = 2)
```

whence_table

Trace where the rows of a summary table came from

Description

`whence_table()` is `whence_trace()` for a pipeline that ends in a table, such as `group_by(region) |> summarise(total = sum(revenue))`. The trace covers the table as a whole. `whence_group()` then explains one row of it: how many rows of its group each step removed, the code that makes that row, and the rows behind it.

Usage

```
whence_table(
  expr,
  env = parent.frame(),
  lazy = getOption("whence.lazy", "stop")
)
```

Arguments

expr	A pipeline that ends in a data frame.
env	The environment in which to evaluate expr.
lazy	How to handle lazy tables: "stop" or "count", with no partial matching. Defaults to getOption("whence.lazy", "stop").

Details

The rows of the table are the groups of its last summary step (`summarise()`, `count()` or `tally()`). A row can be explained when every group column of that step is still in the table.

Value

A `whence_table` object, which is also a `whence_trace`. Its value is the table. Its field `keys` names the group columns, or is `NULL` when the rows cannot be explained one by one.

Examples

```
library(dplyr, warn.conflicts = FALSE)

by_cyl <- whence_table(
  mtcars |>
  filter(hp > 90) |>
  group_by(cyl) |>
  summarise(mpg = mean(mpg))
)
by_cyl
whence_group(by_cyl, 2)
```

whence_trace

Trace where a number came from

Description

`whence_trace()` takes a pipeline that ends in a single value, and runs it one step at a time. It records the number of rows after each step. The result tells you how many rows each step removed, and gives code that makes the value again.

Usage

```
whence_trace(
  expr,
  env = parent.frame(),
  lazy = getOption("whence.lazy", "stop")
)
```

Arguments

expr	A pipeline that ends in a single value: a vector of length one, or a data frame with one row and one column.
env	The environment in which to evaluate expr.
lazy	How to handle lazy tables: "stop" or "count", with no partial matching. Defaults to getOption("whence.lazy", "stop").

Details

The pipeline is split by its verbs. A call to a known verb, such as `filter()` or `summarise()`, is a step, and its first argument is the data that comes from upstream. This works the same for the native pipe `|>`, for the magrittr pipe `%>%` and for nested calls. The chain stops at the first call that is not a known verb, such as `full_join()`. That call and all that is upstream of it become the source.

Each step is evaluated in `env`, as the full expression would be. A step thus sees the same variables, and in 'shiny' it reads the same inputs and reactive values.

Step calls are evaluated once, in order. An `inner_join()` also runs an `anti_join()` to count unmatched rows. Repeated join matches are counted separately from removed rows in the integer added column (NA for other verbs).

Value

A `whence_trace` object. Use `whence_value()`, `whence_code()`, `whence_rows()`, `as.data.frame()` and `print()` with it.

The object is a list. These fields are stable, and you can read them:

- `value`: the value, as `whence_value()` gives it.
- `code`: the reproduce code, as `whence_code()` gives it.
- `traced`: TRUE when the pipeline was split into steps.
- `stages`: a data frame with one row for each step, as `as.data.frame()` gives it.
- `reason`: why the chain stopped short, as one sentence, or NULL when every step was traced.

Do not rely on the other fields. They can change in any version.

Limits

- Lazy tables from 'dbplyr', 'arrow', 'dtplyr', and prudent 'duckplyr' need `lazy = "count"` to count remote steps. With `lazy = "stop"`, the last materialising step becomes the source and only later steps are counted. A table that is still lazy gives an error in this mode. Default 'duckplyr' data frames work without opting in.

- `right_join()` and `full_join()` end the chain. They can bring in rows that are not from the main source.
- Native 'data.table' syntax is not traced.
- The native-pipe placeholder `_` is not supported. A pipeline that uses it is not split, and the value is still given.

Known verbs

These calls are steps: `filter()`, `mutate()`, `transmute()`, `select()`, `rename()`, `arrange()`, `slice()`, `slice_head()`, `slice_tail()`, `slice_min()`, `slice_max()`, `slice_sample()`, `group_by()`, `ungroup()`, `summarise()`, `summarize()`, `count()`, `tally()`, `distinct()`, `drop_na()`, `pivot_longer()`, `pivot_wider()`, `pull()`, `head()` and `tail()`. Joins `inner_join()`, `left_join()`, `semi_join()` and `anti_join()` are steps too. Materialising verbs `collect()`, `compute()`, `as_tibble()` and `as.data.frame()` keep rows and are hidden in the waterfall. All work with or without a namespace such as `dplyr::`. Any other call ends the chain, and the trace says where in its reason.

Lazy tables

Use `lazy = "count"` to trace a lazy pipeline without collecting every intermediate table. This runs one count query for the source and one for each lazy step, plus one for each `inner_join()` to count unmatched rows. Local steps need no count query. The final result is collected to give the value or table. The reproduce code includes this final `collect()`. Count queries and the final collection can be costly on large tables. Each count can run the upstream query again.

Set `options(whence.lazy = "count")` to use this mode by default, including in the render functions. `lazy = "stop"` keeps the default behaviour. Row attribution starts only after a materialising step in the pipeline; it cannot identify rows of a remote source. See [whence_rows\(\)](#).

Examples

