Package ‘rPref’

March 20, 2015

Version 0.5
Date 2015-03-20
Title Database Preferences and Skyline Computation
Author Patrick Roocks <mail@p-roocks.de>
Maintainer Patrick Roocks <mail@p-roocks.de>
Description Routines to select and visualize the maxima for a given strict partial order. This especially includes the computation of the Pareto frontier, also known as (Top-k) Skyline operator, and some generalizations (database preferences).

URL http://www.p-roocks.de/rpref
Depends R (>= 3.0.0), methods
Imports Rcpp (>= 0.11.5), RcppParallel (>= 4.3.6), dplyr (>= 0.4.1), igraph (>= 0.7.1)
SystemRequirements GNU make, Windows: cmd.exe and cscript.exe
License GPL (>= 2)
LazyData true
LinkingTo Rcpp, RcppParallel
Suggests testthat
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-03-20 14:28:57

R topics documented:

base_pref ................................................. 2
base_pref_macros ....................................... 3
complex_pref ........................................... 5
get_big ...................................................... 7
get_hasse_diag .......................................... 8
plot_front ............................................... 9
Description

Base preferences are used to describe the different goals of a preference query.

Usage

- low(expr)
- low_(expr)
- high(expr)
- high_(expr)
- true(expr)
- true_(expr)

Arguments

expr A numerical/logical expression which is the term to evaluate for the current preference. The objective is to search for minimal/maximal values of this expression (for low/high) or for logical TRUE values (for true).

Details

Mathematically, all base preferences are strict weak orders (irreflexive, transitive and negative transitive).

The three fundamental base preferences are:

- low(a), high(a) Search for minimal/maximal values of a, i.e., the induced order is the "smaller than" or "greater than" order on the values of a. The values of a must be numeric values.
- true(a) Search for true values in logical expressions, i.e., TRUE is considered to be better than FALSE. The values of a must be logical values. For a tuplewise evaluation of a complex logical expression one has to use the & and | operators for logical AND/OR (and not the && and || operators).
The term expr may be just a single attribute or may contain an arbitrary expression, e.g., low(a + 2*b + f(c)). There a, b and c are columns of the addressed dataset and f is a previously defined function.

Functions contained in expr are evaluated over the entire dataset, i.e., it is possible to use aggregate functions (min, mean, etc.). Note that all functions (and also variables which are not columns of the dataset, where expr will be evaluated on) must be defined in the same environment (e.g. environment of a function or global scope) as the base preference.

Using expressions in preferences

The low(e), high(e) and true(e) preferences have the same functionality, but expect an expression e or symbol e as argument. For example, low(a) is equivalent to low(expression(a)) or low(as.symbol("a")).

This is very helpful for developing your own base preferences. Assume you want to define a base Preference false as the dual of true. A definition like false <- function(x) -true(x) is the wrong approach, as psel(data.frame(a = c(1,2)), false(a == 1)) will result in the error object 'a' not found. This is because a is considered as a variable and not as an (abstract) symbol to be evaluated later on the data set. By defining
false <- function(x) -true(substitute(x))

one gets a preference which behaves like a "built-in" preference. The object false(a == 1) will output [Preference] -true(a == 1) on the console and psel(data.frame(a = c(1,2)), false(a==1)) returns correctly the second tuple with a==2.

See Also

See complex_pref how to compose complex preferences to retrieve e.g. the Skyline.

See base_pref_macros for more base preferences.

Examples

```r
# Define a preference with a score value combining mpg and hp
p1 <- high(4 * mpg + hp)
# Perform the preference selection
psel(mtcars, p1)

# Define a preference with a given function
f <- function(x, y) (abs(x - mean(x))/max(x) + abs(y - mean(y))/max(y))
p2 <- low(f(mpg, hp))
psel(mtcars, p2)
```

Description

In addition to the fundamental base preferences, rPref offers some macros to define preferences where a given interval or point is preferred.

