Package 'RGMQL'

March 30, 2021

Type Package

Title GenoMetric Query Language for R/Bioconductor

Version 1.10.0

Author Simone Pallotta, Marco Masseroli

Maintainer Simone Pallotta <simonepallotta@hotmail.com>

Description This package brings the GenoMetric Query Language (GMQL) functionalities into the R environment. GMQL is a high-level, declarative language to manage heterogeneous genomic datasets for biomedical purposes, using simple queries to process genomic regions and their metadata and properties. GMQL adopts algorithms efficiently designed for big data using cloud-computing technologies (like Apache Hadoop and Spark) allowing GMQL to run on modern infrastructures, in order to achieve scalability and high performance. It allows to create, manipulate and extract genomic data from different data sources both locally and remotely. Our RGMQL functions allow complex queries and processing leveraging on the R idiomatic paradigm. The RGMQL package also provides a rich set of ancillary classes that allow sophisticated input/output management and sorting, such as: ASC, DESC, BAG, MIN, MAX, SUM, AVG, MEDIAN, STD, Q1, Q2, Q3 (and many others). Note that many RGMQL functions are not directly executed in R environment, but are deferred until real execution is issued.

License Artistic-2.0

URL http://www.bioinformatics.deib.polimi.it/genomic_computing/GMQL/

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports httr, rJava, GenomicRanges, rtracklayer, data.table, utils, plyr, xml2, methods, S4Vectors, dplyr, stats, glue, BiocGenerics

Depends R(>= 3.4.2), RGMQLlib

VignetteBuilder knitr

Suggests BiocStyle, knitr, rmarkdown

biocViews Software, Infrastructure, DataImport, Network, ImmunoOncology, SingleCell

Collate 'AllClasses.R' 'AllGenerics.R' 'GMQLtoGRanges.R'
'GRangesToGMQL.R' 'S3Aggregates.R' 'S3Cover-Param.R'
'S3Distal.R' 'S3Operator.R' 'Utils.R' 'evaluation-functions.R'
'filter-extract-function.R' 'gmql_cover.R' 'gmql_difference.R'
'gmql_extend.R' 'gmql_group.R' 'gmql_init.R' 'gmql_join.R'
'gmql_map.R' 'gmql_materialize.R' 'gmql_merge.R' 'gmql_order.R'
'gmql_project.R' 'gmql_read.R' 'gmql_select.R' 'gmql_union.R'
'onLoad.R' 'ordering-functions.R' 'web-services.R'

git_url https://git.bioconductor.org/packages/RGMQL

git_branch RELEASE_3_12

git_last_commit 61091e1

git_last_commit_date 2020-10-27

Date/Publication 2021-03-29

R topics documented:

aggregate . . . 3
AGGREGATES-Object . . . 4
arrange . . . 6
collect . . . 8
compile_query . . . 9
cover . . . 10
Cover-Param . . . 12
delete_dataset . . . 13
DISTAL-Object . . . 14
download_dataset . . . 16
Evaluation-Function . . . 17
execute . . . 17
export_gmql . . . 18
extend . . . 19
filter . . . 20
filter_and_extract . . . 22
group_by . . . 23
import_gmql . . . 25
init_gmql . . . 26
login_gmql . . . 27
logout_gmql . . . 28
log_job . . . 29
map . . . 30
merge . . . 31
OPERATOR-Object . . . 33
Ordering-Functions . . . 34
read_gmql . . . 35
register_gmql . . . 37
remote_processing . . . 38
run_query . . . 38
sample_metadata . . . 40
sample_region . . . 40
save_query . . . 41
select . . . 42

semijoin 44
setdiff 45
show_datasets_list 46
show_jobs_list 47
show_queries_list 48
show_samples_list 48
show_schema 49
stop_gmql 50
stop_job 51
take 52
union 53
upload_dataset 54

Index **56**

`aggregate` *Method aggregate*

Description

Wrapper to GMQL MERGE operator

It builds a dataset consisting of a single sample having as many regions as the number of regions of all the input dataset samples and as many metadata as the union of the 'attribute-value' tuples of the input samples. If *groupBy* is specified, the samples are then partitioned in groups, each with a distinct value of the grouping metadata attributes. The operation is separately applied to each group, yielding one sample in the result for each group. Samples whose metadata are not present in the grouping metadata parameter are disregarded.

Usage

```
## S4 method for signature 'GMQLDataset'
aggregate(x, groupBy = conds())
```

Arguments

`x`	GMQLDataset class object
`groupBy`	`condition_evaluation` function to support methods with groupBy or JoinBy input paramter

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named "exp"
## using CustomParser

init_gmql()
```

```
test_path <- system.file("example", "DATASET", package = "RGMQL")
exp = read_gmql(test_path)

## This statement creates a dataset called merged which contains one
## sample for each antibody_target and cell value found within the metadata
## of the exp dataset sample; each created sample contains all regions
## from all 'exp' samples with a specific value for their
## antibody_target and cell metadata attributes.

merged = aggregate(exp, conds(c("antibody_target", "cell")))
```

AGGREGATES-Object *AGGREGATES object class constructor*

Description

This class constructor is used to create instances of AGGREGATES object, to be used in GMQL functions that require aggregate on value.

Usage

```
SUM(value)

COUNT()

COUNTSAMP()

MIN(value)

MAX(value)

AVG(value)

MEDIAN(value)

STD(value)

BAG(value)

BAGD(value)

Q1(value)

Q2(value)

Q3(value)
```

Arguments

value	string identifying name of metadata or region attribute

Details

- SUM: It prepares input parameter to be passed to the library function sum, performing all the type conversions needed
- COUNT: It prepares input parameter to be passed to the library function count, performing all the type conversions needed
- COUNTSAMP: It prepares input parameter to be passed to the library function countsamp, performing all the type conversions needed. It is used only with group_by functions
- MIN: It prepares input parameter to be passed to the library function minimum, performing all the type conversions needed
- MAX: It prepares input parameter to be passed to the library function maximum, performing all the type conversions needed
- AVG: It prepares input parameter to be passed to the library function mean, performing all the type conversions needed
- MEDIAN: It prepares input parameter to be passed to the library function median, performing all the type conversions needed
- STD: It prepares input parameter to be passed to the library function standard deviation, performing all the type conversions needed
- BAG: It prepares input parameter to be passed to the library function bag; this function creates comma-separated strings of attribute values, performing all the type conversions needed
- BAGD: It prepares input parameter to be passed to the library function bagd; this function creates comma-separated strings of distinct attribute values, performing all the type conversions needed
- Q1: It prepares input parameter to be passed to the library function fist quartile, performing all the type conversions needed
- Q2: It prepares input parameter to be passed to the library function second quartile, performing all the type conversions needed
- Q3: It prepares input parameter to be passed to the library function third quartile, performing all the type conversions needed

Value

Aggregate object

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "exp" using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
exp = read_gmql(test_path)

## This statement copies all samples of exp dataset into res dataset, and
## then calculates new metadata attribute sum_score for each of them:
## sum_score is the sum of score values of the sample regions.
```

```
res = extend(exp, sum_score = SUM("score"))

## This statement copies all samples of exp dataset into res dataset,
## and then calculates new metadata attribute min_pvalue for each of them:
## min_pvalue is the minimum pvalue of the sample regions.

res = extend(exp, min_pvalue = MIN("pvalue"))

## This statement copies all samples of exp dataset into res dataset,
## and then calculates new metadata attribute max_score for each of them:
## max_score is the maximum score of the sample regions.

res = extend(exp, max_score = MAX("score"))

## The following cover operation produces output regions where at least 2
## and at most 3 regions of exp dataset overlap, having as resulting region
## attribute the average signal of the overlapping regions;
## the result has one sample for each input cell value.

res = cover(exp, 2, 3, groupBy = conds("cell"), avg_signal = AVG("signal"))

## This statement copies all samples of 'exp' dataset into 'out' dataset,
## and then for each of them it adds another metadata attribute, allScore,
## which is the aggregation comma-separated list of all the values
## that the region attribute score takes in the sample.

out = extend(exp, allScore = BAG("score"))

## This statement counts the regions in each sample and stores their number
## as value of the new metadata RegionCount attribute of the sample.

out = extend(exp, RegionCount = COUNT())

## This statement copies all samples of exp dataset into res dataset,
## and then calculates new metadata attribute std_score for each of them:
## std_score is the standard deviation of the score values of the sample
## regions.

res = extend(exp, std_score = STD("score"))

## This statement copies all samples of exp dataset into res dataset,
## and then calculates new metadata attribute m_score for each of them:
## m_score is the median score of the sample regions.

res = extend(exp, m_score = MEDIAN("score"))
```

`arrange` *Method arrange*

Description

Wrapper to GMQL ORDER operator

It is used to order either samples or sample regions or both, according to a set of metadata and/or region attributes. Order can be specified as ascending / descending for every attribute. The number of samples and their regions remain the same (unless fetching options are specified), as well as their attributes, but a new ordering metadata and/or region attribute is added. Sorted samples or regions have a new attribute "_order", added to their metadata, or "order" added to their regions, or to both of them as specified in input.

