# Package: queryr (via r-universe)

October 24, 2024

**Title** Data Validation Queries With Tidy Output

**Version** 0.1.0.9000

**Description** Data validation queries with tidy, stackable output.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Depends** R (>= 2.10)

**Imports** dplyr, rlang

**Suggests** testthat, covr

**Repository** https://epicentre-msf.r-universe.dev

**RemoteUrl** https://github.com/epicentre-msf/queryr

**RemoteRef** HEAD

**RemoteSha** 95b0255f76361ad7ea3796891d9a45a9c11575ef

## Contents

## `ll` *Example dataset, an epidemiological linelist from a treatment centre*

### Description

Example dataset, an epidemiological linelist from a treatment centre

### Usage

```
ll
```

### Format

A data.frame with 12 rows and 10 variables:

**id** Patient identifier

**site** Site identifier

**age** Patient age in years

**status** Patient status

**date_onset** Date of symptom onset

**date_admit** Date of admission to treatment centre

**date_lab** Date of laboratory test

**lab_result** Result of laboratory test

**date_exit** Date of exit from treatment centre

**outcome** Patient outcome

## `ll_queries` *Example set of queries to run on* `ll` *using* `query_vec`

### Description

Example set of queries to run on `ll` using `query_vec`

### Usage

```
ll_queries
```

### Format

A data.frame with 5 rows and 2 variables:

**query_id** Query IDs

**query** Query expressions in string format `query` *Data validation queries with tidy, stackable output*

## Description

Find observations within a data frame matching a given query (a logical expression relating to one or more variables), and return tidy output that can be stacked across different queries on different variables. Stackability is achieved by pivoting the columns indicated in the query expression to long-form, e.g. "variable1", "value1", "variable2", "value2", ...

The query expression can optionally incorporate up to two dot-selectors (".x" and ".y"), which each refer to a *set* of variables specified separately using tidy-selection (see section **Using a dot-selector**). If both selectors are used in a given query expression, the sets of variables they respectively match can either be "crossed" such that all combinations are evaluated, or evaluated in parallel.

By default, only the data columns referenced in the query expression are returned, but additional columns can optionally be added with argument `cols_base`.

## Usage

```
query(
  data,
  cond,
  cols_dotx,
  cols_doty,
  crossed = FALSE,
  cols_base,
  pivot_long = TRUE,
  pivot_var = "variable",
  pivot_val = "value",
  as_chr = TRUE,
  count = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame |
| `cond` | An expression to evaluate with respect to variables within `data`. Can specify multiple variables using a dot-selector (".x" and ".y") within the expression (e.g. `.x > 0`) and then separately specifying the columns that the selector refers to with arguments `cols_dotx`/`cols_doty`. |
| `cols_dotx, cols_doty` | Tidy-selection of one or more columns represented by a .x or .y selector. Only used if `cond` contains the relevant selector. See section **Using a dot-selector** below. |
| `crossed` | if `cond` contains both a .x and .y selector, should the variables matched by `cols_dotx` and `cols_doty` be "crossed" such that all combinations are evaluated (`TRUE`), or should they be evaluated in parallel (`FALSE`). The latter requires |

| | |
|---|---|
| | that the number of variables matched by `cols_dotx` and `cols_doty` is the same. Defaults to `FALSE`. |
| `cols_base` | (Optional) Tidy-selection of other columns within `data` to retain in the output. Can optionally be set for an entire session using option "queryr_cols_base", e.g. `options(queryr_cols_base = quote(id:site))`. |
| `pivot_long` | Logical indicating whether to pivot the variables referenced within cond to a long (i.e. stackable) format, with default column names "variable1", "value1", "variable2", "value2", ... Defaults to `TRUE`. |
| `pivot_var` | Prefix for pivoted variable column(s). Defaults to "variable". Only used if `pivot_long = TRUE`. |
| `pivot_val` | Prefix for pivoted value column(s). Defaults to "value". Only used if `pivot_long = TRUE`. |
| `as_chr` | Logical indicating whether to coerce the columns referenced in the query expression cond to character prior to returning. This enables row-binding multiple queries with variables of different classes, but is only important if `pivot_long = TRUE`. Defaults to `TRUE`. |
| `count` | Logical indicating whether to summarize the output by counting the number of unique combinations across all returned columns (with count column "n"). Defaults to `FALSE`. |

## Value

A data frame reflecting the rows of `data` that match the given query. Returned columns include:

- (optional) columns matched by argument `cols_base`
- columns referenced within the query expression (pivoted to long form by default)
- (optional) count column "n" (if `count` = TRUE)

## Using a dot-selector

A query expression can optionally incorporate up to two dot-selectors ("`.x`" and "`.y`"), which each refer to a *set* of variables specified separately using tidy-selection (arguments `cols_dotx` and `cols_doty`).

When cond contains a `.x` selector, the query expression is evaluated repeatedly with each relevant variable from `cols_dotx` individually substituted into the `.x` position of the expression. The results of these multiple 'subqueries' are then combined with `dplyr::bind_rows`.

If cond contains both a .x and .y selector, the sets of variables matched by `cols_dotx` and `cols_doty` respectively can either be "crossed" such that all combinations are evaluated, or evaluated in parallel. Evaluating in parallel requires that the number of variables matched by `cols_dotx` and `cols_doty` is the same.

Consider a hypothetical query checking that, if a patient has a particular symptom, the date of onset of that symptom is not missing. E.g.