Useful base preference macros

```r
base_pref_macros
```

base_pref_macros  Useful base preference macros
Usage

around(expr, center)
between(expr, left, right)

pos(expr, pos_value)

layered(expr, ...)

Arguments

expr A numerical expression (for around and between) or an arbitrary expression (for pos and layered). The objective are tuples where expr evaluates to a value in the preferred interval, layer, etc. Regarding attributes, functions and variables, the same requirements as for base_pref apply.

center Preferred value for around.

left Lower limit of the preferred interval for between.

right Upper limit of the preferred interval for between.

pos_value A vector containing the preferred values for a pos preference. Has to be of the same type (numeric, logical, character, ...) as expr.

... Layers (sets) for a layered preference. Each variable corresponds to one layer and the first set characterizes the most preferred values.

Definition of the preference macros

between(expr, left, right) Those tuples are preferred where expr evaluates to a value between left and right. For values not in this interval the values nearest to the interval are preferred.

around(expr, center) Same as between(expr, center, center).

pos(expr, pos_value) Those tuples are preferred, where expr evaluates to a value which is contained in pos_value.

layered(expr, layer1, layer2, ..., layerN) For the most preferred tuples expr must evaluate to a value in layer1. The second-best tuples are those where expr evaluates to a value in layer2 and so forth. Values occuring in none of the layers are considered worse than those in layerN. Technically, this is realized by a Prioritization (lexicographical order) chain of true preferences.

Examples

# Search for cars where mpg is near to 25
psel(mtcars, around(mpg, 25))

# cyl = 2 and cyl = 4 are equally good, cyl = 6 is worse
psel(mtcars, layered(cyl, c(2, 4), 6))
Description

Complex preferences are used to compose different preferences orders. For example the Pareto composition (via operator *) is the usual operator to compose the preference for a Skyline query. The Skyline is also known as Pareto frontier. All complex preferences are mathematically strict partial orders (irreflexive and transitive).

Usage

```plaintext
## S3 method for class 'preference'
p1 * p2

## S3 method for class 'preference'
p1 & p2

## S3 method for class 'preference'
p1 | p2

## S3 method for class 'preference'
p1 + p2

reverse(p)

empty()

## S3 method for class 'preference'
length(x)
```

Arguments

```plaintext
p, p1, p2, x
```
Preferences (they can be either base preferences, see `base_pref`, or complex preferences)

Skylines

The most important preference composition operator is the Pareto operator (p1 * p2) to formulate a Skyline query. A tuple t1 is better than t2 w.r.t. p1 * p2 if it is strictly better w.r.t. one of the preferences p1, p2 and is better or equal w.r.t. the other preference.

The syntactical translation from other query languages supporting Skylines/Preferences to rPref is as follows:

- A query in the syntax from Borzsonyi et. al (2001) like
  ```plaintext
  "... SKYLINE OF a MAX, b MIN, c MAX"
  ```
  corresponds in rPref to the preference
complex_pref

high(a) * low(b) * high(c).

- A query in the syntax from Kiessling (2002) like 
  "... PREFERING a LOWEST AND (b HIGHEST PRIOR TO c LOWEST)"
  corresponds in rPref to
  low(a) * (high(b) & low(c)).

- A query in the syntax of the "Skyline" feature of the commercial database "EXASOL EXASolution 5" like 
  "... PREFERING LOW a PLUS (b = 1 PRIOR TO LOW c))"
  corresponds in rPref to
  low(a) * (true(b == 1) & low(c)).

Note that these query conversions can be done by show.query.