Usage

```
## S4 method for signature 'GMQLDataset'
arrange(.data, metadata_ordering = NULL,
  regions_ordering = NULL, fetch_opt = "", num_fetch = 0L,
  reg_fetch_opt = "", reg_num_fetch = 0L)
```

Arguments

`.data`	GMQLDataset class object
`metadata_ordering`	list of ordering functions containing name of metadata attribute. The functions available are: `ASC`, `DESC`. If NULL, fetch_opt and num_fetch are not considered
`regions_ordering`	list of ordering functions containing name of region attribute. The functions available are: `ASC`, `DESC`. If NULL, reg_fetch_opt and reg_num_fetch are not considered
`fetch_opt`	string indicating the option used to fetch the first k samples; it can assume the values: • mtop: it fetches the first k samples • mtopp: it fetches the first k percentage of samples. • mtopg: it fetches the first k samples in each group. if NULL, *num_fetch* is not considered
`num_fetch`	integer value identifying the number of samples to fetch; by default it is 0, that means all samples are fetched
`reg_fetch_opt`	string indicating the option used to fetch the first k regions; it can assume the values: • rtop: it fetches the first k regions. • rtopp: it fetches the first k percentage of regions. • rtopg: it fetches the first k regions in each group. if NULL, *reg_num_fetch* is not considered
`reg_num_fetch`	integer value identifying the number of regions to fetch; by default it is 0, that means all regions are fetched

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
```

```
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named
## "data" using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data = read_gmql(test_path)

## The following statement orders the samples according to the Region_Count
## metadata attribute and takes the two samples that have the highest count.

o = arrange(data, list(ASC("Region_Count")), fetch_opt = "mtop",
    num_fetch = 2)
```

`collect` *Method collect*

Description

Wrapper to GMQL MATERIALIZE operator

It saves the content of a dataset that contains samples metadata and regions. It is normally used to persist the content of any dataset generated during a GMQL query. Any dataset can be materialized, but the operation can be time-consuming. For best performance, materialize the relevant data only.

Usage

```
## S4 method for signature 'GMQLDataset'
collect(x, dir_out = getwd(), name = "ds1")
```

Arguments

`x`	GMQLDataset class object
`dir_out`	destination folder path. By default it is the current working directory of the R process
`name`	name of the result dataset. By default it is the string "ds1"

Details

An error occures if the directory already exist at the destination folder path

Value

None

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named
## "data" using CustomParser
```

```
init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data = read_gmql(test_path)

## The following statement materializes the dataset 'data', previoulsy read,
## at the specific destination test_path into local folder "ds1" opportunely
## created

collect(data, dir_out = test_path)
```

`compile_query` *Compile GMQL query*

Description

It compiles a GMQL query taken from file or inserted as text string, using the proper GMQL web service available on a remote server

Usage

```
compile_query(url, query)

compile_query_fromfile(url, filePath)
```

Arguments

`url`	string url of server: It must contain the server address and base url; service name is added automatically
`query`	string text of a GMQL query
`filePath`	string path of txt file containing a GMQL query

Value

None

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## This statement gets the query as text string and runs the compile
## web service

compile_query(remote_url, "DATASET = SELECT() Example_Dataset_1;
    MATERIALIZE DATASET INTO RESULT_DS;")


## This statement defines the path to the file "query1.txt" in the
```

```
## subdirectory "example" of the package "RGMQL" and run the compile
## web service

test_path <- system.file("example", package = "RGMQL")
test_query <- file.path(test_path, "query1.txt")
compile_query_fromfile(remote_url, test_query)

## Logout from GMQL REST services suite

logout_gmql(remote_url)
```

`cover` *Method cover*

Description

Wrapper to GMQL COVER operator

It takes as input a dataset containing one or more samples and returns another dataset (with a single sample, if no *groupBy* option is specified) by "collapsing" the input dataset samples and their regions according to certain rules specified by the input parameters. The attributes of the output genomic regions are only the region coordinates, and Jaccard indexes (*JaccardIntersect* and *JaccardResult*). Jaccard Indexes are standard measures of similarity of the contributing regions, added as default region attributes. The JaccardIntersect index is calculated as the ratio between the lengths of the intersection and of the union of the contributing regions; the JaccardResult index is calculated as the ratio between the lengths of the result and the union of the contributing regions. If aggregate functions are specified, a new region attribute is added for each aggregate function specified. Output metadata are the union of the input ones. If *groupBy* clause is specified, the input samples are partitioned in groups, each with distinct values of the grouping metadata attributes, and the *cover* operation is separately applied to each group, yielding to one sample in the result for each group. Input samples that do not satisfy the *groupBy* condition are disregarded.

Usage

```
cover(.data, ...)

## S4 method for signature 'GMQLDataset'
cover(.data, min_acc, max_acc, groupBy = conds(),
  variation = "cover", ...)
```

Arguments

`.data`	GMQLDataset class object
`...`	a series of expressions separated by comma in the form *key* = *aggregate*. The *aggregate* is an object of class AGGREGATES. The aggregate functions available are: SUM, COUNT, MIN, MAX, AVG, MEDIAN, STD, BAG, BAGD, Q1, Q2, Q3. Every aggregate accepts a string value, except for COUNT, which does not have any value. Argument of 'aggregate function' must exist in schema, i.e. among region attributes. Two styles are allowed: • list of key-value pairs: e.g. sum = SUM("pvalue") • list of values: e.g. SUM("pvalue")

"mixed style" is not allowed

`min_acc` minimum number of overlapping regions to be considered during execution. It is an integer number, declared also as string. minAcc accepts also:

- PARAMETER class object: `ALL`, that represents the number of samples in the input dataset
- an expression built using PARAMETER object: (ALL() + N) / K or ALL() / K, with N and K integer values

`max_acc` maximum number of overlapping regions to be considered during execution. It is an integer number, declared also as string. maxAcc accept also:

- PARAMETER class object: `ALL`, that represents the number of samples in the input dataset
- PARAMETER class object: `ANY`, that acts as a wildcard, considering any amount of overlapping regions.
- an expression built using PARAMETER object: (ALL() + N) / K or ALL() / K, with N and K integer values

`groupBy` `conds` function to support methods with groupBy or JoinBy input parameter

`variation` string identifying the cover GMQL operator variation. The admissible strings are:

- FLAT: It returns the regions that start from the first end and stop at the last end of the regions which would contribute to each region of the *cover*.
- SUMMIT: It returns regions that start from a position where the number of intersecting regions is not increasing afterwards and stop at a position where either the number of intersecting regions decreases, or it violates the max accumulation index.
- HISTOGRAM: It returns the non-overlapping regions contributing to the *cover*, each with its accumulation index value, which is assigned to the *AccIndex* region attribute.
- COVER: default value.

It can be all caps or lowercase

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named "exp"
## using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
exp = read_gmql(test_path)

## The following statement produces an output dataset with a single output
## sample. The COVER operation considers all areas defined by a minimum
## of two overlapping regions in the input samples, up to any amount of
## overlapping regions.
```

```
res = cover(exp, 2, ANY())

## The following GMQL statement computes the result grouping the input
## exp samples by the values of their cell metadata attribute,
## thus one output res sample is generated for each cell value;
## output regions are produced where at least 2 and at most 3 regions
## of grouped exp samples overlap, setting as attributes of the resulting
## regions the minimum pvalue of the overlapping regions (min_pvalue)
## and their Jaccard indexes (JaccardIntersect and JaccardResult).

res = cover(exp, 2, 3, groupBy = conds("cell"), min_pValue = MIN("pvalue"))
```

Cover-Param *PARAM object class constructor*

Description

This class constructor is used to create instances of PARAM object to be used in GMQL cover method

Usage

```
ALL()

ANY()
```

Details

- ALL: It represents the number of samples in the input dataset.
- ANY: It represents any amount of overlapping regions to be considered.

Value

Param object

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the file "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named "exp"
## using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
exp = read_gmql(test_path)

## The following statement produces an output dataset with a single
## output sample. The COVER operation considers all areas defined by
## a minimum of two overlapping regions in the input samples,
```

```
## up to maximum amount of overlapping regions equal to the number
## of input samples.

res = cover(exp, 2, ALL())

## The following statement produces an output dataset with a single
## output sample. The COVER operation considers all areas defined by
## a minimum of two overlapping regions in the input samples,
## up to any amount of overlapping regions.

res = cover(exp, 2, ANY())

## The following statement produces an output dataset with a single
## output sample. The COVER operation considers all areas defined by
## minimum of overlapping regions in the input samples equal to half of
## the number of input samples, up to any amount of overlapping regions.

res = cover(exp, ALL()/2, ANY())
```

`delete_dataset` *Delete dataset*

Description

It deletes single private dataset specified by name from remote repository using the proper GMQL web service available on a remote server

Usage

```
delete_dataset(url, datasetName)
```

Arguments

`url`	string url of server: It must contain the server address and base url; service name is added automatically
`datasetName`	string name of dataset to delete

Details

If no error occurs, it prints "Deleted Dataset", otherwise a specific error is printed

Value

None

Examples

```
## Not run:

## This dataset does not exist

remote_url <- "http://www.gmql.eu/gmql-rest/"
```

```
login_gmql(remote_url)
delete_dataset(remote_url, "test1_20170604_180908_RESULT_DS")


## End(Not run)
```

DISTAL-Object *DISTAL object class constructor*

Description

This class constructor is used to create instances of DISTAL object to be used in GMQL JOIN operations (RGMQL merge functions) that use genometric predicate parameter requiring distal condition on value

Usage

