Package: glitter (via r-universe)

January 1, 2025

Type Package

Title glitter makes SPARQL

Version 0.2.999

Description This package aims at writing and sending SPARQL queries. It makes the exploration and use of Linked Open Data (Wikidata in particular) easier for those who do not know SPARQL.

License GPL-2

URL https://lvaudor.github.io/glitter/, https://github.com/lvaudor/glitter

BugReports https://github.com/lvaudor/glitter/issues

Depends R (>= 3.5.0)

Imports anytime, cli, dplyr, glue, httr2, lifecycle, magrittr, purrr, rlang, tibble, xml2, xmlparsedata

Suggests covr, httptest2, httpuv, knitr, rmarkdown, testthat (>= 3.1.10.9000), withr

Config/Needs/website DT, ggplot2, leaflet, lvaudor/sequins, sf, tidyr, WikidataR

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Remotes r-lib/testthat

Config/pak/sysreqs libxml2-dev libssl-dev

Repository https://ar-puuk.r-universe.dev

RemoteUrl https://github.com/lvaudor/glitter

RemoteRef HEAD

RemoteSha b7e7d51e90a4d53bf981296e7e6df19a35567144

Contents

`set_functions` *Correspondence between R-DSL functions and SPARQL functions/operators.*

Description

Correspondence between R-DSL functions and SPARQL functions/operators.

Usage

```
set_functions

term_functions

misc_functions

string_functions

numeric_functions
```

`datetime_functions`

`operators`

`all_correspondences`

Format

A data frame.

R R-DSL function

SPARQL SPARQL function

args list-column with R vs SPARQL argument names

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 21 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 9 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 12 rows and 3 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 4 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 6 rows and 2 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 66 rows and 3 columns.

`spq` *SPARQL escaping.*

Description

Like `dbplyr::spq()`.

Usage

```
spq(...)

is.spq(x)

as.spq(x)
```

Arguments

`...`	Character vectors that will be combined into a single SPARQL expression.
`x`	Object to coerce

spq_add *Add a triple pattern statement to a query*

Description

Add a triple pattern statement to a query

Usage

```
spq_add(
  .query = NULL,
  .triple_pattern = NULL,
  .subject = NULL,
  .verb = NULL,
  .object = NULL,
  .prefixes = NULL,
  .required = TRUE,
  .label = NA,
  .within_box = c(NA, NA),
  .within_distance = c(NA, NA),
  .filter = NULL,
  .sibling_triple_pattern = NA
)
```

Arguments

.query	query
.triple_pattern	the triple pattern statement (replaces arguments subject verb and object)
.subject	an anonymous variable (for instance, and by default, "?subject") or item (for instance "wd:Q456"))
.verb	the property (for instance "wdt:P190")
.object	an anonymous variable (for instance, and by default, "?object") or item (for instance "wd:Q456"))
.prefixes	Custom prefixes
.required	whether the existence of a value for the triple is required or not (defaults to TRUE). If set to FALSE, then other triples in the query are returned even if this particular triple is missing)
.label	**[Deprecated]** See spq_label().
.within_box	if provided, rectangular bounding box for the triple query. Provided as list(southwest=c(long=...,lat=...),no
.within_distance	if provided, circular bounding box for the triple query. Provided as list(center=c(long=...,lat=...), radius=...), with radius in kilometers. The center can also be provided as a variable (for instance, "?location") for the center coordinates to be retrieved directly from the query.

`.filter` Filter for the triple. Only use this with `.required=FALSE`

`.sibling_triple_pattern`
Triple this triple is to be grouped with, especially (only?) useful if the sibling triple is optional.

Details

The arguments `.subject`, `.verb`, `.object` are most useful for programmatic usage, they are actually used within glitter code itself.