```
library(dplyr, warn.conflicts = FALSE)

trace <- whence_trace(
  mtcars |>
  filter(cyl == 4) |>
  filter(mpg > 25) |>
  summarise(mean_hp = mean(hp))
)
trace
as.data.frame(trace)
```

whence_value

Parts of a trace

Description

`whence_value()` gives the value that the pipeline made. `whence_code()` gives code that makes the value again.

Usage

```
whence_value(trace)
```

```
whence_code(trace)
```

Arguments

trace A whence_trace from `whence_trace()`.

Details

In the code, `input$name` and `input[["name"]]` are replaced by their values at the time of the trace. Nothing else is replaced: each other name is written as it is in the pipeline. A wider rule could replace a name that a verb reads as a column, and give wrong code.

Value

`whence_value()` gives a vector of length one. `whence_code()` gives a single string.

Examples

```
library(dplyr, warn.conflicts = FALSE)

input <- list(cyl = 6)
trace <- whence_trace(
  mtcars |>
  filter(cyl == input$cyl) |>
  summarise(n = n())
)
whence_value(trace)
cat(whence_code(trace))
```

whenceOutput

An output that explains where its number came from

Description

`whenceOutput()` shows a single value. The viewer can click the value (or press Enter or Space on it) to open a panel. The panel shows how many rows each step of the pipeline removed, and the code that makes the value again. A button in the panel shows the rows that make the value.

Usage

```
whenceOutput(outputId, label = NULL)

renderWhence(
  expr,
  format = NULL,
  lazy = getOption("whence.lazy", "stop"),
  env = parent.frame(),
  quoted = FALSE,
  outputArgs = list()
)
```

Arguments

<code>outputId</code>	The output id.
<code>label</code>	Text to show before the value, or <code>NULL</code> for none. The panel uses it in its heading too.
<code>expr</code>	A pipeline that ends in a single value.
<code>format</code>	A function that takes the value and gives the text to show. A formatter of 'scales' works, for example <code>scales::label_dollar()</code> , and so does <code>function(x) paste0(round(x), " kg")</code> . With <code>NULL</code> , a number gets a comma for each thousand, and any other value goes through as <code>.character()</code> .
<code>lazy</code>	How to handle lazy tables: "stop" or "count", with no partial matching. Defaults to <code>getOption("whence.lazy", "stop")</code> .
<code>env</code>	The environment in which to evaluate <code>expr</code> .
<code>quoted</code>	Is <code>expr</code> already quoted? Use <code>TRUE</code> when you give an expression made with <code>quote()</code> .
<code>outputArgs</code>	A list of arguments for <code>whenceOutput()</code> , such as <code>label</code> , when the render is used in an R Markdown document with <code>runtime: shiny</code> .

Details

`renderWhence()` makes the value. Give it a pipeline that ends in a single value, as for [whence_trace\(\)](#). The pipeline can be inside braces, as in `renderWhence({ ... })`.

Value

`whenceOutput()` gives an HTML tag for the UI. `renderWhence()` gives a render function for the server.

The panel

The panel is a popover next to the value. On a screen narrower than 576 pixels it is a sheet at the bottom of the screen. Escape and a click outside it close it, and the focus then goes back to the value. The panel and the waterfall have no colours of their own: they use `currentColor` and the variables of the Bootstrap theme.

These CSS classes are stable, and you can style them: `.whence-value` (the value), `.whence-panel` (the panel), `.whence-backdrop`, `.whence-waterfall` (the list of steps), `.whence-delta` (a count of removed rows) and `.whence-code` (the reproduce code).

Reactivity

Each step runs in the environment of the render. It reads inputs and reactive values as the full expression would, so the output updates when they change. `format` runs on the server too, so it can read inputs, for example to choose a currency.

Examples