Definition of additional preference operators

Additionally, rPref supports the following preference composition operators:

p1 & p2 Prioritization (lexicographical order): A tuple t1 is better than t2 w.r.t. p1 & p2 if it is strictly better w.r.t. p1 or is equal w.r.t. p1 and is better w.r.t. p2.

p1 | p2 Intersection preference: A tuple t1 is better than t2 w.r.t. p1 | p2 if it is strictly better w.r.t. both preferences. This is a stricter variant of the Pareto operator. The evaluation of psel(df, p1 | p2) is always a subset of psel(df, p1 * p2).

p1 + p2 Union preference: A tuple t1 is better than t2 w.r.t. p1 + p2 if it is strictly better w.r.t. to one of the preferences. Note that this can violate the strict partial order property, if the domains (the tuples on which p1 and p2 define better-than-relationships) of the preferences are not disjoint.

reverse(p1) or ~p1 Reverse preference (converse relation): A tuple t1 is better than t2 w.r.t. ~p1 if t2 is better than t1 w.r.t. p1. The unary minus operator, i.e. ~p1, is a short notation for reverse(p1).

empty() Empty preference, i.e. a neutral element for all complex preference compositions. It holds that op(empty(), p) is equal to p for all preference operators op and all preferences p.

Preference term length

The function length(p) gives the term length of the preference term p which is defined as the number of base preferences in a complex preference term.

References


get_btg

See Also

See base_pref for the construction of base preferences. See psel for the evaluation of preferences.

Examples

# Define preference for cars with low consumption (high mpg-value)
# and simultaneously high horsepower
p1 <- high(mpg) * high(hp)

# Perform the preference search
psel(mtcars, p1)

get_btg            Better-Than-Graph

Description

Returns a Hasse-Diagramm of a preference order (also called the Better-Than-Graph) on a given dataset to be plotted with the igraph package.

Usage

get_btg(df, pref)

Arguments

df          A dataframe.
pref        A preference on the columns of df, see psel for details.

Details

This function returns a list l with the following list entries:

l$graph    An igraph object, created with the igraph package.
l$layout    A typical Hasse-diagram layout for plotting the graph, also created with igraph.

To plot the resulting graph use the plot function as follows: plot(l$graph, layout = l$layout).
For more details see igraph.plotting and the examples below.

The Hasse diagram of a preference visualizes all the better-than-relationships on a given dataset. All edges which can be retrieved by transitivity of the order are omitted.

The names of the vertices are characters ranging from "1" to as.character(nrow(df)) and they correspond to the row numbers of df.

See Also

igraph.plotting
Examples

```r
# Pick a small data set and create preference and BTG
df <- mtcars[1:10,]
pref <- high(mpg) * low(wt)
btg <- get_btg(df, pref)

# Create labels for the nodes with relevant values
labels <- paste0(df$mpg, " \n", df$wt)

# Plot the graph using igraph
library(igraph)
plot(btg$graph, layout = btg$layout, vertex.label = labels,
     vertex.size = 25)

# Add colors for the maxima nodes and plot again
colors <- rep(rgb(1, 1, 1), nrow(df))
colors[psel.indices(df, pref)] <- rgb(0,1,0)
plot(btg$graph, layout = btg$layout, vertex.label = labels,
     vertex.size = 25, vertex.color = colors)

# Show lattice structure of 3-dimensional Pareto preference
df <- merge(merge(data.frame(x = 1:3), data.frame(y = 1:3)), data.frame(z = 1:2))
labels <- paste0(df$x, ",", df$y, ",", df$z)
btg <- get_btg(df, low(x) * low(y) * low(z))
plot(btg$graph, layout = btg$layout, vertex.label = labels,
     vertex.size = 20)
```

---

get_hasse_diag

Adjacency list of Hasse diagramm

Description

Returns the adjacency list as a (n x 2) matrix. This is the transitive reduction of the preference relation.

Usage

```r
get_hasse_diag(df, pref)
```

Arguments

- `df`: A dataframe.
- `pref`: A preference on the columns of `df`, see `psel` for details.

Details

A row (i, j) in the resulting matrix means that `df[i,]` is better than `df[j,]` with regard to the preference `p`. The matrix is the transitive reduction (Hasse diagram) of the induced relations, i.e., if (1,2) and (2,3) occur in the result, then (1,3) will not be contained. The number of rows in the result depends on the number of non-transitive Better-Than-Relationships in `df` w.r.t. `p`. 
See Also

`get_btg` to plot the Hasse Diagram.