Examples

```
# find the cities
spq_init() %>%
  spq_add("?city wdt:P31/wdt:P279* wd:Q486972") %>%
  spq_label(city) %>%
  spq_mutate(coords = wdt::P625(city),
          .within_distance=list(center=c(long=4.84,lat=45.76),
                                radius=5)) %>%
  spq_perform()

# find the individuals of the species
spq_init() %>%
  spq_add("?mayor wdt:P31 ?species") %>%
  # dog, cat or chicken
  spq_set(species = c('wd:Q144','wd:Q146', 'wd:Q780')) %>%
  # who occupy the function
  spq_add("?mayor p:P39 ?node") %>%
  # of mayor
  spq_add("?node ps:P39 wd:Q30185") %>%
  # of some places
  spq_add("?node pq:P642 ?place") %>%
  spq_perform()
```

---

spq_arrange *Arrange results by variable value*

Description

Arrange results by variable value

Usage

```
spq_arrange(.query, ..., .replace = FALSE)
```

Arguments

.query	a list with elements of the query
...	variables by which to arrange (or SPARQL strings escaped with spq(), or strings, see examples)
.replace	whether to replace the pre-existing arranging

Value

A query object

Examples

```
# descending length, ascending item_label, "R" syntax
spq_init() %>%
  spq_add("?item wdt:P31/wdt:P279* wd:Q4022") %>%
  spq_label(item) %>%
  spq_add("?item wdt:P2043 ?length") %>%
  spq_add("?item wdt:P625 ?location") %>%
  spq_arrange(desc(length), item_label) %>%
  spq_head(50)

# descending length, ascending item_label,
# "R" syntax with quotes e.g. for a loop
variable = "length"
spq_init() %>%
  spq_add("?item wdt:P31/wdt:P279* wd:Q4022") %>%
  spq_label(item) %>%
  spq_add("?item wdt:P2043 ?length") %>%
  spq_add("?item wdt:P625 ?location") %>%
  spq_arrange(sprintf("desc(%s)", variable), item_label) %>%
  spq_head(50)

# descending length, ascending item_label, SPARQL syntax
spq_init() %>%
  spq_add("?item wdt:P31/wdt:P279* wd:Q4022") %>%
  spq_label(item) %>%
  spq_add("?item wdt:P2043 ?length") %>%
  spq_add("?item wdt:P625 ?location") %>%
  spq_arrange(spq("DESC(?length) ?item_label")) %>%
  spq_head(50)

# descending xsd:integer(mort), R syntax
spq_init() %>%
  spq_add("?oeuvre dcterms:creator ?auteur") %>%
  spq_add("?auteur bio:death ?mort") %>%
  spq_add("?auteur foaf:familyName ?nom") %>%
  spq_filter(as.integer(mort) < as.integer("1924")) %>%
  spq_group_by(auteur, nom, mort) %>%
  spq_arrange(desc(as.integer(mort)))

# descending as.integer(mort), SPARQL syntax
```

```
spq_init() %>%
  spq_add("?oeuvre dcterms:creator ?auteur") %>%
  spq_add("?auteur bio:death ?mort") %>%
  spq_add("?auteur foaf:familyName ?nom") %>%
  spq_filter(as.integer(mort) < as.integer("1924")) %>%
  spq_group_by(auteur, nom, mort) %>%
  spq_arrange(spq("DESC(xsd:integer(?mort))"))

# Usage of the .replace argument
# .replace = FALSE (default)
spq_init() %>%
  spq_add("?item wdt:P31/wdt:P279* wd:Q4022") %>%
  spq_label(item) %>%
  spq_add("?item wdt:P2043 ?length") %>%
  spq_add("?item wdt:P625 ?location") %>%
  spq_arrange(desc(length)) %>%
  spq_arrange(location) %>%
  spq_head(50)

# .replace = TRUE
spq_init() %>%
  spq_add("?item wdt:P31/wdt:P279* wd:Q4022") %>%
  spq_label(item) %>%
  spq_add("?item wdt:P2043 ?length") %>%
  spq_add("?item wdt:P625 ?location") %>%
  spq_arrange(desc(length)) %>%
  spq_arrange(location, .replace = TRUE) %>%
  spq_head(50)

# Mixing syntaxes
spq_init() %>%
  spq_add("?item wdt:P31/wdt:P279* wd:Q4022") %>%
  spq_label(item) %>%
  spq_add("?item wdt:P2043 ?length") %>%
  spq_add("?item wdt:P625 ?location") %>%
  spq_arrange(desc(length), spq("?location")) %>%
  spq_head(50)
```