```
cond = .x == "Yes" & is.na(.y)
cols_dotx = c(symptom_fever, symptom_headache)
cols_doty = c(date_symptom_fever, date_symptom_headache)
```

If argument `crossed` is `FALSE`, the relevant variables from `cols_dotx` and `cols_doty` will be evaluated in parallel, as in:
`has_symptom_fever == "Yes" & is.na(date_symptom_fever)`
`has_symptom_headache == "Yes" & is.na(date_symptom_headache)`

Conversely, if argument `crossed` is `TRUE`, all combinations of the relevant variables will be evaluated, which for this particular query wouldn't make sense:
`symptom_fever == "Yes" & is.na(date_symptom_fever)`
`symptom_fever == "Yes" & is.na(date_symptom_headache) # not relevant`
`symptom_headache == "Yes" & is.na(date_symptom_fever) # not relevant`
`symptom_headache == "Yes" & is.na(date_symptom_headache)`

Note that if a dot-selector is used with argument `pivot_long = FALSE`, the row-binding of multiple subqueries may result in a sparse output with respect to the variables represented by the dot-selector, because for each subquery only the columns matched by expression cond are returned.

## Examples

```
# load example dataset, an epidemiological 'linelist'
data(ll)

# find observations where date_exit is earlier than date_admit
query(
  ll,
  date_exit < date_admit,
  cols_base = id:site
)

# find any date value in the future using a .x column selector
query(
  ll,
  .x > Sys.Date(),
  cols_dotx = starts_with("date"),
  cols_base = id:site
)

# incorporate an external object into the query expression
lab_result_valid <- c("Positive", "Negative", "Inc.", NA)

query(
  ll,
  !lab_result %in% lab_result_valid,
  cols_base = id:site,
)
```

## `query2` *Data validation queries across two data frames*

## Description

Find observations matching a query that concerns two data frames, and return tidy, stackable output. Entails three steps:

1. separately query each of the two data frames using `query`
2. combine the resulting query outputs based on a given join type (semi, anti, left, or inner)
3. execute a third query on the joined output

Each of the query steps is optional — unspecified query expressions are replaced with `TRUE` such that all rows of the relevant input are returned.

## Usage

