Package 'cohortBuilder.db'

November 23, 2026

Type Package

Title Database Source Layer for 'cohortBuilder'

Version 1.0.0

Maintainer Krystian Igras <krystian8207@gmail.com>

Description Extends 'cohortBuilder' with a source layer for database tables. Cohort filtering steps are translated to 'dbplyr' queries and executed on the database server, so no data needs to be pulled into R. Supports the 'shinyCohortBuilder' graphical interface and offers configurable strategies for how intermediate step results are passed between filtering steps.

License MIT + file LICENSE

URL https://github.com/r-world-devs/cohortBuilder.db

BugReports https://github.com/r-world-devs/cohortBuilder.db/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports rlang, tibble, purrr, dplyr (>= 1.0.0), magrittr, glue, S7, DBI, dbplyr, cohortBuilder (>= 1.0.0), queryBuilder, shinyWidgets, shinyCohortBuilder (>= 1.0.0)

KeepSource true

Suggests shiny, shinyQueryBuilder, RSQLite, RPostgres, knitr, rmarkdown, withr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Krystian Igras [cre, aut]

Repository CRAN

Date/Publication 2026-11-23 14:50:02 UTC

Contents

cohortBuilder.db-package . . . 2
.available_filters_choices.db . . . 3
.filter_position.db . . . 4
.step_attrition.db . . . 4
autofilter.db . . . 5
create_data_object . . . 5
db-filter-date-range . . . 6
db-filter-datetime-range . . . 6
db-filter-discrete . . . 7
db-filter-discrete-text . . . 7
db-filter-multi-discrete . . . 8
db-filter-query . . . 8
db-filter-range . . . 9
dbtables . . . 9
rendering-filters . . . 10
set_source.db . . . 10
source-layer . . . 11
step-strategy . . . 13
updating-data-statistics . . . 13

Index **15**

`cohortBuilder.db-package`

cohortBuilder methods for database source

Description

Adds a ‘dbtables‘ source backend to ‘cohortBuilder‘, allowing filtering pipelines to run directly against database tables (via ‘dbplyr‘).

Details

The package plugs into the ‘cohortBuilder‘ S7 filter generics (‘cb_filter_data()‘, ‘cb_get_filter_stats()‘, ...) and the S3 source-layer generics (‘.init_step()‘, ‘.pre_filtering()‘, ...) to describe how each filter type and source operation behaves for a ‘"db"‘ source.

The following filter types are supported: ‘discrete‘, ‘discrete_text‘, ‘range‘, ‘date_range‘, ‘datetime_range‘, ‘multi_discrete‘ and ‘query‘.

Options

- ‘cb_db_query_max_values‘ - maximum number of distinct values collected per column when computing ‘query‘ filter specifications (500 by default), see [db-filter-query].

Author(s)

Maintainer: Krystian Igras <krystian8207@gmail.com>

Authors:

- Krystian Igras <krystian8207@gmail.com>

See Also

Useful links:

- https://github.com/r-world-devs/cohortBuilder.db
- Report bugs at https://github.com/r-world-devs/cohortBuilder.db/issues

---

`.available_filters_choices.db` *Generate available filters choices based on the db Source data*

Description

The method returns the available choices for the virtualSelect input.

Usage

```
## S3 method for class 'db'
.available_filters_choices(source, cohort, ...)
```

Arguments

`source`	Source object of db type.
`cohort`	cohortBuilder cohort object.
`...`	Extra arguments passed to a specific method.

Value

'shinyWidgets::prepare_choices' output value.

.filter_position.db *Create target filter position in UI panel for db source*

Description

Create target filter position in UI panel for db source

Usage

```
## S3 method for class 'db'
.filter_position(source, step_id, filter, ns, ...)
```

Arguments

source	Source object of db type.
step_id	Id of the step that the filter belongs to.
filter	Filter object.
ns	Namespace function.
...	Extra arguments passed to related method.

Value

A CSS selector string locating the filter's dataset group.

.step_attrition.db *Generate output of attrition plot for db source*

Description

Generate output of attrition plot for db source

Usage

```
## S3 method for class 'db'
.step_attrition(source, id, cohort, session, ...)
```

Arguments

source	Source object of db type.
id	Id of attrition output.
cohort	Cohort object.
session	Shiny session object.
...	Extra arguments passed to specific method.

Value

List of two objects: ‘render‘ and ‘output‘ defining rendering and output placeholder for the step attrition plot feature.

`autofilter.db` *Generate filters definition based on the db Source data*

Description

The method analyses the source data structure, generates filters based on the column types and attaches them to the source (as a step or as available filters metadata).