`spq_assemble` *Assemble query parts into a proper SPARQL query*

Description

Assemble query parts into a proper SPARQL query

Usage

```
spq_assemble(.query, strict = TRUE)
```

Arguments

.query	a list with elements of the query
strict	whether to perform some linting on the query, and error in case a problem is detected.

Value

A query object

Examples

```
spq_init() %>%
  spq_add("?city wdt:P31 wd:Q515") %>%
  spq_label(city, .languages = "fr$") %>%
  spq_add("?city wdt:P1082 ?pop") %>%
  spq_assemble() %>%
  cat()
```

---

spq_control_request *Create the request control object for* spq_init()

Description

Create the request control object for spq_init()

Usage

```
spq_control_request(
  user_agent = getOption("glitter.ua",
    "glitter R package (https://github.com/lvaudor/glitter)"),
  max_tries = getOption("glitter.max_tries", 3L),
  max_seconds = getOption("glitter.max_seconds", 120L),
  timeout = getOption("glitter.timeout", 1000L),
  request_type = c("url", "body-form"),
  rate = NULL,
  realm = NULL
)
```

Arguments

user_agent	a string indicating the user agent to send with the query.
max_tries, max_seconds	Cap the maximal number of attemps with max_tries or the total elapsed time from the first request with max_seconds.
timeout	maximum number of seconds to wait (httr2::req_timeout()).
request_type	a string indicating how the query should be sent: in the URL (url, default, most common) or as a body form (body-form).

rate	Maximum rate, i.e. maximum number of requests per second. Usually easiest expressed as a fraction, number_of_requests / number_of_seconds, e.g. 15 requests per minute is 15 / 60.
realm	An unique identifier that for throttle pool. If not supplied, defaults to the hostname of the request.

Value

A list to be used in spq_init()'s request_control argument.

Examples

```
# Defaults
spq_control_request()
# Tweaking values
spq_control_request(
  user_agent = "Jane Doe https://example.com",
  max_tries = 1L,
  max_seconds = 10L,
  timeout = 10L,
  request_type = "url"
)
```

---

spq_endpoint_info *Create the endpoint info object for* spq_init()

Description

Create the endpoint info object for spq_init()

Usage

```
spq_endpoint_info(label_property = "rdfs:prefLabel")
```

Arguments

label_property	Property used by the endpoint for labelling.

Value

A list to be used in spq_init()'s endpoint_info argument.

Examples

```
spq_endpoint_info(label_property = "skos:preflabel")
```

`spq_filter` *Filters results by adding conditions*

Description

Filters results by adding conditions

Usage

```
spq_filter(
  .query = NULL,
  ...,
  .label = NA,
  .within_box = c(NA, NA),
  .within_distance = c(NA, NA)
)
```

Arguments

`.query`	a list with elements of the query
`...`	variables by which to arrange (or SPARQL strings escaped with `spq()`, or strings, see examples)
`.label`	**[Deprecated]** See `spq_label()`.
`.within_box`	if provided, rectangular bounding box for the triple query. Provided as list(southwest=c(long=...,lat=...),no
`.within_distance`	if provided, circular bounding box for the triple query. Provided as list(center=c(long=...,lat=...), radius=...), with radius in kilometers. The center can also be provided as a variable (for instance, "?location") for the center coordinates to be retrieved directly from the query.