```
DL(value)

DG(value)

DLE(value)

DGE(value)

MD(value)

UP()

DOWN()
```

Arguments

value	string identifying distance between genomic regions in base pair

Details

- DL: It denotes the less distance clause, which selects all the regions of a joined experiment dataset sample such that their distance from the anchor region of the joined reference dataset sample is less than 'value' bases.
- DLE: It denotes the less equal distance clause, which selects all the regions of a joined experiment dataset sample such that their distance from the anchor region of the joined reference dataset sample is less than, or equal to, 'value' bases.
- DG: It denotes the great distance clause, which selects all the regions of a joined experiment dataset sample such that their distance from the anchor region of the joined reference dataset sample is greater than 'value' bases.
- DGE: It denotes the great equal distance clause, which selects all the regions of a joined experiment dataset sample such that their distance from the anchor region of the joined reference dataset sample is greater than, or equal to, 'value' bases.

- MD: It denotes the minimum distance clause, which selects the first 'value' regions of the joined experiment at minimial distance from the anchor region of the joined reference dataset sample.
- UP: It denotes the upstream direction of the genome. It makes predicates to be hold on the upstream of the regions of the joined reference dataset sample. UP is true when region of the joined experiment dataset sample is in the upstream genome of the anchor region of the joined reference dataset sample. When this clause is not present, distal conditions apply to both directions of the genome.
- DOWN: It denotes the downstream direction of the genome. It makes predicates to be hold on the downstream of the regions of the joined reference dataset sample. DOWN is true when region of the joined experiment dataset sample is in the downstream genome of the anchor region of the joined reference dataset sample. When this clause is not present, distal conditions apply to both directions of the genome.

Value

Distal object

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folders "DATASET" and "DATASET_GDM" in the subdirectory
## "example" of the package "RGMQL", and opens such folders as a GMQL
## datasets named "TSS" and "HM", respectively, using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
TSS = read_gmql(test_path)
HM = read_gmql(test_path2)

## Given a dataset HM and one called TSS with a sample including
## Transcription Start Site annotations, this statement  searches for those
## regions of HM that are at a minimal distance from a transcription
## start site (TSS) and takes the first/closest one for each TSS, provided
## that such distance is lesser than 1200 bases and joined TSS and HM
## samples are obtained from the same provider (joinby clause).

join_data = merge(TSS, HM,
    genometric_predicate = list(MD(1), DL(1200)), conds("provider"),
    region_output = "RIGHT")

## Given a dataset HM and one called TSS with a sample including
## Transcription Start Site annotations, this statement searches for those
## regions of HM that are downstream and at a minimal distance from a
## transcription start site (TSS) and takes the first/closest one for each
## TSS, provided that such distance is greater than 12K bases and joined
## TSS and HM samples are obtained from the same provider (joinby clause).

join_data = merge(TSS, HM,
    genometric_predicate = list(MD(1), DGE(12000), DOWN()),
    conds("provider"), region_output = "RIGHT")
```

download_dataset *Download Dataset*

Description

It donwloads private dataset as zip file from remote repository to local path, or donwloads and saves it into R environment as GRangesList, using the proper GMQL web service available on a remote server

Usage

```
download_dataset(url, datasetName, path = getwd())

download_as_GRangesList(url, datasetName)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
datasetName	string name of dataset to download
path	string local path folder where to store dataset, by default it is R working directory

Details

If error occurs, a specific error is printed

Value

None

GRangesList containing all GMQL samples in dataset

Examples

```
## Download dataset in R working directory
## In this case we try to download a dataset of the user
## (public datasets from remote repository cannot be downloaded)

## Not run:

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)
download_dataset(remote_url, "Example_Dataset_1", path = getwd())

## Create GRangesList from user dataset Example_Dataset1 got
## from repository

download_as_GRangesList(remote_url, "Example_Dataset_1")

## End(Not run)
```

Evaluation-Function *Condition evaluation functions*

Description

This function is used to support joinBy and/or groupBy function parameter.

Usage

```
conds(default = c(""), full = c(""), exact = c(""))
```

Arguments

default	concatenation of string identifying a name of metadata attribute to be evaluated. It defines a DEFAULT evaluation of the input values. DEFAULT evaluation: the two attributes match if both end with value.
full	concatenation of string identifying a name of metadata attribute to be evaluated. It defines a FULL (FULLNAME) evaluation of the input values. FULL evaluation: two attributes match if they both end with value and, if they have further prefixes, the two prefix sequences are identical.
exact	concatenation of string identifying a name of metadata attribute to be evaluated. It defines a EXACT evaluation of the input values. EXACT evaluation: only attributes exactly as value match; no further prefixes are allowed.

Value

list of 2-D array containing method of evaluation and metadata attribute name

execute *GMQL Function: EXECUTE*

Description

It executes GMQL query. The function works only after invoking at least one collect

Usage

```
execute()
```

Value

None

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "data"

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data = read_gmql(test_path)

## The following statement materializes the dataset "data", previoulsy read,
## at the specific destination test_path into local folder "ds1" opportunely
## created

collect(data, dir_out = test_path)

## This statement executes GMQL query.
## Not run:

execute()

## End(Not run)
```

`export_gmql` *Create GMQL dataset from GRangesList*

Description

It creates GMQL dataset from GRangesList. All samples are in GDM (tab-separated values) or GTF file format

Usage

```
export_gmql(samples, dir_out, is_gtf)
```

Arguments

`samples`	GRangesList
`dir_out`	folder path where to create a folder and write the sample files
`is_gtf`	logical value indicating if samples have to be exported with GTF or GDM format

Details

The GMQL dataset is made up by two different file types:

- metadata files: they contain metadata associated with corrisponding sample.
- region files: they contain genomic regions data.
- region schema file: XML file that contains region attribute names (e.g. chr, start, end, pvalue)

Sample region files and metadata files are associated through file name: for example S_0001.gdm for region file and S_0001.gdm.meta for its metadata file

Value

None

See Also

import_gmql

Examples

```
## Load and attach add-on GenomicRanges package
library(GenomicRanges)

## These statemens create two GRanges with the region attributes: seqnames,
## ranges (region coordinates) and strand, plus two column elements:
## score and GC

gr1 <- GRanges(seqnames = "chr2", ranges = IRanges(3, 6), strand = "+",
    score = 5L, GC = 0.45)
gr2 <- GRanges(seqnames = c("chr1", "chr1"),
    ranges = IRanges(c(7,13), width = 3), strand = c("+", "-"),
    score = 3:4, GC = c(0.3, 0.5))

## This statement creates a GRangesList using the previous GRanges

grl = GRangesList(gr1, gr2)

## This statement defines the path to the subdirectory "example" of the
## package "RGMQL" and exports the GRangesList as GMQL datasets with sample
## files in GTF file format, using the last name of 'dir_out' path as
## dataset name

test_out_path <- system.file("example", package = "RGMQL")
export_gmql(grl, test_out_path, TRUE)
```

---

extend *Method extend*

Description

Wrapper to GMQL EXTEND operator

For each sample in an input dataset, it generates new metadata attributes as result of aggregate functions applied to sample region attributes and adds them to the existing metadata attributes of the sample. Aggregate functions are applied sample by sample.

Usage

```
extend(.data, ...)

## S4 method for signature 'GMQLDataset'
extend(.data, ...)
```

Arguments

.data	GMQLDataset class object
...	a series of expressions separated by comma in the form *key* = *aggregate*. The *aggregate* is an object of class AGGREGATES. The aggregate functions available are: SUM, COUNT, MIN, MAX, AVG, MEDIAN, STD, BAG, BAGD, Q1, Q2, Q3. Every aggregate accepts a string value, except for COUNT, which does not have any value. Argument of 'aggregate function' must exist in schema, i.e. among region attributes. Two styles are allowed: • list of key-value pairs: e.g. sum = SUM("pvalue") • list of values: e.g. SUM("pvalue") "mixed style" is not allowed

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "data"

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data <- read_gmql(test_path)

## This statement counts the regions in each sample and stores their number
## as value of the new metadata attribute RegionCount of the sample.

e <- extend(data, RegionCount = COUNT())

## This statement copies all samples of data dataset into 'res' dataset,
## and then calculates for each of them two new metadata attributes:
##  1. RegionCount is the number of sample regions;
##  2. MinP is the minimum pvalue of the sample regions.
## res sample regions are the same as the ones in data.

res = extend(data, RegionCount = COUNT(), MinP = MIN("pvalue"))
```

---

filter *Method filter*

Description

Wrapper to GMQL SELECT operator

It creates a new dataset from an existing one by extracting a subset of samples and/or regions from the input dataset according to the predicate. Each sample in the output dataset has the same

region attributes, values, and metadata as in the input dataset. When semijoin function is defined, it extracts those samples containing all metadata attributes defined in semijoin clause with at least one metadata value in common with semijoin dataset. If no metadata in common between input dataset and semijoin dataset, no sample is extracted.

Usage