Examples

```r
get_hasse_diag(mtcars, low(mpg))
```

### Description

Connects the points of a Pareto front (also known as Pareto frontier) and hence visualizes the dominance region of a Skyline.

### Usage

```r
plot_front(df, pref, ...)
```

### Arguments

- **df**: The dataframe for which the Pareto front is plotted. This may be already a maxima set w.r.t. the preference `pref`, but anyway the maxima set is recalculated via `psel(df, pref)`.
- **pref**: The preference representing the Skyline goals. This must be a pareto composition (`p1 * p2`) or intersection composition (`p1 | p2`) of two `low` or `high` preferences.
- **...**: Additional graphic parameters which are passed to the `segments` function (internally used to plot the front).

### Details

`plot_front` assumes that there is an existing plot, where the value of the first preference was plotted as x-coordinate and the value of the second preference as y-coordinate.

### Examples

```r
# Plots Pareto fronts for the hp/mpg values of mtcars
show_front <- function(pref) {
  plot(mtcars$hp, mtcars$mpg)
  sky <- psel(mtcars, pref)
  plot_front(mtcars, pref, col = rgb(0, 0, 1))
  points(sky$hp, sky$mpg, lwd = 3)
}

# Do this for all four combinations of pareto compositions
show_front(low(hp) * low(mpg))
show_front(low(hp) * high(mpg))
```
show_front(high(hp) * low(mpg))
show_front(high(hp) * high(mpg))

# Compare this to the front of a intersection preference
show_front(high(hp) | high(mpg))

---

**pred_succ**

**Predecessor and successor functions**

**Description**

Function for traversing the BTG (Better-Than-Graph) of a preference.

**Usage**

```r
init_pred_succ(df, p)
hasse_pred(p, v, intersect = FALSE)
hasse_succ(p, v, intersect = FALSE)
all_pred(p, v, intersect = FALSE)
all_succ(p, v, intersect = FALSE)
```

**Arguments**

- `df` A dataframe characterizing the set wherein predecessors/successors are searched.
- `p` A preference. Worse tuples in the induced order are successors and better tuples are predecessors.
- `v` A numeric vector of indices in `df`. The represents the set of tuples for which predecessors/successors are searched.
- `intersect` Logical value. If `FALSE` (by default) the union of all predecessors/successors of `v` are returned. For `intersect = TRUE` the intersection of those is returned.

**Details**

These functions return the predecessors and successors in the Better-Than-Graph of a preference which can be plotted via `get_btg`. Before any of the successor/predecessor functions can be used the initialization has to be called as follows:

```r
init_pred_succ(p, df)
```

There `p` is a preference object and `df` a dataframe. This statement calculates the Better-Than-Relation on `df` w.r.t. `p`. Afterwards the subsequent predecessor and successor functions can be called. The value of `v` is a numeric vectors within `1:nrow(df)` and characterizes a subset of tuples in `df`. The return value of these functions is again a numeric vector referring to the row numbers in `df` and it is always ordered ascending, independent of the order of the indices in `v`. 
all_pred(p, v) Returns all predecessors of v, i.e. indices of better tuples than v.
all_succ(p, v) Returns all predecessors of v, i.e. indices of worse tuples than v.
hasse_pred(p, v) Returns the direct predecessors of v, i.e. indices of better tuples than v where the Better-Than-Relation is contained in the transitive reduction.
hasse_succ(p, v) Returns the direct successors of v, i.e. indices of worse tuples than v where the Better-Than-Relation is contained in the transitive reduction.