Value

A query object

Some examples

```
spq_init() %>%
spq_filter(item == wdt::P31(wd::Q13442814))

# Lexemes in English that match an expression
# here starting with "pota"
query <- spq_init() |>
  spq_prefix(prefixes = c(dct = "http://purl.org/dc/terms/")) |>
  spq_add(spq('?lexemeId dct:language wd:Q1860')) |>
  spq_mutate(lemma = wikibase::lemma(lexemeId)) |>
  spq_filter(str_detect(lemma, '^pota.*')) |>
  spq_select(lexemeId, lemma)
```

spq_group_by *Group the results by one or more variables*

Description

Group the results by one or more variables

Usage

```
spq_group_by(.query, ...)
```

Arguments

.query	query
...	Either R-DSL or strings with variable names

Value

A query object

Examples

```
spq_init() %>%
spq_add("?item wdt:P361 wd:Q297853") %>%
spq_add("?item wdt:P1082 ?folkm_ngd") %>%
spq_add("?area wdt:P31 wd:Q1907114") %>%
spq_label(area) %>%
spq_add("?area wdt:P527 ?item") %>%
spq_group_by(area, area_label)  %>%
spq_summarise(total_folkm = sum(folkm_ngd))
```

spq_head *Return the first lines of results*

Description

Return the first lines of results

Usage

```
spq_head(.query, n = 5)
```

Arguments

.query	a list with elements of the query
n	the maximum number of lines to return

Value

A query object

Subsetting

spq_offset() and spq_head() are only useful when used with spq_arrange() that makes the order of results predictable.

Examples

```
# Return the default of 5 items
spq_init() %>%
spq_add("?item wdt:P31 wd:Q5") %>%
spq_label(item) %>%
spq_add("?item wdt:P19/wdt:P131* wd:Q60") %>%
spq_add("?item wikibase:sitelinks ?linkcount") %>%
spq_arrange(desc(linkcount)) %>%
spq_head()

# Return 42 items
spq_init() %>%
spq_add("?item wdt:P31 wd:Q5") %>%
spq_label(item) %>%
spq_add("?item wdt:P19/wdt:P131* wd:Q60") %>%
spq_add("?item wikibase:sitelinks ?linkcount") %>%
spq_arrange(desc(linkcount)) %>%
spq_head(42)
```

---

spq_init *Initialize a query object.*

Description

Initialize a query object.

Usage

```
spq_init(
  endpoint = "wikidata",
  request_control = spq_control_request(user_agent = getOption("glitter.ua",
    "glitter R package (https://github.com/lvaudor/glitter)"), max_tries =
   getOption("glitter.max_tries", 3L), max_seconds = getOption("glitter.max_seconds",
    120L), timeout = getOption("glitter.timeout", 1000L), request_type = c("url",
    "body-form")),
  endpoint_info = spq_endpoint_info(label_property = "rdfs:label")
)
```

Arguments

endpoint	Endpoint, either name if it is in usual_endpoints, or an URL
request_control	An object as returned by spq_control_request()
endpoint_info	Do not use for an usual endpoint in usual_endpoints! Information about the endpoint

Value

A query object

Printing

SPARQL queries are shown using the cli package, with a built-in theme. You can change it by using the cli.user_theme option. We use

- .emph for keywords and functions,
- .field for variables,
- .pkg for prefixes,
- .val for strings,
- .url for prefix URLs.

You can also turn off the cli behavior by setting the environment variable "GLITTER.NOCLI" to any non-empty string. That's what we do in glitter snapshot tests.

---

spq_label *Label variables*

Description

Label variables

Usage

```
spq_label(
  .query,
  ...,
  .required = FALSE,
  .languages = getOption("glitter.lang", "en$"),
  .overwrite = FALSE
)
```

Arguments

.query	a list with elements of the query
...	variables by which to arrange (or SPARQL strings escaped with spq(), or strings, see examples)
.required	whether the existence of a value for the triple is required or not (defaults to TRUE). If set to FALSE, then other triples in the query are returned even if this particular triple is missing)
.languages	Languages for which to query labels. Use NULL for removing restrictions on language (defined or not), "*" for any defined language. If you write "en" you can get labels for regional variants such as "en-GB". If you want results for "en" only, write "en$".
.overwrite	whether to replace variables with their labels. spq_select(blop) means you get both blop and blop_label. spq_select(blop, .overwrite = TRUE) means you get the label as blop, the "original" blop variable isn't returned.