```
## S4 method for signature 'GMQLDataset'
filter(.data, m_predicate = NULL,
  r_predicate = NULL, semijoin = NULL)
```

Arguments

.data	GMQLDataset class object
m_predicate	logical predicate made up by R logical operations on metadata attributes. Only !, |, ||, &, && are admitted.
r_predicate	logical predicate made up by R logical operations on region attributes. Only !, |, ||, &, && are admitted.
semijoin	semijoin function to define filter method with semijoin condition (see examples).

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "data"

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data <- read_gmql(test_path)

## This statement selects from input the data samples of patients younger
## than 70 years old, based on filtering on sample metadata attribute
## 'patient_age'

filter_data <- filter(data, patient_age < 70)

## This statement defines the path to the folder "DATASET_GDM" in the
## subdirectory "example" of the package "RGMQL" and opens such folder
## as a GMQL dataset named "join_data"

test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
join_data <- read_gmql(test_path2)

## This statement creates a new dataset called 'jun_tf' by selecting those
## samples and their regions from the existing 'data' dataset such that:
## Each output sample has a metadata attribute called antibody_target
## with value JUN.
```

```
## Each output sample also has not a metadata attribute called "cell"
## that has the same value of at least one of the values that a metadata
## attribute equally called cell has in at least one sample
## of the 'join_data' dataset.
## For each sample satisfying previous conditions, only its regions that
## have a region attribute called 'pvalue' with the associated value
## less than 0.01 are conserved in output

jun_tf <- filter(data, antibody_target == "JUN", pvalue < 0.01,
    semijoin(join_data, FALSE, conds("cell")))
```

`filter_and_extract` *Filter and extract function*

Description

This function lets user to create a new GRangesList with fixed information: seqnames, ranges and strand, and a variable part made up by the regions defined as input. The metadata and metadata_prefix are used to filter the data and choose only the samples that match at least one metdatata with its prefix. The input regions are shown for each sample obtained from filtering.

Usage

```
filter_and_extract(data, metadata = NULL, metadata_prefix = NULL,
  region_attributes = NULL, suffix = "antibody_target")
```

Arguments

`data`	string GMQL dataset folder path or GRangesList object
`metadata`	vector of strings containing names of metadata attributes to be searched for in metadata files. Data will be extracted if at least one condition is satisfied: this condition is logically "ANDed" with prefix filtering (see below) if NULL no filtering action occures (i.e every sample is taken for region filtering)
`metadata_prefix`	vector of strings that will support the metadata filtering. If defined, each 'metadata' is concatenated with the corresponding prefix.
`region_attributes`	vector of strings that extracts only region attributes specified; if NULL no regions attribute is taken and the output is only GRanges made up by the region coordinate attributes (seqnames, start, end, strand)
`suffix`	name for each metadata column of GRanges. By default it is the value of the metadata attribute named "antibody_target". This string is taken from sample metadata file or from metadata() associated. If not present, the column name is the name of selected regions specified by 'region_attributes' input parameter

Details

This function works only with datatset or GRangesList all whose samples or Granges have the same region coordinates (chr, ranges, strand) ordered in the same way for each sample

In case of GRangesList data input, the function searches for metadata into metadata() function associated to GRangesList.

Value

GRanges with selected regions

Examples

```
## This statement defines the path to the folder "DATASET" in the
## subdirectory "example" of the package "RGMQL" and filters such folder
## dataset including at output only "pvalue" and "peak" region attributes

test_path <- system.file("example", "DATASET", package = "RGMQL")
filter_and_extract(test_path, region_attributes = c("pvalue", "peak"))

## This statement imports a GMQL dataset as GRangesList and filters it
## including at output only "pvalue" and "peak" region attributes, the sort
## function makes sure that the region coordinates (chr, ranges, strand)
## of all samples are ordered correctly


grl = import_gmql(test_path, TRUE)
sorted_grl = sort(grl)
filter_and_extract(sorted_grl, region_attributes = c("pvalue", "peak"))
```

---

group_by *Method group_by*

Description

Wrapper to GMQL GROUP operator

It performs the grouping of samples and/or sample regions of the input dataset based on one specified metadata and/or region attribute. If the metadata attribute is multi-value, i.e., it assumes multiple values for sample (e.g., both <disease, cancer> and <disease, diabetes>), the grouping identifies different groups of samples for each attribute value combination (e.g., group1 for samples that feature the combination <disease, cancer>, group2 for samples that feature the combination <disease, diabetes>, and group3 for samples that feature both combinations <disease, cancer> and <disease, diabetes>). For each obtained group, it is possible to request the evaluation of aggregate functions on metadata attributes; these functions consider the metadata contained in all samples of the group. The regions, their attributes and their values in output are the same as the ones in input for each sample, and the total number of samples does not change. All metadata in the input samples are conserved with their values in the output samples, with the addition of the "_group" attribute, whose value is the identifier of the group to which the specific sample is assigned; other metadata attributes can be added as aggregate functions computed on specified metadata. When used on region attributes, group_by can group regions of each sample individually, based on their coordinates (chr, start, stop, strand) and possibly also on other specified grouping region attributes (when these are present in the schema of the input dataset). In each sample, regions found in the same group (i.e., regions with same coordinates and grouping attribute values), are combined into a single region; this allows to merge regions that are duplicated inside the same sample (based on the values of their coordinates and of other possible specified region attributes). For each grouped region, it is possible to request the evaluation of aggregate functions on other region attributes (i.e., which are

not coordinates, or grouping region attributes). This use is independent on the possible grouping realised based on metadata. The generated output schema only contains the original region attributes on which the grouping has been based, and additionally the attributes in case calculated as aggregated functions. If the group_by is applied only on regions, the output metadata and their values are equal to the ones in input. Both when applied on metadata and on regions, the group_by operation returns a number of output samples equal to the number of input ones. Note that the two possible uses of group_by, on metadata and on regions, are perfectly orthogonal, therefore they can be used in combination or independently.

Usage

```
## S4 method for signature 'GMQLDataset'
group_by(.data, groupBy_meta = conds(),
  groupBy_regions = c(""), region_aggregates = NULL,
  meta_aggregates = NULL)
```

Arguments

`.data`	GMQLDataset object
`groupBy_meta`	`conds` function to support methods with groupBy or JoinBy input parameter
`groupBy_regions`	vector of strings made up by region attribute names
`region_aggregates`	It accepts a list of aggregate functions on region attribute. All the elements in the form *key* = *aggregate*. The *aggregate* is an object of class AGGREGATES. The aggregate functions available are: `SUM`, `COUNT`, `MIN`, `MAX`, `AVG`, `MEDIAN`, `STD`, `BAG`, `BAGD`, `Q1`, `Q2`, `Q3`. Every aggregate accepts a string value, except for COUNT, which does not have any value. Argument of 'aggregate function' must exist in schema, i.e. among region attributes. Two styles are allowed: • list of key-value pairs: e.g. sum = SUM("pvalue") • list of values: e.g. SUM("pvalue") "mixed style" is not allowed
`meta_aggregates`	It accepts a list of aggregate functions on metadata attribute. All the elements in the form *key* = *aggregate*. The *aggregate* is an object of class AGGREGATES. The aggregate functions available are: `SUM`, `COUNTSAMP`, `MIN`, `MAX`, `AVG`, `MEDIAN`, `STD`, `BAG`, `BAGD`, `Q1`, `Q2`, `Q3`. Every aggregate accepts a string value, except for COUNTSAMP, which does not have any value. Argument of 'aggregate function' must exist in schema, i.e. among region attributes. Two styles are allowed: • list of key-value pairs: e.g. sum = SUM("cell") • list of values: e.g. SUM("cell") "mixed style" is not allowed

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
```

```
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named "exp"
## using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
exp = read_gmql(test_path)

## This GMQL statement groups samples of the input 'exp' dataset according
## to their value of the metadata attribute 'tumor_type' and computes the
## maximum value that the metadata attribute 'size' takes inside the samples
## belonging to each group. The samples in the output GROUPS_T dataset
## have a new _group metadata attribute which indicates which group they
## belong to, based on the grouping on the metadata attribute tumor_type.
## In addition, they present the new metadata aggregate attribute 'MaxSize'.
## Note that the samples without metadata attribute 'tumor_type' are
## assigned to a single group with _group value equal 0

GROUPS_T = group_by(exp, conds("tumor_type"),
    meta_aggregates = list(max_size = MAX("size")))

## This GMQL statement takes as input dataset the same input dataset as
## the previous example. Yet, it calculates new _group values based on the
## grouping attribute 'cell', and adds the metadata aggregate attribute
## 'n_samp', which counts the number of samples belonging to the respective
## group. It has the following output GROUPS_C dataset samples
## (note that now no sample has metadata attribute _group with value
## equal 0 since all input samples include the metadata attribute cell,
## with different values, on which the new grouping is based)

GROUPS_C = group_by(exp, conds("cell"),
    meta_aggregates = list(n_samp = COUNTSAMP()))

## This GMQL statement groups the regions of each 'exp' dataset sample by
## region coordinates chr, left, right, strand  (these are implicitly
## considered) and the additional region attribute score (which is
## explicitly specified), and keeps only one region for each group.
## In the output GROUPS dataset schema, the new region attributes
## avg_pvalue and max_qvalue are added, respectively computed as the
## average of the values taken by the pvalue and the maximum of the values
## taken by the qvalue region attributes in the regions grouped together,
## and the computed value is assigned to each region of each output sample.
## Note that the region attributes which are not coordinates or score are
## discarded.