```
whenceOutput("total", label = "Total revenue")

if (interactive() && requireNamespace("dplyr", quietly = TRUE)) {
  library(shiny)
  library(dplyr, warn.conflicts = FALSE)

  ui <- fluidPage(
    selectInput("cyl", "Cylinders", c(4, 6, 8)),
    whenceOutput("mean_mpg", label = "Mean mpg")
  )
  server <- function(input, output, session) {
    output$mean_mpg <- renderWhence(
      mtcars |>
        filter(cyl == input$cyl) |>
        summarise(mpg = mean(mpg)),
      format = function(x) paste(round(x, 1), "mpg")
    )
  }
  shinyApp(ui, server)
}
```

whencePlotOutput

A plot whose bars and points explain where they came from

Description

`whencePlotOutput()` shows a plot of a summary table. The viewer clicks a bar or a point to open the panel of that row of the table: how many rows of its group each step removed, the code that makes the row, and the rows of the group. Under the plot, "Show the data" gives the same table, so a keyboard user can open each panel too.

Usage

```
whencePlotOutput(outputId, label = NULL, width = "100%", height = "400px")

renderWhencePlot(
  expr,
```

```

plot,
format = NULL,
width = "auto",
height = "auto",
res = 72,
alt = NA,
lazy = getOption("whence.lazy", "stop"),
env = parent.frame(),
quoted = FALSE,
outputArgs = list()
)

```

Arguments

outputId	The output id.
label	Text to show before the value, or NULL for none. The panel uses it in its heading too.
width, height	The size of the plot, as for shiny::plotOutput() .
expr	A pipeline that ends in a summary table.
plot	A function that takes the summary table and gives a plot.
format	As for renderWhenceTable() : for the table under the plot, and the panels.
res, alt	As for shiny::renderPlot() .
lazy	How to handle lazy tables: "stop" or "count", with no partial matching. Defaults to <code>getOption("whence.lazy", "stop")</code> .
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Use TRUE when you give an expression made with <code>quote()</code> .
outputArgs	A list of arguments for <code>whenceOutput()</code> , such as label, when the render is used in an R Markdown document with runtime: shiny.

Details

`renderWhencePlot()` makes the plot. `expr` is a pipeline that ends in a summary, as for [whence_table\(\)](#). `plot` is a function that takes that summary and gives a 'ggplot2' plot. A click finds its row through the aesthetics of the plot: on a discrete x axis, by the bar under the click; otherwise, by the nearest point. The x (and y) aesthetics must be columns of the summary.

Value

`whencePlotOutput()` gives an HTML tag for the UI. `renderWhencePlot()` gives a render function for the server.

Examples