Usage

```
## S3 method for class 'db'
autofilter(source, attach_as = c("step", "meta"), ...)
```

Arguments

`source`	Source object of db type.
`attach_as`	Choose whether the filters should be attached as a new step, or as a list of available filters (‘"meta"‘, used in the filtering panel when ‘new_step = "configure"‘). By default attached as a ‘step‘.
`...`	Extra arguments passed to a specific method.

Value

Source object having step configuration attached.

`create_data_object` *Build the db data object (table connections) for a source*

Description

Returns a named list of ‘dbplyr‘ lazy table connections for every table in the ‘db‘ source. Exported so that it can be referenced from the reproducible code generated by ‘cohortBuilder‘ (the code shown for the ‘.init_step‘ stage).

Usage

```
create_data_object(source)
```

Arguments

`source`	A ‘db‘ source object (see [set_source.db()]).

Value

A named list of ‘dbplyr‘ table connections.

`db-filter-date-range` *Date range filter methods for the ‘db‘ source*

Description

S7 methods implementing the ‘date_range‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterDateRange, db_class)‘.

Details

Dates are frequently stored as numeric values in databases (e.g. SQLite), so collected min/max values are coerced back with [as.Date()].

Value

See [db-filter-discrete].

`db-filter-datetime-range`
Datetime range filter methods for the ‘db‘ source

Description

S7 methods implementing the ‘datetime_range‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterDatetimeRange, db_class)‘.

Details

Datetimes are commonly stored as numeric values in databases (e.g. SQLite keeps seconds since the epoch), so filter bounds are sent to the database as numbers.

Following the ‘tblist‘ implementation, ‘cb_get_filter_stats‘ reports ‘min‘, ‘max‘ and the frequency bounds as numeric seconds since the epoch, whereas ‘cb_domain_from_data‘ returns ‘POSIXct‘ values (coerced with [as.POSIXct()] in the ‘UTC‘ time zone).

Value

See [db-filter-discrete].

`db-filter-discrete` *Discrete filter methods for the ‘db‘ source*

Description

S7 methods implementing the ‘discrete‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterDiscrete, db_class)‘ and mirror the built-in ‘tblist‘ behaviour, but operate lazily on ‘dbplyr‘ table connections (statistics are pulled with ‘dplyr::collect()‘).

Value

- ‘cb_filter_data‘: the filtered ‘data_object‘. - ‘cb_get_filter_stats‘: a named list (or single value) of statistics. - ‘cb_get_filter_data‘: the filter-related column (lazy). - ‘cb_get_filter_defaults‘: a list of default parameter values. - ‘cb_filter_to_expr‘: an ‘rlang‘ expression for reproducible code. - ‘cb_domain_from_data‘: the observed distinct values.

Method arguments

- ‘filter‘: A ‘cohortBuilder‘ discrete filter (‘CbFilterDiscrete‘). - ‘source‘: A ‘db‘ source object. - ‘data_object‘: Named list of ‘dbplyr‘ table connections. - ‘name‘: Optional statistic name(s) to compute. - ‘cache_object‘: Cached statistics used to derive default parameter values. - ‘...‘: Extra arguments.

`db-filter-discrete-text` *Discrete text filter methods for the ‘db‘ source*

Description

S7 methods implementing the ‘discrete_text‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterDiscreteText, db_class)‘.

Value

See [db-filter-discrete].

`db-filter-multi-discrete` *Multi discrete filter methods for the ‘db‘ source*

Description

S7 methods implementing the ‘multi_discrete‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterMultiDiscrete, db_class)‘ and mirror the built-in ‘tblist‘ behaviour, but operate lazily on ‘dbplyr‘ table connections (statistics are pulled with ‘dplyr::collect()‘).

Value

See [db-filter-discrete].

`db-filter-query` *Query filter methods for the ‘db‘ source*

Description

S7 methods implementing the ‘query‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterQuery, db_class)‘ and mirror the built-in ‘tblist‘ behaviour, but operate lazily on ‘dbplyr‘ table connections (statistics are pulled with ‘dplyr::collect()‘).

Value

- ‘cb_filter_data‘: the filtered ‘data_object‘. - ‘cb_get_filter_stats‘: a named list (or single value) of statistics. - ‘cb_get_filter_data‘: the filter-related columns (lazy). - ‘cb_get_filter_defaults‘: a list of default parameter values. - ‘cb_filter_to_expr‘: an ‘rlang‘ expression for reproducible code.

Method arguments

- ‘filter‘: A ‘cohortBuilder‘ query filter (‘CbFilterQuery‘). - ‘source‘: A ‘db‘ source object. - ‘data_object‘: Named list of ‘dbplyr‘ table connections. - ‘name‘: Optional statistic name(s) to compute. - ‘cache_object‘: Cached statistics used to derive default parameter values. - ‘...‘: Extra arguments.