GROUPS = group_by(exp, groupBy_regions = "score",
    region_aggregates = list(avg_pvalue = AVG("pvalue"),
    max_qvalue = MAX("qvalue")))
```

`import_gmql` *Create GRangesList from GMQL dataset*

Description

It creates a GRangesList from GMQL samples in dataset. It reads sample files in GTF or GDM/tab-delimited format.

Usage

```
import_gmql(dataset_path, is_gtf)
```

Arguments

dataset_path	string with GMQL dataset folder path
is_gtf	logical value indicating if dataset samples are in GTF format; if TRUE and dataset does not contain GTF samples, an error occurs

Value

GRangesList containing all GMQL samples in dataset

See Also

export_gmql

Examples

```
## This statement defines the path to the subdirectory "example" of the
## package "RGMQL" and imports as GRangesList the contained GMQL dataset

test_path <- system.file("example", "DATASET", package = "RGMQL")
grl = import_gmql(test_path, TRUE)
```

---

init_gmql *Init GMQL server*

Description

It initializes and runs GMQL server for executing GMQL query. It also performs a login to GMQL REST services suite, if needed

Usage

```
init_gmql(output_format = "GTF", remote_processing = FALSE,
  url = NULL, username = NULL, password = NULL)
```

Arguments

output_format	string that identifies the output format of all sample files. It can be TAB, GTF or COLLECT: • TAB: tab-delimited file format • GTF: tab-delimited text standard format based on the General Feature Format • COLLECT: used for storing output in memory (only in the case of local processing, i.e., remote_processing = FALSE)
remote_processing	logical value specifying the processing mode. True for processing on cluster (remote), false for local processing.
url	string url of server: It must contain the server address and base url; service name is added automatically. If NULL, no login is performed. You can always perform it by calling the function login_gmql explicitly
username	string name used during remote server signup
password	string password used during remote server signup

Value

None

Examples

```
## This statement initializes GMQL with local processing with sample files
## output format as tab-delimited

init_gmql("tab", FALSE)

## This statement initializes GMQL with remote processing

remote_url = "http://www.gmql.eu/gmql-rest/"
init_gmql(remote_processing = TRUE, url = remote_url)
```

---

login_gmql *Login to GMQL*

Description

Login to GMQL REST services suite as a registered user, specifying username and password, or as guest, using the proper GMQL web service available on a remote server

Usage

```
login_gmql(url, username = NULL, password = NULL)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
username	string name used during signup
password	string password used during signup

Details

If both username and password are missing, you will be logged as guest. After login you will receive an authentication token. As token remains valid on server (until the next login / registration or logout), a user can safely use a token for a previous session as a convenience; this token is saved in R Global environment to perform subsequent REST call even on complete R restart (if the environment has been saved). If error occurs, a specific error is printed

Value

None

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)
```

---

logout_gmql *Logout from GMQL*

Description

Logout from GMQL REST services suite using the proper GMQL web service available on a remote server

Usage

```
logout_gmql(url)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically

Details

After logout the authentication token will be invalidated. The authentication token is removed from R Global environment. If error occurs, a specific error is printed

Value

None

Examples

```
## Login to GMQL REST services suite as guest, then logout

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)
logout_gmql(remote_url)
```

---

log_job *Show a job log or trace*

Description

It shows a job log or traces a specific job

Usage

```
show_job_log(url, job_id)

trace_job(url, job_id)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
job_id	string id of the job

Details

If error occurs, a specific error is printed

Value

Log or trace text

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## List all jobs
list_jobs <- show_jobs_list(remote_url)

## Not run:
jobs_1 <- list_jobs$jobs[[1]]

## Show jobs_1 log
show_job_log(remote_url, jobs_1)

## Trace jobs_1
```

```
trace_job(remote_url, jobs_1)


## End(Not run)
```

---

map *Method map*

Description

Wrapper to GMQL MAP operator

It computes, for each sample in the right dataset, aggregates over the values of the right dataset regions that intersect with a region in a left dataset sample, for each region of each sample in the left dataset. The number of generated output samples is the Cartesian product of the samples in the two input datasets; each output sample has the same regions as the related input left dataset sample, with their attributes and values, plus the attributes computed as aggregates over right region values. Output sample metadata are the union of the related input sample metadata, whose attribute names are prefixed with 'left' or 'right' respectively.

Usage

```
map(x, y, ...)

## S4 method for signature 'GMQLDataset'
map(x, y, ..., joinBy = conds(),
  count_name = "")
```

Arguments

x	GMQLDataset class object
y	GMQLDataset class object
...	a series of expressions separated by comma in the form *key* = *aggregate*. The *aggregate* is an object of class AGGREGATES. The aggregate functions available are: SUM, COUNT, MIN, MAX, AVG, MEDIAN, STD, BAG, BAGD, Q1, Q2, Q3. Every aggregate accepts a string value, except for COUNT, which does not have any value. Argument of 'aggregate function' must exist in schema, i.e. among region attributes. Two styles are allowed: • list of key-value pairs: e.g. sum = SUM("pvalue") • list of values: e.g. SUM("pvalue") "mixed style" is not allowed
joinBy	conds function to support methods with groupBy or JoinBy input parameter
count_name	string defining the metadata count name; if it is not specified the name is "count_left_right"

Details

When the joinby clause is present, only pairs of samples of x dataset and of y dataset with metadata M1 and M2, respectively, that satisfy the joinby condition are considered.

The clause consists of a list of metadata attribute names that must be present with equal values in both M1 and M2

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folders "DATASET" and "DATASET_GDM" in the subdirectory
## "example" of the package "RGMQL", and opens such folders as a GMQL
## dataset named "exp" and "ref", respectively, using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
exp = read_gmql(test_path)
ref = read_gmql(test_path2)

## This statement counts the number of regions in each sample from exp
## dataset that overlap with a ref dataset region, and for each ref region
## it computes the minimum score of all the regions in each exp sample that
## overlap with it. The MAP joinBy option ensures that only the exp samples
## referring to the same 'cell_tissue' of a ref sample are mapped on such
## ref sample; exp samples with no cell_tissue metadata attribute, or with
## such metadata attribute, but with a different value from the one(s)
## of ref sample(s), are disregarded.

out = map(ref, exp, minScore = MIN("score"), joinBy = conds("cell_tissue"))
```

`merge` *Method merge*

Description

Wrapper to GMQL JOIN operator

It takes in input two datasets, respectively known as anchor (left) and experiment (right) and returns a dataset of samples consisting of regions extracted from the operands according to the specified conditions (a.k.a *genometric_predicate* and *region_attribute* predicate). The number of generated output samples is the Cartesian product of the number of samples in the anchor and in the experiment dataset (if *joinBy* is not specified). The output metadata are the union of the input metadata, with their attribute names prefixed with left or right dataset name, respectively.

Usage

```
## S4 method for signature 'GMQLDataset,GMQLDataset'
merge(x, y,
  genometric_predicate = NULL, region_output = "CAT",
  joinBy = conds(), reg_attr = c(""))
```

Arguments

x	GMQLDataset class object
y	GMQLDataset class object
genometric_predicate	it is a list of DISTAL objects. For details of DISTAL objects see: DLE, DGE, DL, DG, MD, UP, DOWN
region_output	single string that declares which region is given in output for each input pair of left dataset and right dataset regions satisfying the genometric predicate and/or the region attribute predicate: • LEFT: It outputs the anchor regions from 'x' that satisfy the genometric and/or region attribute predicate • RIGHT: It outputs the experiment regions from 'y' that satisfy the genometric and/or region attribute predicate • INT (intersection): It outputs the overlapping part (intersection) of the 'x' and 'y' regions that satisfy the genometric and/or region attribute predicate; if the intersection is empty, no output is produced • CAT: It outputs the concatenation between the 'x' and 'y' regions that satisfy the genometric and/or region attribute predicate, (i.e. the output regions defined as having left (right) coordinates equal to the minimum (maximum) of the corresponding coordinate values in the 'x' and 'y' regions satisfying the genometric and/or region attribute predicate) • LEFT_DIST: It outputs the duplicate elimination of 'x' output regions with the same coordinates and values, regardless the 'y' paired region and its values. In this case, the output region attributes and their values are all and only those of 'x', and the output metadata are equal to the 'x' metadata, without additional prefixes • RIGHT_DIST: It outputs the duplicate elimination of 'y' output regions with the same coordinates and values, regardless the 'x' paired region and its values. In this case, the output regions attributes and their values are all and only those of 'y', and the output metadata are equal to the 'y' metadata, without additional prefixes • BOTH: It outputs the same regions as LEFT, but it adds in the output region attributes the coordinates of the 'y' paired region that, together with the 'x' output region, satisfies the genometric and/or region attribute predicate
joinBy	condition_evaluation function to support methods with groupBy or JoinBy input paramter
reg_attr	vector of strings made up by region field attribute names, whose values in the paired left and right dataset regions must be equal in order to consider the two paired regions. If specified, *region_output* cannot be INT or CAT.

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folders "DATASET" and "DATASET_GDM" in the subdirectory
```

```
## "example" of the package "RGMQL" and opens such folders as a GMQL
## datasets named TSS and HM, respectively, using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
TSS = read_gmql(test_path)
HM = read_gmql(test_path2)

## Given a dataset HM and one called TSS with a sample including
## Transcription Start Site annotations, this statement searches for those
## regions of HM that are at a minimal distance from a transcription start
## site (TSS) and takes the first/closest one for each TSS, provided that
## such distance is lesser than 120K bases and joined TSS and HM
## samples are obtained from the same provider (joinby clause).

join_data = merge(TSS, HM, genometric_predicate = list(MD(1), DLE(120000)),
    conds("provider"), region_output = "RIGHT")
```

OPERATOR-Object *OPERATOR object class constructor*

Description

This class constructor is used to create instances of OPERATOR object, to be used in GMQL functions that require operator on value.