If v has length 1, then the value of intersect does not matter, as there is nothing to intersect or join. For scalar values x and y the following identities hold, where f is one of the predecessor/successor functions:
\[ f(p, c(x, y), \text{intersect} = \text{FALSE}) = \text{union}(f(p, x), f(p, y)) \]
\[ f(p, c(x, y), \text{intersect} = \text{TRUE}) = \text{intersect}(f(p, x), f(p, y)) \]

Examples

# Preference on mtcars for high mpg and low weight
p <- high(mpg) * low(wt)
init_pred_succ(mtcars, p)

# Helper to show mpg/hp values
show_vals <- function(x) mtcars[x, c('mpg', 'wt')]

# Pick some tuple "in the middle"
show_vals(10)

# Show (direct) predecessors/successors of tuple 10
show_vals(hasse_pred(p, 10)) # Next better car
show_vals(hasse_succ(p, 10)) # Next worse car
show_vals(all_pred(p, 10)) # All better cars
show_vals(all_succ(p, 10)) # All worse cars

psel Preference selection

Description

Evaluates a preference on a given dataset, i.e., return the maximal elements of a dataset for a given preference order.

Usage

psel(df, pref, ...)

psel.indices(df, pref, ...)
Arguments

- **df**: A dataframe or, for a grouped preference selection, a grouped dataframe. See below for details.
- **pref**: The preference order constructed via `complex_pref` and `base_pref`. All variables occurring in the definition of `pref` must be either columns of the dataframe `df` or variables/functions of the environment where `pref` was defined.

... Additional parameters for Top(-Level)-k selections:

- **top**: A top value of k means that the k-best tuples of the dataset are returned. This may be non-deterministic, see below for details.
- **at_least**: A at_least value of k returns the Top-k tuples and additionally all tuples which are not dominated by the worst tuple (i.e. the minima) of the Top-k set. The number of tuples returned is greater or equal than at_least. In contrast to top-k, this is deterministic.
- **top_level**: A top_level value of k returns all tuples from the k-best levels. See below for the definition of a level.
- **and_connected**: Logical value, which is only relevant if more than one of the above {top, at_least, top_level} values are given. Then and_connected = TRUE (which is the default) means that all top-conditions must hold for the returned tuples: Let cond1 and cond2 be top-conditions like top=2 or top_level=3, then `psel([[...], cond1, cond2])` is equivalent to the intersection of `psel([[...], cond1])` and `psel([[...], cond2])`. If we have and_connected = FALSE these conditions are or-connected. This corresponds to the union of `psel([[...], cond1])` and `psel([[...], cond2])`.
- **show_level**: Logical value. If TRUE, a column `.level` is added to the returned dataframe, containing all level values (see below for details). This is only relevant if at least one of the {top, at_least, top_level} values are given. For `psel` this is TRUE by default, for `psel.indices` this is FALSE by default.

Details

The difference between the two variants of the preference selection is:

- The `psel` function returns a subset of the dataset which are the maxima according to the given preference.
- The function `psel.indices` returns just the row indices of the maxima (except Top-k queries with `show_level = TRUE`, see Top-k preference selection). Hence `psel(df, pref)` is equivalent to `df[psel.indices(df, pref),]` for non-grouped dataframes.

Top-k preference selection

For a given top value "k" the k best elements and their level values are returned. The level values are determined as follows:

- All the maxima of a dataset w.r.t. a preference have level 1.
- The maxima of the remainder, i.e. the dataset without the level 1 maxima, have level 2.
- The n-th iteration of "Take the maxima from the remainder" leads to tuples of level n.
By default, `psel.indices` does not return the level values. By setting `show_level = TRUE` this function returns a dataframe with the columns `.indices` and `.level`.

By definition, a top-k preference selection is non-deterministic. A top-1 query of two equivalent tuples (equivalence according to `pref`) can return on both of these tuples. E.g., for tuples `(a=1,b=1),(a=1,b=2)` a top=1 selection with a `pref=low(a)` preference can return either the `b=1` or the `b=2` tuple.

On the contrary a preference selection with `at_least` is deterministic by adding all tuples having the same level as the worst level of the corresponding top-k query, i.e., which are incomparable to the minimum of the top-k result. A preference selection with `top_level-k` returns all tuples having level k or better.