Limitations

‘queryBuilder::queryToExpr()‘ renders some operators as R functions that have no SQL translation. Those are rewritten to database friendly equivalents, which makes two operator groups behave slightly differently than for ‘tblist‘:

- ‘contains‘, ‘begins_with‘ and ‘ends_with‘ are translated to SQL ‘LIKE‘, so the provided value is matched as a literal pattern and not as a regular expression. - ‘LIKE‘ case sensitivity is defined by the database backend (’SQLite’ matches ASCII case insensitively, ’PostgreSQL’ case sensitively).

db-filter-range *Range filter methods for the ‘db‘ source*

Description

S7 methods implementing the ‘range‘ filter for a ‘dbtables‘ source. They dispatch on ‘list(CbFilterRange, db_class)‘ and mirror the built-in ‘tblist‘ behaviour, but operate lazily on ‘dbplyr‘ table connections (min/max and frequency statistics are pulled with ‘dplyr::collect()‘).

Value

See [db-filter-discrete].

dbtables *Configure database tables connection*

Description

The object should be used as ‘dtconn‘ argument of set_source.

Usage

```
dbtables(connection, tables, schema = "public")
```

Arguments

connection	Database connection.
tables	Vector of tables names to use in the source.
schema	Name of the schema to connect to.

Value

Object of class ’db’.

Examples

```
## Not run:
conn <- DBI::dbConnect(RSQLite::SQLite(), dbname = ":memory:")
dbtables(conn, tables = c("books", "borrowers"), schema = "cb")

## End(Not run)
```

rendering-filters *Filters rendering method for db source*

Description

Filters rendering method for db source

Usage

```
## S3 method for class 'db'
.render_filters(source, cohort, step_id, ns, ...)
```

Arguments

source	Source object of db type.
cohort	Cohort object.
step_id	Id of the filtering step.
ns	Namespace function.
...	Extra arguments passed to a specific method.

Value

A ‘shiny.tag‘ containing the rendered filters panel.

set_source.db *Create db-based Source*

Description

This is a method for cohortBuilder [cohortBuilder::set_source()] generic. It creates Source object which is based on connection to database tables.

Usage

```
## S3 method for class 'db'
set_source(
  dtconn,
  primary_keys = NULL,
  binding_keys = NULL,
  source_code = NULL,
  description = NULL,
  ...
)
```

Arguments

dtconn	Object of class "db" defining connection to the database, schema and tables names. Should be created with dbtables.
primary_keys	Definition of data 'primary_keys'. See primary_keys.
binding_keys	Binding keys definition see binding-keys.
source_code	An expression presenting low-level code for creating source. When provided, used as a part of reproducible code output.
description	A named list storing the source objects description. Can be accessed with description Cohort method.
...	Source type specific parameters passed to set_source (e.g. 'available_filters', 'compute_meta_stats').

Value

Object of class 'c("db", "Source", "R6")'.

---

source-layer *Source compatibility methods for "db" layer.*

Description

These are methods for 'cohortBuilder' generics used for integrity between source layers. See source-layer.

Usage

```
## S3 method for class 'db'
.init_step(source, ...)

## S3 method for class 'db'
.pre_filtering(source, data_object, step_id)

## S3 method for class 'db'
.post_binding(source, data_object, step_id)

## S3 method for class 'db'
.collect_data(source, data_object)

## S3 method for class 'db'
.get_stats(source, data_object)

## S3 method for class 'db'
.run_binding(source, binding_key, data_object_pre, data_object_post, ...)

## S3 method for class 'db'
```

```
.get_attrition_label(source, step_id, step_filters, dataset, ...)

## S3 method for class 'db'
.get_attrition_count(source, data_stats, dataset, ...)

## S3 method for class 'db'
.propagate_domains(source, data_object, step_id, cohort, mode, ...)

## S3 method for class 'db'
.repro_code_tweak(source, code_data)
```

Arguments

`source`	Source object of "db" type.
`...`	Extra arguments passed to a specific method.
`data_object`	Object that allows source data access. For "db" source it's a list of connections to (temporary) tables in the connected database.
`step_id`	Id of the currently used step.
`binding_key`	Binding key definition. See binding-keys.
`data_object_pre, data_object_post`	Version of `data_object` before and after filtering in the step was applied.
`step_filters`	List of step filters.
`dataset`	Dataset name to present attrition for.
`data_stats`	Data frame presenting statistics for each filtering step.
`cohort`	Cohort object the step belongs to.
`mode`	Domain propagation mode, one of "filter", "stats" or "data". See cohort.
`code_data`	Data frame collecting the reproducible-code expressions.