Usage

```
META(value, type = NULL)

NIL(type)

SQRT(value)
```

Arguments

value	string identifying name of metadata attribute
type	string identifying the type of the attribute value; it must be: INTEGER, DOUBLE or STRING. For NIL() function, only INTEGER and DOUBLE are allowed

Details

- META: It prepares input parameter to be passed to library function meta, performing all the type conversions needed
- SQRT: It prepares input parameter to be passed to library function sqrt, performing all the type conversions needed
- NIL: It prepares input parameter to be passed to library function null, performing all the type conversions needed

Value

Operator object

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "exp"

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
exp = read_gmql(test_path)

## This statement allows to select, in all input samples, all those regions
## for which the region attribute score has a value which is greater
## than the metadata attribute value "avg_score" in their sample.

data = filter(exp, r_predicate = score > META("avg_score"))

## This statement defines new numeric region attributes with "null" value.
## The syntax for creating a new attribute with null value is
## attribute_name = NULL(TYPE), where type may be INTEGER or DOUBLE.

out = select(exp, regions_update = list(signal = NIL("INTEGER"),
    pvalue = NIL("DOUBLE")))

## This statement allows to build an output dataset named 'out' such that
## all the samples from the input dataset 'exp' are conserved,
## as well as their region attributes (and their values)
## and their metadata attributes (and their values).
## The new metadata attribute 'concSq' is added to all output samples
## with value correspondent to the mathematical squared root
## of the pre-existing metadata attribute 'concentration'.

out = select(exp, metadata_update = list(concSq = SQRT("concentration")))
```

Ordering-Functions *Ordering functions*

Description

These functions are used to create a series of metadata as string that require ordering on value; it is used only in arrange method (see example).

Usage

```
DESC(...)

ASC(...)
```

Arguments

...	series of metatdata as string

Details

- ASC: It defines an ascending order for input value
- DESC: It defines a descending order for input value

Value

Ordering object

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such file as a GMQL dataset named
## "data" using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data = read_gmql(test_path)

## This statement orders the samples according to the Region_Count metadata
## attribute and takes the two samples that have the lowest count.

asc = arrange(data, list(ASC("Region_Count")), fetch_opt = "mtop",
    num_fetch = 2)

## This statement orders the regions of each samples according to their
## pvalue attribute value and in each sample it takes the first seven
## regions with the highest pvalue

desc = arrange(data, regions_ordering = list(DESC("pvalue")),
    reg_fetch_opt = "rtop", reg_num_fetch = 7)
```

---

read_gmql *Function read*

Description

It reads a GMQL dataset, as a folder containing some homogenus samples on disk or as a GRanges-List, saving it in Scala memory in a way that can be referenced in R. It is also used to read a repository dataset in case of remote processing.

Usage

```
read_gmql(dataset, parser = "CustomParser", is_local = TRUE,
  is_GMQL = TRUE)

read_GRangesList(samples)
```

Arguments

dataset	folder path for GMQL dataset or dataset name on repository
parser	string used to parsing dataset files. The Parsers available are: • BedParser • BroadPeakParser • NarrowPeakParser • CustomParser Default is CustomParser.
is_local	logical value indicating local or remote dataset
is_GMQL	logical value indicating GMQL dataset or not
samples	GRangesList

Details

Normally, a GMQL dataset contains an XML schema file that contains name of region attributes. (e.g chr, start, stop, strand) The CustomParser reads this XML schema; if you already know what kind of schema your files have, use one of the parsers defined, without reading any XML schema.

If GRangesList has no metadata: i.e. metadata() is empty, two metadata are generated:

- "provider" = "PoliMi"
- "application" = "RGMQL"

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "data" using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data = read_gmql(test_path)

## This statement opens such folder as a GMQL dataset named "data" using
## "NarrowPeakParser"
dataPeak = read_gmql(test_path,"NarrowPeakParser")

## This statement reads a remote public dataset stored into GMQL system
## repository. For a public dataset in a (remote) GMQL repository the
## prefix "public." is needed before dataset name

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)
data1 = read_gmql("public.Example_Dataset_1", is_local = FALSE)
```

`register_gmql` *Register into remote GMQL*

Description

Register to GMQL REST services suite using the proper GMQL web service available on a remote server.

Usage

```
register_gmql(url, username, psw, email, first_name, last_name)
```

Arguments

`url`	string url of server: It must contains the server address and base url; service name is added automatically
`username`	string user name used to login in
`psw`	string password used to login in
`email`	string user email
`first_name`	string user first name
`last_name`	string user last name

Details

After registration you will receive an authentication token. As token remains valid on server (until the next login / registration or logout), a user can safely use a token for a previous session as a convenience; this token is saved in R Global environment to perform subsequent REST calls or batch processing even on complete R restart (if the environment has been saved). If error occurs, a specific error is printed.

Value

None

Examples

```
## Register to GMQL REST services suite

remote_url = "http://www.gmql.eu/gmql-rest/"
## Not run:
register_gmql(remote_url,"foo","foo","foo@foo.com","foo","foo")

## End(Not run)
```

remote_processing *Disable or Enable remote processing*

Description

It allows to enable or disable remote processing

Usage

```
remote_processing(is_remote)
```

Arguments

is_remote	logical value used in order to set the processing mode. TRUE: you set a remote query processing mode, otherwise it will be local

Details

The invocation of this function allows to change mode of processing. After invoking collect() function, it is not possbile to switch the processing mode.

Value

None

Examples

```
## This statement initializes GMQL with local processing with sample
## files output format as tab-delimited, and then it changes processing
## mode to remote

init_gmql("tab", remote_processing = FALSE)

remote_processing(TRUE)
```

run_query *Run a GMQL query*

Description

It runs a GMQL query into repository taken from file or inserted as text string, using the proper GMQL web service available on a remote server

Usage

```
run_query(url, queryName, query, output_gtf = TRUE)

run_query_fromfile(url, filePath, output_gtf = TRUE)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
queryName	string name of the GMQL query file
query	string text of the GMQL query
output_gtf	logical value indicating file format used for storing samples generated by the query. The possiblities are: • GTF • TAB
filePath	string path of a txt file containing a GMQL query

Details

If error occurs, a specific error is printed

Value

None

Examples

```
## Not run:

## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## Run query as string input parameter
## NOTE: not very suitable for long queries

run_query(remote_url, "query_1", "DATASET = SELECT() Example_Dataset1;
    MATERIALIZE DATASET INTO RESULT_DS;", output_gtf = FALSE)

## With system.file() this statement defines the path to the folder
## "example" of the package "RGMQL", and then it executes the query
## written in the text file "query1.txt"

test_path <- system.file("example", package = "RGMQL")
test_query <- file.path(test_path, "query1.txt")
run_query_fromfile(remote_url, test_query, output_gtf = FALSE)

## End(Not run)
```

sample_metadata *Show metadata list from dataset sample*

Description

It retrieves metadata of a specific sample in dataset using the proper GMQL web service available on a remote server

Usage

```
sample_metadata(url, datasetName, sampleName)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
datasetName	string name of dataset of interest
sampleName	string name of sample of interest

Details

If error occurs, a specific error is printed

Value

List of metadata in the form 'key = value'

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## This statement retrieves metadata of sample 'S_00000' from public
## dataset 'Example_Dataset_1'

sample_metadata(remote_url, "public.Example_Dataset_1", "S_00000")
```

sample_region *Show regions data from a dataset sample*

Description

It retrieves regions data of a specific sample (whose name is specified in the parameter "sampleName") in a specific dataset (whose name is specified in the parameter "datasetName") using the proper GMQL web service available on a remote server

Usage

```
sample_region(url, datasetName, sampleName)
```

Arguments

url	string url of server. It must contain the server address and base url; service name is added automatically
datasetName	string name of dataset of interest
sampleName	string name of sample of interest

Details

If error occurs, a specific error is printed

Value

GRanges data containing regions of sample

Examples

```
## Not run:

## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## This statement retrieves regions data of sample "S_00000" from public
## dataset "Example_Dataset_1"

sample_region(remote_url, "public.Example_Dataset_1", "S_00000")


## End(Not run)
```

---

save_query *Save GMQL query*

Description

It saves a GMQL query into repository, taken from file or inserted as text string, using the proper GMQL web service available on a remote server

Usage

```
save_query(url, queryName, queryTxt)

save_query_fromfile(url, queryName, filePath)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
queryName	string name of query
queryTxt	string text of GMQL query
filePath	string local file path of a txt file containing a GMQL query

Details

If you save a query with the same name of another query already stored in repository, you will overwrite it; if no error occurs, it prints: "Saved", otherwise it prints the error

Value

None

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## This statement saves query written directly as input string parameter
## with name "dna_query"

save_query(remote_url, "example_query",
"DATASET = SELECT() Example_Dataset_1; MATERIALIZE DATASET INTO RESULT_DS;")

## With system.file() this statement  defines the path to the folder
## "example" of the package "RGMQL", and then it saves the query written
## in the text file "query1.txt" into remote repository

test_path <- system.file("example", package = "RGMQL")
test_query <- file.path(test_path, "query1.txt")
save_query_fromfile(remote_url, "query1", test_query)
```