Details

spq_label() uses the property:

- associated with the usual endpoint see usual_endpoints
- the property indicated in spq_endpoint_info()

Value

A query object

Example

```
spq_init() %>%
  spq_add("?mayor wdt:P31 ?species") %>%
# dog, cat or chicken
  spq_set(species = c('wd:Q144','wd:Q146', 'wd:Q780')) %>%
# who occupy the function
  spq_add("?mayor p:P39 ?node") %>%
# of mayor
  spq_add("?node ps:P39 wd:Q30185") %>%
# of some places
  spq_add("?node pq:P642 ?place") %>%
  spq_label(mayor, place, .languages = c("fr", "en", "de")) %>%
  spq_perform()
```

spq_mutate *Create and modify variables in the results*

Description

Create and modify variables in the results

Usage

```
spq_mutate(
  .query,
  ...,
  .label = NA,
  .within_box = c(NA, NA),
  .within_distance = c(NA, NA)
)
```

Arguments

.query	a list with elements of the query
...	variables by which to arrange (or SPARQL strings escaped with spq(), or strings, see examples)
.label	**[Deprecated]** See spq_label().
.within_box	if provided, rectangular bounding box for the triple query. Provided as list(southwest=c(long=...,lat=...),no
.within_distance	if provided, circular bounding box for the triple query. Provided as list(center=c(long=...,lat=...), radius=...), with radius in kilometers. The center can also be provided as a variable (for instance, "?location") for the center coordinates to be retrieved directly from the query.

Value

A query object

Some examples

```
# common name of a plant species in different languages
# the triplet pattern "wd:Q331676 wdt:P1843 ?statement"
# creates the variable statement
# hence our writing it in reverse within the spq_mutate() function
spq_init() %>%
spq_mutate(statement = wdt::P1843(wd::Q331676)) %>%
spq_mutate(lang = lang(statement))
```

spq_offset *Offset the first generated result*

Description

Offset the first generated result

Usage

```
spq_offset(.query, n = 5)
```

Arguments

.query	a list with elements of the query
n	the maximum number of lines to return

Value

A query object

Subsetting

spq_offset() and spq_head() are only useful when used with spq_arrange() that makes the order of results predictable.

Examples

```
# Return 42 items
spq_init() %>%
spq_add("?item wdt:P31 wd:Q5") %>%
spq_label(item) %>%
spq_add("?item wdt:P19/wdt:P131* wd:Q60") %>%
spq_add("?item wikibase:sitelinks ?linkcount") %>%
spq_arrange(desc(linkcount)) %>%
spq_head(n=42)

# Return 42 items after the first 11 items
spq_init() %>%
spq_add("?item wdt:P31 wd:Q5") %>%
spq_label(item) %>%
spq_add("?item wdt:P19/wdt:P131* wd:Q60") %>%
spq_add("?item wikibase:sitelinks ?linkcount") %>%
spq_arrange(desc(linkcount)) %>%
spq_head(42) %>%
spq_offset(11)
```

`spq_perform` *Assemble query parts into a sparql query and send it to endpoint to get a tibble as a result.*

Description

Assemble query parts into a sparql query and send it to endpoint to get a tibble as a result.

Usage

```
spq_perform(
  .query,
  endpoint = lifecycle::deprecated(),
  user_agent = lifecycle::deprecated(),
  max_tries = lifecycle::deprecated(),
  max_seconds = lifecycle::deprecated(),
  timeout = lifecycle::deprecated(),
  request_type = lifecycle::deprecated(),
  dry_run = FALSE,
  replace_prefixes = FALSE
)
```

Arguments

`.query`	a list with elements of the query
`endpoint`	a string or url corresponding to a SPARQL endpoint. Defaults to "Wikidata"
`user_agent`	**[Deprecated]** a string indicating the user agent to send with the query.
`max_tries`, `max_seconds`	**[Deprecated]** Cap the maximal number of attemps with `max_tries` or the total elapsed time from the first request with `max_seconds`.
`timeout`	**[Deprecated]** maximum number of seconds to wait (`httr2::req_timeout()`).
`request_type`	**[Deprecated]** a string indicating how the query should be sent: in the URL (`url`, default, most common) or as a body form (`body-form`).
`dry_run`	Boolean indicating whether to return the output of `httr2::req_dry_run()` rather than of `httr2::req_perform()`. Useful for debugging.
`replace_prefixes`	Boolean indicating whether to replace used prefixes in the results table, for instance making, for instance "http://www.wikidata.org/entity/" "wd:".