If the top or `at_least` value is greater than the number of elements in `df` (i.e. `nrow(df)`), or `top_level` is greater than the highest level in `df`, then all elements of `df` will be returned without further warning.

**Grouped preference selection**

With `psel` it is also possible to perform a preference selection where the maxima are calculated for every group separately. The groups have to be created with `group_by` from the dplyr package. The preference selection preserves the grouping, i.e., the groups are restored after the preference selection.

For example, if the summarize function from dplyr is applied to `psel(group_by(...), pref)`, the summarizing is done for the set of maxima of each group. This can be used to e.g. calculate the number of maxima in each group, see examples below.

A `{top, at_least, top_level}` preference selection is applied to each group separately. A `top=k` selection returns the k best tuples for each group. Hence if there are 3 groups in `df`, each containing at least 2 elements, and we have `top = 2`, then 6 tuples will be returned.

**Parallel computation**

On multicore machines the preference selection runs in parallel using a divide-and-conquer approach. If you prefer a single-threaded computation, use the following code to deactivate parallel computation within rPref:

```r
options(rpref.parallel = FALSE)
```

If this option is not set, rPref will use parallel computation by default.

**See Also**

See `complex_pref` on how to construct a Skyline preference. See `plot_front` on how to plot the pareto front of a Skyline.

**Examples**

```r
# Skyline and Top-K/At-least skyline
psep(mtcars, low(mpg) * low(hp))
psep(mtcars, low(mpg) * low(hp), top = 5)
psep(mtcars, low(mpg) * low(hp), at_least = 5)

# Visualize the skyline in a plot
```
library(dplyr) psel(group_by(mtcars, cyl), low(mpg))

# Return size of each maxima group
summarise(psel(group_by(mtcars, cyl), low(mpg)), n())

---

Description

rPref contains routines to select and visualize the maxima for a given strict partial order. This especially includes the computation of the Pareto frontier, also known as (Top-k) Skyline operator, and some generalizations (database preferences).

Preference composition/selection

- Preferences are composed from base preference (see `base_pref`) and complex preferences (see `complex_pref`), where especially the Pareto operator for Skylines is such a complex preferences. Additionally some base preference macros can be found in `base_pref_macros`.
- The (Top-Level)-k preference selection `psel` allows to retrieve the maxima of a preference (or Pareto frontier, Skyline), constructed with the functions above, on a given dataset.

Visualization and analysis of preferences

- The visualization of the preference order in a Better-Than-Graph is possible via the `get_btg` function in connection with the `igraph` package.
- The adjacency list of the Better-Than-Graph (Hasse diagramm) can be accessed via `get_hasse_diag`.
- Predecessors/successors in the Hasse diagram are calculated with the `pred_succe` functions.
- The pareto frontier can be plotted using the `plot_front` function.

String output of preferences

- The preference query for some preference-supporting DBMS can be given by `show.query`.
- A preference is partially evaluated and printed with `show.pref`.

Further information

The rPref homepage is [http://www.p-roocks.de/rpref](http://www.p-roocks.de/rpref). To submit bugs, feature requests or other comments, feel free to write a mail to me.

Author(s)

Patrick Roocks, <mail@p-roocks.de>
Partial evaluation and string output of preferences

Description

Functions to substitute variables and functions in preferences which can be calculated before the preference is evaluated on a dataframe. This is especially used for string output of preferences.

Usage

show.pref(p, df = NULL)
pref.str(p, df = NULL)
eval.pref(p, df = NULL)

## S3 method for class 'preference'
as.character(x, ...)

Arguments

p,x The preference to be shown or partially evaluated.
df (Optional) A dataframe on which the preference operates.
... Optional arguments passed to as.character.

Details

The function pref.str (or as.character(p) for a preference p) returns the preference string while show.pref outputs it directly to the console, preceded by 'Preference'. If df is specified, then a partial evaluation of the preference is done before converting it to a string.