---

select *Method select*

Description

Wrapper to GMQL PROJECT operator

It creates, from an existing dataset, a new dataset with all the samples from input dataset, but keeping for each sample in the input dataset only those metadata and/or region attributes specified. Region coordinates and values of the remaining metadata and/or region attributes remain equal to those in the input dataset. It allows to:

- Remove existing metadata and/or region attributes from a dataset
- Update or set new metadata and/or region attributes in the result

Usage

```
## S4 method for signature 'GMQLDataset'
select(.data, metadata = NULL,
  metadata_update = NULL, all_but_meta = FALSE, regions = NULL,
  regions_update = NULL, all_but_reg = FALSE)
```

Arguments

.data	GMQLDataset class object
metadata	vector of strings made up by metadata attributes
metadata_update	list of updating rules in the form of key = value generating new metadata attributes and/or attribute values. The following options are available: • All aggregation functions already defined by AGGREGATES object • All basic mathematical operations (+, -, *, /), including parenthesis • SQRT constructor object defined by OPERATOR object
all_but_meta	logical value indicating which metadata you want to exclude; If FALSE, only the metadata attributes specified in *metadata* argument are kept in the output of the operation; if TRUE, the metadata are all kept except those in *metadata* argument. If *metadata* input parameter is not defined *all_but_meta* is not considerd.
regions	vector of strings made up by region attributes
regions_update	list of updating rules in the form of key = value generating new genomic region attributes and/or values. The following options are available: • All aggregation functions already defined by AGGREGATES object • All basic mathematical operations (+, -, *, /), including parenthesis • SQRT, META, NIL constructor objects defined by OPERATOR object
all_but_reg	logical value indicating which region attributes you want to exclude; if FALSE, only the regions attributes specified in *regions* argumentare kept in the output of the operation; if TRUE, the regions attributes are all kept except those in *regions* argument. If *regions* is not defined, *all_but_reg* is not considerd.

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "data"

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
data = read_gmql(test_path)

## This statement creates a new dataset called CTCF_NORM_SCORE by preserving
## all region attributes apart from score, and creating a new region
## attribute called new_score by dividing the existing score value of each
## region by 1000.0 and incrementing it by 100.
```

```
## It also generates, for each sample of the new dataset,
## a new metadata attribute called normalized with value 1,
## which can be used in future selections.


CTCF_NORM_SCORE = select(data, metadata_update = list(normalized = 1),
    regions_update = list(new_score = (score / 1000.0) + 100),
    regions = c("score"), all_but_reg = TRUE)
```

`semijoin` *Semijoin condition*

Description

This function is used as support to the filter method to define semijoin conditions on metadata

Usage

```
semijoin(.data, is_in = TRUE, groupBy)
```

Arguments

`.data`	GMQLDataset class object
`is_in`	logical value: TRUE => for a given sample of input dataset '.data' in `filter` method, if and only if there exists at least one sample in dataset 'data' with metadata attributes defined in groupBy and these attributes of 'data' have at least one value in common with the same attributes defined in at least one sample of '.data' in `filter` method, FALSE => semijoin condition is evaluated accordingly.
`groupBy`	`condition_evaluation` function to support methods with groupBy or JoinBy input paramter

Value

semijoin condition as list

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folders "DATASET" and "DATASET_GDM" in the subdirectory
## "example" of the package "RGMQL" and opens such folders as GMQL datasets
## named "data" and "join_data", respectively

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
data <- read_gmql(test_path)
join_data <-  read_gmql(test_path2)
```

```
## This statement creates a new dataset called 'jun_tf' by selecting those
## samples and their regions from the existing 'data' dataset such that:
## Each output sample has a metadata attribute called antibody_target
## with value JUN.
## Each output sample also has not a metadata attribute called cell
## that has the same value of at least one of the values that a metadata
## attribute equally called cell has in at least one sample
## of the 'join_data' dataset.
## For each sample satisfying previous conditions, only its regions that
## have a region attribute called pValue with the associated value
## less than 0.01 are conserved in output

jun_tf <- filter(data, antibody_target == "JUN", pvalue < 0.01,
    semijoin(join_data, FALSE, conds("cell")))
```

`setdiff` *Method setdiff*

Description

Wrapper to GMQL DIFFERENCE operator

It produces one sample in the result for each sample of the left operand, by keeping the same metadata of the left input sample and only those regions (with their attributes and values) of the left input sample which do not intersect with any region in any right operand sample. The optional *joinBy* clause is used to extract a subset of pairs from the Cartesian product of the two input datasets *x* and *y* on which to apply the DIFFERENCE operator: only those samples that have the same value for each specified metadata attribute are considered when performing the difference.

Usage

```
## S4 method for signature 'GMQLDataset,GMQLDataset'
setdiff(x, y, joinBy = conds(),
  is_exact = FALSE)
```

Arguments

`x`	GMQLDataset class object
`y`	GMQLDataset class object
`joinBy`	`conds` function to support methods with groupBy or JoinBy input parameter
`is_exact`	single logical value: TRUE means that the region difference is executed only on regions in 'x' dataset with exactly the same coordinates of at least one region present in 'y' dataset; if is_exact = FALSE, the difference is executed on all regions in 'x' dataset that overlap (even just one base) with at least one region in 'y' dataset

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folders "DATASET" and "DATASET_GDM" in the subdirectory
## "example" of the package "RGMQL" and opens such folders as a GMQL
## datasets named "data1" and "data2", respectively, using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
data1 = read_gmql(test_path)
data2 = read_gmql(test_path2)

## This statement returns all the regions in the first dataset
## that do not overlap any region in the second dataset.

out = setdiff(data1, data2)

## This statement extracts for every pair of samples s1 in data1
## and s2 in data2 having the same value of the metadata
## attribute 'cell' the regions that appear in s1 but
## do not overlap any region in s2.
## Metadata of the result are the same as the metadata of s1.

out_t = setdiff(data1, data2, conds("cell"))
```

---

show_datasets_list *Show datasets*

Description

It shows all GMQL datasets stored by the user or public in remote repository, using the proper GMQL web service available on a remote server

Usage

```
show_datasets_list(url)
```

Arguments

url	single string url of server: It must contain the server address and base url; service name is added automatically

Details

If error occurs, a specific error is printed

Value

List of datasets. Every dataset in the list is described by:

- name: name of dataset
- owner: public or name of the user

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## List all datasets

list <- show_datasets_list(remote_url)
```

---

show_jobs_list *Show all jobs*

Description

It shows all jobs (run, succeded or failed) invoked by the user on remote server using, the proper GMQL web service available on a remote server

Usage

```
show_jobs_list(url)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically

Details

If error occurs, a specific error is printed

Value

List of jobs. Every job in the list is described by:

- id: unique job identifier

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## List all jobs
list_jobs <- show_jobs_list(remote_url)
```

show_queries_list *Show all queries*

Description

It shows all the GMQL queries saved by the user on remote repository, using the proper GMQL web service available on a remote server

Usage

```
show_queries_list(url)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically

Details

If error occurs, a specific error is printed

Value

List of queries. Every query in the list is described by:

- name: name of query
- text: text of GMQL query

Examples

```
## Login to GMQL REST services suite
remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## List all queries executed on remote GMQL system
list <- show_queries_list(remote_url)
```

show_samples_list *Show dataset samples*

Description

It show all samples from a specific GMQL dataset on remote repository, using the proper GMQL web service available on a remote server

Usage

```
show_samples_list(url, datasetName)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
datasetName	name of dataset containing the samples whose list we like to get; if the dataset is a public dataset, we have to add "public." as prefix, as shown in the example below, otherwise no prefix is needed

Details

If error occurs, a specific error is printed

Value

List of samples in dataset. Every sample in the list is described by:

- id: id of sample
- name: name of sample
- path: sample repository path

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## This statement shows all samples present into public dataset
## 'Example_Dataset_1'

list <- show_samples_list(remote_url, "public.Example_Dataset_1")
```

---

show_schema *Show dataset schema*

Description

It shows the region attribute schema of a specific GMQL dataset on remote repository, using the proper GMQL web service available on a remote server

Usage

```
show_schema(url, datasetName)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
datasetName	name of dataset to get the schema; if the dataset is a public dataset, we have to add "public." as prefix, as shown in the example below, otherwise no prefix is needed

Details

If error occurs, a specific error is printed

Value

List of region schema fields. Every field in the list is described by:

- name: name of field (e.g. chr, start, end, strand, ...)
- fieldType: (e.g. STRING, DOUBLE, ...)