```
whencePlotOutput("by_cyl", label = "Mean mpg by cylinders")
```

```

if (interactive() && requireNamespace("dplyr", quietly = TRUE) &&
    requireNamespace("ggplot2", quietly = TRUE)) {
  library(shiny)
  library(dplyr, warn.conflicts = FALSE)
  library(ggplot2)

  ui <- fluidPage(whencePlotOutput("by_cyl", label = "Mean mpg by cylinders"))
  server <- function(input, output, session) {
    output$by_cyl <- renderWhencePlot(
      mtcars |>
        filter(hp > 90) |>
        group_by(cyl = factor(cyl)) |>
        summarise(mpg = mean(mpg)),
      plot = function(data) ggplot(data, aes(cyl, mpg)) + geom_col()
    )
  }
  shinyApp(ui, server)
}

```

whenceTableOutput	<i>A summary table whose rows explain where they came from</i>
-------------------	--

Description

`whenceTableOutput()` shows a table, such as the result of `group_by(region) |> summarise(total = sum(revenue))`. The viewer clicks the first cell of a row (or presses Enter or Space on it) to open the panel of that row: how many rows of its group each step removed, the code that makes the row, and the rows of the group. A link under the table opens the panel of the table as a whole.

Usage

```
whenceTableOutput(outputId, label = NULL)
```

```

renderWhenceTable(
  expr,
  format = NULL,
  lazy = getOption("whence.lazy", "stop"),
  env = parent.frame(),
  quoted = FALSE,
  outputArgs = list()
)

```

Arguments

<code>outputId</code>	The output id.
<code>label</code>	Text to show before the value, or NULL for none. The panel uses it in its heading too.
<code>expr</code>	A pipeline that ends in a summary table.

format	NULL, a function for every number column that is not a group column, or a named list of functions, one for each column to change, such as <code>list(total = scales::label_dollar())</code> .
lazy	How to handle lazy tables: "stop" or "count", with no partial matching. Defaults to <code>getOption("whence.lazy", "stop")</code> .
env	The environment in which to evaluate <code>expr</code> .
quoted	Is <code>expr</code> already quoted? Use TRUE when you give an expression made with <code>quote()</code> .
outputArgs	A list of arguments for <code>whenceOutput()</code> , such as <code>label</code> , when the render is used in an R Markdown document with <code>runtime: shiny</code> .

Details

`renderWhenceTable()` makes the table. Give it a pipeline that ends in a summary, as for `whence_table()`. The table itself is sent with the render. The panel of a row needs one trip to the server, because it replays the steps for its group.

The rows can be explained when the group columns of the last summary step are in the table. The table shows at most 500 rows.

Value

`whenceTableOutput()` gives an HTML tag for the UI. `renderWhenceTable()` gives a render function for the server.

Examples

```
whenceTableOutput("by_region", label = "Revenue by region")

if (interactive() && requireNamespace("dplyr", quietly = TRUE)) {
  library(shiny)
  library(dplyr, warn.conflicts = FALSE)

  ui <- fluidPage(whenceTableOutput("by_cyl", label = "Mean mpg by cylinders"))
  server <- function(input, output, session) {
    output$by_cyl <- renderWhenceTable(
      mtcars |>
        filter(hp > 90) |>
        group_by(cyl) |>
        summarise(mpg = mean(mpg)),
      format = list(mpg = function(x) format(round(x, 1)))
    )
  }
  shinyApp(ui, server)
}
```

Index

`renderWhence` (`whenceOutput`), [9](#)
`renderWhencePlot` (`whencePlotOutput`), [11](#)
`renderWhenceTable` (`whenceTableOutput`),
[13](#)
`renderWhenceTable()`, [12](#)

`shiny::plotOutput()`, [12](#)
`shiny::renderPlot()`, [12](#)

`whence_code` (`whence_value`), [8](#)
`whence_code()`, [7](#)
`whence_group`, [2](#)
`whence_group()`, [4](#), [5](#)
`whence_opened`, [3](#)
`whence_rows`, [4](#)
`whence_rows()`, [2](#), [7](#), [8](#)
`whence_table`, [5](#)
`whence_table()`, [2](#), [4](#), [12](#), [14](#)
`whence_trace`, [6](#)
`whence_trace()`, [4](#), [5](#), [9](#), [10](#)
`whence_value`, [8](#)
`whence_value()`, [7](#)
`whenceOutput`, [9](#)
`whenceOutput()`, [3](#)
`whencePlotOutput`, [11](#)
`whencePlotOutput()`, [3](#)
`whenceTableOutput`, [13](#)
`whenceTableOutput()`, [3](#)