Value

A query object

Request control

Control the way the query is performed via the `control_request` argument of `spq_init()`. This way you can create a basic spq object with all the correct options corresponding to the SPARQL service you are using, and then use it as the basis of all your subsequent glitter pipelines.

Examples

```
## Not run:
spq_init() %>%
  spq_add(.subject="?city",.verb="wdt:P31",.object="wd:Q515") %>%
  spq_add(.subject="?city",.verb="wdt:P1082",.object="?pop") %>%
  spq_label(city) %>%
  spq_head(n=5) %>%
  spq_perform()

## End(Not run)
```

---

spq_prefix *Add prefixes to the query*

Description

Add prefixes to the query

Usage

```
spq_prefix(.query = NULL, auto = TRUE, prefixes = NULL)
```

Arguments

`.query`	a list with elements of the query
`auto`	whether to use built-in prefixes
`prefixes`	a vector of prefixes

Value

A query object

Examples

```
spq_init() %>%
spq_prefix(prefixes=c(dbo="http://dbpedia.org/ontology/"))
```

spq_select *Select (and create) particular variables*

Description

Select (and create) particular variables

Usage

```
spq_select(.query = NULL, ..., .spq_duplicate = NULL)
```

Arguments

.query	a list with elements of the query
...	variables by which to arrange (or SPARQL strings escaped with spq(), or strings, see examples)
.spq_duplicate	How to handle duplicates: keep them (NULL), eliminate (distinct) or reduce them (reduced, advanced usage).

Value

A query object

Examples

```
spq_init() |>
  spq_prefix(prefixes = c(dct = "http://purl.org/dc/terms/")) |>
  spq_add(spq('?lexemeId dct:language wd:Q1860')) |>
  spq_add(spq("?lexemeId wikibase:lemma ?lemma")) |>
  spq_filter(str_detect(lemma, '^pota.*')) |>
  spq_select(- lemma)

spq_init() |>
  spq_prefix(prefixes = c(dct = "http://purl.org/dc/terms/")) |>
  spq_add(spq('?lexemeId dct:language wd:Q1860')) |>
  spq_add(spq("?lexemeId wikibase:lemma ?lemma")) |>
  spq_filter(str_detect(lemma, '^pota.*')) |>
  spq_select(lemma)
```

spq_set *Set helper values for the query*

Description

Set helper values for the query (helps with readability)

Usage

```
spq_set(.query, ...)
```

Arguments

.query	query
...	Helper values and their definition.

Value

A query object

Some examples

```
# find the individuals of the species
spq_init() %>%
# dog, cat or chicken
spq_set(species = c('wd:Q144','wd:Q146', 'wd:Q780'), mayorcode = "wd:Q30185") %>%
spq_filter(mayor == wdt::P31(species)) %>%
spq_add("?mayor p:P39 ?node") %>%
# of mayor
spq_add("?node ps:P39 ?mayorcode") %>%
# of some places
spq_add("?node pq:P642 ?place") %>%
spq_label(species, mayor, place) %>%
spq_select(-species, -place, -node, -mayor, -mayorcode) %>%
spq_perform()
```

spq_summarise *Summarise each group of results to fewer results*

Description

Summarise each group of results to fewer results

Usage

```
spq_summarise(.query, ...)

spq_summarize(.query, ...)
```

Arguments

.query	a list with elements of the query
...	variables by which to arrange (or SPARQL strings escaped with spq(), or strings, see examples)