Examples

```
## Login to GMQL REST services suite as guest

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## Show schema of public dataset 'Example_Dataset_1'

list <- show_schema(remote_url, "public.Example_Dataset_1")
```

---

stop_gmql *Stop GMQL server*

Description

It stops GMQL server processing

Usage

```
stop_gmql()
```

Value

None

Examples

```
## This statement first initializes GMQL with local processing and with
## sample file output format as tab-delimited, and then stops it

init_gmql("tab", FALSE)

stop_gmql()
```

stop_job *Stop a job*

Description

It stops a specific current query job

Usage

```
stop_job(url, job_id)
```

Arguments

url	string url of server: It must contain the server address and base url; service name is added automatically
job_id	string id of the job

Details

If error occurs, a specific error is printed

Value

None

Examples

```
## Not run:

## Login to GMQL REST services suite at remote url

remote_url = "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## This statement shows all jobs at GMQL remote system and selects one
## running job, saving it into 'jobs_1' (in this case is the first of the
## list), and then stop it

list_jobs <- show_jobs_list(remote_url)
jobs_1 <- list_jobs$jobs[[1]]
stop_job(remote_url, jobs_1)

## End(Not run)
```

take *Method take*

Description

Wrapper to TAKE operation

It saves the content of a dataset that contains samples metadata and regions as GRangesList. It is normally used to store in memory the content of any dataset generated during a GMQL query. The operation can be very time-consuming. If you invoked any materialization before take function, all those datasets are materialized as folders.

Usage

```
take(.data, ...)

## S4 method for signature 'GMQLDataset'
take(.data, rows = 0L)
```

Arguments

.data	returned object from any GMQL function
...	Additional arguments for use in other specific methods of the generic take function
rows	number of regions rows for each sample that you want to retrieve and store in memory. By default it is 0, that means take all rows for each sample

Value

GRangesList with associated metadata

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folder "DATASET" in the subdirectory "example"
## of the package "RGMQL" and opens such folder as a GMQL dataset
## named "rd" using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
rd = read_gmql(test_path)

## This statement creates a dataset called 'aggr' which contains one
## sample for each antibody_target and cell value found within the metadata
## of the 'rd' dataset sample; each created sample contains all regions
## from all 'rd' samples with a specific value for their
## antibody_target and cell metadata attributes.

aggr = aggregate(rd, conds(c("antibody_target", "cell")))

## This statement performs the query and returns the resulted dataset as
## GRangesList named 'taken'. It returns only the first 45 regions of
```

```
## each sample present into GRangesList and all the medatata associated
## with each sample

taken <- take(aggr, rows = 45)
```

---

union *Method union*

Description

Wrapper to GMQL UNION operator

It is used to integrate samples of two homogeneous or heterogeneous datasets within a single dataset; for each sample of either input dataset, a result sample is created as follows:

- Metadata are the same as in the original sample.
- Resulting schema is the schema of the left input dataset.
- Regions are the same (in coordinates and attribute values) as in the original sample, if it is from the left input dataset; if it is from the right input dataset, its regions are the same in coordinates, but only region attributes identical (in name and type) to those of the left input dataset are retained, with the same values. Region attributes which are missing in an input dataset sample w.r.t. the merged schema are set to null.

Usage

```
## S4 method for signature 'GMQLDataset,GMQLDataset'
union(x, y)
```

Arguments

x	GMQLDataset object
y	GMQLDataset object

Value

GMQLDataset object. It contains the value to use as input for the subsequent GMQLDataset method

Examples

```
## This statement initializes and runs the GMQL server for local execution
## and creation of results on disk. Then, with system.file() it defines
## the path to the folders "DATASET" and "DATASET_GDM" in the subdirectory
## "example" of the package "RGMQL" and opens such folders as a GMQL
## datasets named "data1" and "data2", respectively, using CustomParser

init_gmql()
test_path <- system.file("example", "DATASET", package = "RGMQL")
test_path2 <- system.file("example", "DATASET_GDM", package = "RGMQL")
data1 <- read_gmql(test_path)
data2 <- read_gmql(test_path2)
```

```
## This statement creates a dataset called 'full' which contains all samples
## from the datasets 'data1' and 'data2'

res <- union(data1, data2)
```

`upload_dataset` *Upload dataset*

Description

It uploads a folder (GMQL or not) containing sample files using the proper GMQL web service available on a remote server: a new dataset is created on remote repository

Usage

```
upload_dataset(url, datasetName, folderPath, schemaName = NULL,
  isGMQL = TRUE)
```

Arguments

`url`	string url of server: It must contain the server address and base url; service name is added automatically
`datasetName`	name of dataset to create in repository
`folderPath`	string local path to the folder containing the samples files
`schemaName`	string name of schema used to parse the samples; schemaName available are: • NARROWPEAK • BROADPEAK • VCF • BED • BEDGRAPH if schemaName is NULL, it looks for a XML schema file to read in the folderPath
`isGMQL`	logical value indicating whether it is uploaded a GMQL dataset or not

Details

If no error occurs, it prints "Upload Complete", otherwise a specific error is printed

Value

None

Examples

```
## Not run:

## This statement defines the path to the folder "DATASET_GDM" in the
## subdirectory "example" of the package "RGMQL"

test_path <- system.file("example", "DATASET_GDM", package = "RGMQL")

## Login to GMQL REST services suite at remote url

remote_url <- "http://www.gmql.eu/gmql-rest/"
login_gmql(remote_url)

## Upload of GMQL dataset with "dataset1" as name, without specifying any
## schema

upload_dataset(remote_url, "dataset1", folderPath = test_path)


## End(Not run)
```

Index

aggregate, 3
aggregate,GMQLDataset-method (aggregate), 3
AGGREGATES-Object, 4
ALL, *11*
ALL (Cover-Param), 12
ANY, *11*
ANY (Cover-Param), 12
arrange, 6
arrange,GMQLDataset-method (arrange), 6
ASC, *7*
ASC (Ordering-Functions), 34
AVG, *10*, *20*, *24*, *30*
AVG (AGGREGATES-Object), 4

BAG, *10*, *20*, *24*, *30*
BAG (AGGREGATES-Object), 4
BAGD, *10*, *20*, *24*, *30*
BAGD (AGGREGATES-Object), 4

collect, 8
collect,GMQLDataset-method (collect), 8
compile_query, 9
compile_query_fromfile (compile_query), 9
condition_evaluation, *3*, *32*, *44*
condition_evaluation (Evaluation-Function), 17
conds, *11*, *24*, *30*, *45*
conds (Evaluation-Function), 17
COUNT, *10*, *20*, *24*, *30*
COUNT (AGGREGATES-Object), 4
COUNTSAMP, *24*
COUNTSAMP (AGGREGATES-Object), 4
cover, 10
cover,GMQLDataset-method (cover), 10
Cover-Param, 12

delete_dataset, 13
DESC, *7*
DESC (Ordering-Functions), 34
DG, *32*
DG (DISTAL-Object), 14
DGE, *32*
DGE (DISTAL-Object), 14
DISTAL-Object, 14
DL, *32*
DL (DISTAL-Object), 14
DLE, *32*
DLE (DISTAL-Object), 14
DOWN, *32*
DOWN (DISTAL-Object), 14
download_as_GRangesList (download_dataset), 16
download_dataset, 16

Evaluation-Function, 17
execute, 17
export_gmql, 18, *26*
extend, 19
extend,GMQLDataset-method (extend), 19
extend-method (extend), 19

filter, 20, *44*
filter,GMQLDataset-method (filter), 20
filter_and_extract, 22

group_by, 23
group_by,GMQLDataset-method (group_by), 23

import_gmql, *19*, 25
init_gmql, 26

log_job, 29
login_gmql, *27*, 27
logout_gmql, 28

map, 30
map,GMQLDataset-method (map), 30
map-method (map), 30
MAX, *10*, *20*, *24*, *30*
MAX (AGGREGATES-Object), 4
MD, *32*
MD (DISTAL-Object), 14
MEDIAN, *10*, *20*, *24*, *30*
MEDIAN (AGGREGATES-Object), 4
merge, 31

merge,GMQLDataset,GMQLDataset-method (merge), 31
META (OPERATOR-Object), 33
MIN, *10*, *20*, *24*, *30*
MIN (AGGREGATES-Object), 4

NIL (OPERATOR-Object), 33

OPERATOR-Object, 33
Ordering-Functions, 34

Q1, *10*, *20*, *24*, *30*
Q1 (AGGREGATES-Object), 4
Q2, *10*, *20*, *24*, *30*
Q2 (AGGREGATES-Object), 4
Q3, *10*, *20*, *24*, *30*
Q3 (AGGREGATES-Object), 4

read_gmql, 35
read_GRangesList (read_gmql), 35
register_gmql, 37
remote_processing, 38
run_query, 38
run_query_fromfile (run_query), 38

sample_metadata, 40
sample_region, 40
save_query, 41
save_query_fromfile (save_query), 41
select, 42
select,GMQLDataset-method (select), 42
semijoin, *21*, 44
setdiff, 45
setdiff,GMQLDataset,GMQLDataset-method (setdiff), 45
show_datasets_list, 46
show_job_log (log_job), 29
show_jobs_list, 47
show_queries_list, 48
show_samples_list, 48
show_schema, 49
SQRT (OPERATOR-Object), 33
STD, *10*, *20*, *24*, *30*
STD (AGGREGATES-Object), 4
stop_gmql, 50
stop_job, 51
SUM, *10*, *20*, *24*, *30*
SUM (AGGREGATES-Object), 4

take, 52
take,GMQLDataset-method (take), 52
take-method (take), 52
trace_job (log_job), 29

union, 53
union,GMQLDataset,GMQLDataset-method (union), 53
UP, *32*
UP (DISTAL-Object), 14
upload_dataset, 54