Details

‘.pre_filtering.db‘ and ‘.post_binding.db‘ delegate to the strategy selected by the ‘cohortBuilder.db.step_strategy‘ option. See step-strategy.

Value

- ‘.init_step.db‘, ‘.pre_filtering.db‘, ‘.post_binding.db‘: a named list of ‘dbplyr‘ table connections (the ‘data_object‘). - ‘.collect_data.db‘: a named list of collected (in-memory) tibbles. - ‘.get_stats.db‘: a named list of per-dataset statistics (‘n_rows‘). - ‘.run_binding.db‘: the updated ‘data_object_post‘. - ‘.get_attrition_label.db‘ / ‘.get_attrition_count.db‘: attrition label and count (reused from the ‘tblist‘ implementation). - ‘.propagate_domains.db‘: invisibly ‘NULL‘; called for the side effect of narrowing the step filters domains (reused from the ‘tblist‘ implementation).

`step-strategy` *Step execution strategy*

Description

Controls how each filtering step's result is handed to the next step, that is what .pre_filtering.db and .post_binding.db do.

Details

The strategy is read from the `cohortBuilder.db.step_strategy` option on every step run, so it can be changed at any time with `options(cohortBuilder.db.step_strategy = "lazy")`.

Available strategies:

"lazy" (default) No materialization at all. `.post_binding` returns the lazy `dbplyr` query unchanged and `.pre_filtering` only resets the per-table "filtered" flag, so no temporary tables are ever created or dropped. Each downstream collect or count is pushed down to the database as a single query. Best when the cohort is shallow (few steps) or the source tables are too large to copy; the trade-off is deeper query nesting as filters and bindings accumulate. This is also the only strategy that works on databases without a `pg_temp` schema (e.g. Amazon Redshift), because nothing is dropped.

"materialize" The original behaviour. `.post_binding` writes every source table's result to a temporary table, so later steps read a materialized result instead of re-deriving it; `.pre_filtering` drops this step's temporary tables first via dbRemoveTable so a re-run can recreate them. The trade-off is a full table scan per source table on every run, even for tables no filter touched. Note that `dbRemoveTable(temporary = TRUE)` is emitted by `RPostgres` as `DROP TABLE pg_temp.<name>`, which fails on Redshift.

"materialize_selective" Like "materialize", but only tables whose row set actually changed in this step (a filter or an active binding touched them) are copied to a temporary table. Untouched tables are passed through unchanged, which points them at the previous step's data object. Temporary tables are dropped with a plain `DROP TABLE IF EXISTS`, so this strategy also works on Redshift.

An unset or unrecognised option falls back to "lazy" (with a warning in the latter case).

`updating-data-statistics`
Update data statistics method for db source

Description

Update data statistics method for db source

Usage

```
## S3 method for class 'db'
.update_data_stats(source, step_id, cohort, session, ...)
```

Arguments

source	Source object of db type.
step_id	Id of the filtering step.
cohort	Cohort object.
session	Shiny session object.
...	Extra arguments passed to a specific method.

Value

Invisibly ‘NULL‘; called for the side effect of updating the panel.

Index

.available_filters_choices.db, 3
.collect_data.db *(source-layer)*, 11
.filter_position.db, 4
.get_attrition_count.db *(source-layer)*, 11
.get_attrition_label.db *(source-layer)*, 11
.get_stats.db *(source-layer)*, 11
.init_step.db *(source-layer)*, 11
.post_binding.db, *13*
.post_binding.db *(source-layer)*, 11
.pre_filtering.db, *13*
.pre_filtering.db *(source-layer)*, 11
.propagate_domains.db *(source-layer)*, 11
.render_filters.db *(rendering-filters)*, 10
.repro_code_tweak.db *(source-layer)*, 11
.run_binding.db *(source-layer)*, 11
.step_attrition.db, 4
.update_data_stats.db *(updating-data-statistics)*, 13

autofilter.db, 5

binding-keys, *11, 12*

cohort, *12*
cohortBuilder.db *(cohortBuilder.db-package)*, 2
cohortBuilder.db-package, 2
create_data_object, 5

db-filter-date-range, 6
db-filter-datetime-range, 6
db-filter-discrete, 7
db-filter-discrete-text, 7
db-filter-multi-discrete, 8
db-filter-query, 8
db-filter-range, 9
dbRemoveTable, *13*
dbtables, 9, *11*
description, *11*

primary_keys, *11*

rendering-filters, 10

set_source, *9*, *11*
set_source.db, 10
source-layer, *11*, 11
step-strategy, *12*, 13

updating-data-statistics, 13