Value

A query object

Examples

```
result = spq_init() %>%
spq_add("?item wdt:P361 wd:Q297853") %>%
spq_add("?item wdt:P1082 ?folkm_ngd") %>%
spq_add("?area wdt:P31 wd:Q1907114") %>%
spq_label(area) %>%
spq_add("?area wdt:P527 ?item") %>%
spq_group_by(area, area_label)  %>%
spq_summarise(total_folkm = sum(folkm_ngd))
```

---

spq_tally *Count the observations*

Description

These functions are inspired by dplyr::count() and dplyr::tally(). spq_tally() assumes you've already done the grouping.

Usage

```
spq_tally(.query, sort = FALSE, name = "n")

spq_count(.query, ..., sort = FALSE, name = "n")
```

Arguments

.query	a list with elements of the query
sort	If TRUE, will show the largest groups at the top. (like the sort argument of dplyr::tally())
name	Name for the count column (like the name argument of dplyr::tally())
...	variables by which to arrange (or SPARQL strings escaped with spq(), or strings, see examples)

Value

A query object

Examples

```
## Not run:
spq_init() %>%
spq_add("?film wdt:P31 wd:Q11424") %>%
spq_mutate(narrative_location = wdt::P840(film)) %>%
spq_label(narrative_location) %>%
spq_tally(name = "n_films") %>%
spq_perform()

# the same with spq_count

spq_init() %>%
spq_add("?film wdt:P31 wd:Q11424") %>%
spq_mutate(narrative_location = wdt::P840(film)) %>%
spq_label(narrative_location) %>%
spq_count(name = "n_films") %>%
spq_perform()

# Now with grouping
spq_init() %>%
spq_add("?film wdt:P31 wd:Q11424") %>%
spq_mutate(narrative_location = wdt::P840(film)) %>%
spq_label(film, narrative_location) %>%
spq_group_by(narrative_location_label) %>%
spq_tally(sort = TRUE, name = "n_films") %>%
spq_perform()


# More direct with spq_count()
spq_init() %>%
spq_add("?film wdt:P31 wd:Q11424") %>%
spq_mutate(narrative_location = wdt::P840(film)) %>%
spq_label(film, narrative_location) %>%
spq_count(narrative_location_label, sort = TRUE, name = "n_films") %>%
spq_perform()

## End(Not run)
```

`usual_endpoints` *Usual endpoints: this dataset allows the user to refer to them using a simplified name rather than their full url.*

Description

Usual endpoints: this dataset allows the user to refer to them using a simplified name rather than their full url.

Usage

```
usual_endpoints
```

Format

A data frame with usual SPARQL endpoints and abbreviated names

name the abbreviated name of the SPARQL endpoint

url the full address of the SPARQL endpoint

label_property the property used for labelling

---

usual_prefixes *Usual prefixes: this dataset allows the user to refer to usual prefixes in their queries without manually specifying the associated urls.*

Description

Usual prefixes: this dataset allows the user to refer to usual prefixes in their queries without manually specifying the associated urls.

Usage

```
usual_prefixes
```

Format

A data frame with usual prefixes

type the type of prefix

name the prefix itself

url the corresponding ontology ...

---

wd_properties *Wikidata properties*

Description

Wikidata properties

Usage

```
wd_properties
```

Format

A data frame with 8939 rows and 5 variables:

id id

type property type

label property label

description property description

altLabel alternative labels ...

Source

Wikidata https://query.wikidata.org/#SELECT%20%3Fproperty%20%3FpropertyType%20%3FpropertyLabel%20%3FpropertyDescription%20%3FpropertyAltLabel%0AWHERE%20%7B%0A%20%20%3Fproperty%20wikibase%3ApropertyType%20%3FpropertyType.%0A%20%20SERVICE%20wikibase%3Alabel%20%7Bbd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%7D%0AORDER%20BY%20ASC%20%28xsd%3Ainteger%28STRAFTER%28STR%28%3Fproperty%29%2C%20%27P%27%29%29%29

Index