```
query2(
  data1,
  data2,
  cond1,
  cond2,
  cols_base1,
  cols_base2,
  join_type,
  join_by,
  cond3,
  pivot_long = TRUE,
  pivot_var = "variable",
  pivot_val = "value",
  as_chr = TRUE
)
```

## Arguments

| | |
|---|---|
| `data1` | Data frame to query (#1) |
| `data2` | Data frame to query (#2) |
| `cond1` | (Optional) Expression to evaluate with respect to `data1`. If missing will be set to `TRUE` to select all rows. |
| `cond2` | (Optional) Expression to evaluate with respect to `data2`. If missing will be set to `TRUE` to select all rows. |
| `cols_base1` | (Optional) Tidy-selection of other columns within `data1` to retain in the final output. Can be set for an entire session using option "queryr_cols_base", e.g. `options(queryr_cols_base = quote(id:site))`. |
| `cols_base2` | (Optional) Tidy-selection of other columns within `data2` to retain in the final output. |
| `join_type` | How to join the output from the two initial queries ("semi", "anti", "left", or "inner"). Based on dplyr `join` types. |
| `join_by` | A character vector of variables to join by. If the join key columns have different names in `data1` and `data2`, use a named vector. For example, `by = c("a" = "b")` will match `data1$a` to `data2$b`. |

| | |
|---|---|
| `cond3` | (Optional) Expression to evaluate with respect to the joined output of the two initial queries. If missing will be set to `TRUE` to select all rows.<br>Note that if `join_type` is a filtering join ("anti" or "semi"), only variables from `data1` can be referenced in `cond3` (referencing a variable that only exists in `data2` will result in an error).<br>If `join_type` is instead a mutating join ("left" or "inner"), all variables from `data1` and `data2` will be available to `cond3`, even if not otherwise referenced with `cond1`/`cond2` or `cols_base1`/`cols_base2`. |
| `pivot_long` | Logical indicating whether to pivot the variables referenced within the query expression(s) to a long (i.e. stackable) format, with default column names "variable1", "value1", "variable2", "value2", ... Defaults to `TRUE`. If `cond3` is specified and `pivot_long` is `TRUE`, the pivot happens only in the final query (i.e. `cond3`). |
| `pivot_var` | Prefix for pivoted variable column(s). Defaults to "variable". Only used if `pivot_long = TRUE`. |
| `pivot_val` | Prefix for pivoted value column(s). Defaults to "value". Only used if `pivot_long = TRUE`. |
| `as_chr` | Logical indicating whether to coerce the columns referenced in the query expression(s) to character prior to returning. This enables row-binding multiple queries with variables of different classes, but is only important if `pivot_long = TRUE`. Defaults to `TRUE`. |

## Value

A data frame reflecting the rows of `data1` that match the given query. Returned columns include:

- Columns matched by argument `cols_base1`
- Columns matched by argument `cols_base2` (only if join type is "left" or "inner")
- Columns referenced within the relevant condition statements (pivoted to long form by default).

  If the join type is a mutating join ("left" or "inner"), variables from `data1` or `data2` referenced in *any* of the condition statements (`cond1`, `cond2`, or `cond3`) will appear in the output. However, with a filtering join ("anti" or "semi") only variables from `data1` will appear in the output.

## Examples

```
# example datasets: two related epidemiological linelists
data(ll)  # ll from treatment center (all cases, confirmed and non-confirmed)
data(sll) # summary linelist (only confirmed/probable cases)

# find patients in ll that don't appear in sll
query2(
  ll,
  sll,
  cols_base1 = c(id, site, status),
  join_type = "anti",
  join_by = c("id" = "tc_id")
)
```

```
# find patients with different outcome status in ll vs sll
query2(
  ll,
  sll,
  cols_base1 = id:site,
  join_type = "inner",
  join_by = c("id" = "tc_id"),
  cond3 = status != sll_status
)
```

---

## query_list *Data validation queries across a list of data frames*

---

### Description

Find observations matching a query that concerns one or more data frames within a list of data frames, and return tidy, stackable output. Like `query` but enables query expressions that reference variables in multiple data frames.

If the query expression references variables from data frames (i.e. list elements) other than the focal element, the relevant variable(s) will be joined to the focal element before the query expression is evaluated, see arguments `join_type` and `join_by` below.

### Usage

```
query_list(
  x,
  cond,
  element,
  cols_base,
  join_type = "left",
  join_by,
  pivot_long = TRUE,
  pivot_var = "variable",
  pivot_val = "value",
  as_chr = TRUE
)
```

### Arguments

| | |
|---|---|
| `x` | A list of data frames |
| `cond` | Expression to evaluate with respect to one or more variables in one or more of the data frames within `x`. |
| `element` | Name or integer index of the focal list element of `x` for the given query. If the query expression `cond` references variables from list elements apart from `element`, the relevant variable(s) will be joined to `x[[element]]` before the |

| | |
|---|---|
| | query expression is evaluated, based on the `join_type` and `join_by` arguments described below. |
| `cols_base` | (Optional) Tidy-selection of other columns within `data` to retain in the output. Can optionally be set for an entire session using option "queryr_cols_base", e.g. `options(queryr_cols_base = quote(id:site))`. |
| `join_type` | If `cond` references variables within elements of `x` apart from `x[[element]]`, what type of join should be used to join the relevant elements? Options are "left" (the default) and "inner". Based on dplyr `join` types. |
| `join_by` | A character vector of variables to join by. If the join key columns have different names in `x[[element]]` and `x[[other]]`, use a named vector. For example, `join_by = c("a" = "b")` will match `x[[element]]$a` to `x[[other]]$b`. |
| `pivot_long` | Logical indicating whether to pivot the variables referenced within `cond` to a long (i.e. stackable) format, with default column names "variable1", "value1", "variable2", "value2", ... Defaults to `TRUE`. |
| `pivot_var` | Prefix for pivoted variable column(s). Defaults to "variable". Only used if `pivot_long = TRUE`. |
| `pivot_val` | Prefix for pivoted value column(s). Defaults to "value". Only used if `pivot_long = TRUE`. |
| `as_chr` | Logical indicating whether to coerce the columns referenced in the query expression `cond` to character prior to returning. This enables row-binding multiple queries with variables of different classes, but is only important if `pivot_long = TRUE`. Defaults to `TRUE`. |

## Value

A data frame reflecting the rows of `x[[element]]` that match the given query. Returned columns include:

- (optional) columns matched by argument `cols_base`
- columns referenced within the query expression (pivoted to long form by default)

## query_vec — *Data validation queries vectorized over multiple query expressions*

## Description

Data validation queries with `query` or `query_list`, but vectorized over a set of query expressions in string format (and optionally a corresponding vector of query names/IDs). Results of the multiple queries are stacked and returned in a single tidy data frame, with columns referenced in the query expressions pivoted to long-form (e.g. "variable1", "value1", "variable2", "value2", ...).

## Usage