The function eval.pref (with given dataframe df) partially evaluates the internal preference expression and returns again a preference object. All expressions in p are evaluated in the environment where p was defined, except the the column names in df (which are potential attributes in p). The content of the dataframe df does not matter; only names(df) is used to get the "free variables" in p.

Partial evaluation before string output

The functions show.pref and pref.str have the optional parameter df. If this parameter is given, these functions call eval.pref before they output/return the preference string. The following equalities hold:

- \( \text{as.character(eval.pref(p, df))} = \text{pref.str(p, df)} \)
- \( \text{show(eval.pref(p, df))} \) produces the same console output as show.pref(p, df)
**Examples**

```r
f <- function(x) 2*x
p <- true(cyl == f(1))

# prints 'cyl == f(x)'
p

# prints 'cyl == 2'
show.pref(p, mtcars)
eval.pref(p, mtcars)
```

**Description**

For a given preference this shows the PREFERING clause of a database query in different SQL dialects which support preferences.

**Usage**

```r
show.query(p, dialect = "EXASOL", df = NULL)
```

**Arguments**

- `p` A preference.
- `dialect` The preference query dialect, which determines the Syntax of the returned query. This has to be one of the following:
  - 'EXASOL': Syntax of the "Skyline" feature of the commercial database EXASOL EXASolution 5.
  - 'Preference SQL' or 'PSQL': Syntax of the Preference SQL system. This is a research prototype developed at the Chair of Databases and Information Systems of the University of Augsburg. See references for details.
- `df` Optional parameter to specify a dataframe on which the preference operates causing a partial evaluation. See `show.pref` for details.

**Details**

There are few database systems supporting Skyline queries. A Skyline query consists of a usual SQL query followed by a PREFERING-clause (in some rarely used dialects also SKYLINE OF). For example consider a database table `r(a,b)`. The preference selection `psel(r, low(a) * high(b))` can be expressed by (in the Exasol dialect):

```sql
SELECT * FROM r PREFERING LOW a PLUS HIGH b
```

The `show.query` function generates just the PREFERING-clause, i.e. `show.query(low(a) * high(b))` returns
PREFERRING LOW a PLUS HIGH b

As usual in SQL queries, all keywords are not case sensitive, i.e., PLUS or plus does not make any difference.

References


Examples

show.query(low(a) * high(b))

show.query(low(a) * high(b), dialect = 'Preference SQL')
Index

*Topic skyline

complex_pref, 5
*.preference (complex_pref), 5
+.preference (complex_pref), 5
-.preference (complex_pref), 5
&.preference (complex_pref), 5

all_pred (pred_succ), 10
all_succ (pred_succ), 10
around (base_pref_macros), 3
as.character.preference (show.pref), 15

base_pref, 2, 4, 5, 7, 12, 14
base_pref_macros, 3, 3, 14
between (base_pref_macros), 3

complex_pref, 3, 5, 12–14

easy (complex_pref), 5
eval.pref (show.pref), 15

get_btg, 7, 9, 10, 14
get_hasse.diag, 8, 14
group_by, 13

hasse_pred (pred_succ), 10
hasse_succ (pred_succ), 10
high, 9
high (base_pref), 2
high_ (base_pref), 2

igraph, 7, 14
igraph.plotting, 7
init_pred_succ (pred_succ), 10

layered (base_pref_macros), 3
length.preference (complex_pref), 5
low, 9
low (base_pref), 2
low_ (base_pref), 2

plot_front, 9, 13, 14

pos (base_pref_macros), 3
pred_succ, 10, 14
pref.str (show.pref), 15
psel, 7, 8, 11, 14

reverse (complex_pref), 5
rPref, 14
rPref-package (rPref), 14

segments, 9
show.pref, 14, 15, 16
show.query, 6, 14, 16

ture, 4
true (base_pref), 2
ture_ (base_pref), 2