```
query_vec(
  x,
  cond,
  element,
  name,
  cols_base,
  name_col = "query_id",
  join_type = "left",
  join_by = NULL,
  pivot_var = "variable",
  pivot_val = "value",
  as_chr = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | A data frame or a list of data frames to query. If a single data frame will vectorize with `query`, whereas given a list of data frames will use `query_list`. |
| `cond` | Character vector of expressions to evaluate with respect to variables within `x`. |
| `element` | If `x` is a list of data frames, the names or integer indexes of the focal list element of `x` corresponding to each query expression (i.e. each element of `cond`). Only used if `x` is a list of data frames (see `query_list`). |
| `name` | (Optional) Character vector giving query names/IDs for each of the expressions within `cond`. If missing the expressions themselves (in string format) are used as names. |
| `cols_base` | (Optional) Tidy-selection of other columns within `x` (or `x[[element]]`) to retain in the final output. Can be set for an entire session using option "queryr_cols_base", e.g. `options(queryr_cols_base = quote(id:site))`. |
| `name_col` | Column name for the query names/IDs. Defaults to "query_id". |
| `join_type` | If `x` is a list of data frames and `cond` references variables within elements of `x` apart from `x[[element]]`, what type of join should be used to join the relevant elements? Options are "left" (the default) and "inner". Based on dplyr `join` types. Can specify different join types for different query expressions by passing a vector the same length as `cond`. |
| `join_by` | A character vector of variables to join by, or list of vectors the same length as `cond`. If the join key columns have different names in `x[[element]]` and `x[[other]]`, use a named vector. For example, `join_by = c("a" = "b")` will match `x[[element]]$a` to `x[[other]]$b`. Can specify different join columns for different query expressions by passing a *list* of vectors the same length as `cond`. |
| `pivot_var` | Prefix for pivoted variable column(s). Defaults to "variable". |
| `pivot_val` | Prefix for pivoted value column(s). Defaults to "value". |
| `as_chr` | Logical indicating whether to coerce the columns referenced in the query expression(s) to character prior to returning. This enables row-binding multiple queries with variables of different classes. Defaults to `TRUE`. |

### Value

A data frame reflecting the rows of `data` that match the given queries. Returned columns include:

- query name/ID column (name taken from argument `name_col`)
- (optional) columns matched by argument `cols_base`
- columns referenced within the query expressions, pivoted to long form

### See Also

`query`

### Examples

```
data(ll)          # example dataset, an epidemiological linelist
data(ll_queries)  # example data frame defining queries to run on ll

# run all queries defined in ll_queries
query_vec(
  ll,
  cond = ll_queries$query,
  name = ll_queries$query_id,
  cols_base = c(id, site)
)
```

---

## `sll` *Example dataset, a summary epidemiological linelist containing only confirmed/probable cases*

---

### Description

Example dataset, a summary epidemiological linelist containing only confirmed/probable cases

### Usage

```
sll
```

### Format

A data.frame with 10 rows and 8 variables:

**sll_id** Patient identifier in the summary linelist
**tc_admit** Was patient admitted to a treatment centre?
**tc_id** Patient ID at treatment centre
**tc_site** Site of treatment centre
**sll_age** Date of symptom onset
**sll_status** Date of admission to hospital
**sll_date_outcome** Date of laboratory test
**sll_outcome** Patient outcome

# Index