Genominator

April 19, 2010

`aggregateExpData` *Collapse data into unique entries*

Description

Collapses data based on unique combinations of values in a set of columns, by default adding a column giving counts of data entries with a particular combination.

Usage

```
aggregateExpData(expData, by = getIndexColumns(expData), tablename = NULL, delet
```

Arguments

`expData`	An object of class `ExpData`.
`by`	Vector containing column names used to define unique entries.
`tablename`	Name of database table to write output data to.
`deleteOriginal`	Logical indicating whether original database table in `ExpData` object should be deleted.
`overwrite`	Logical indicating whether database table referred to in `tablename` argument should be overwritten.
`verbose`	Logical indicating whether details should be printed.
`colname`	Name of column for recording aggregation output (by default, `counts`).
`aggregator`	SQLite code used for aggregating. See `Details` for more information.

Details

By default this function counts instances of data entries with a particular combination of the values in the set of columns indicated in the `by` argument. Other SQLite commands can be indicated using the `aggregator` argument.

Value

Returns an `ExpData` object.

Author(s)

James Bullard <bullard@stat.berkeley.edu>, Kasper Daniel Hansen <khansen@stat.berkeley>

See Also

See Genominator vignette for more information.

Examples

```
N  <- 10000 # the number of observations.
df <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                 location = sample(1:1000, size = N, replace = TRUE),
                 strand = sample(c(1L,-1L), size = N, replace = TRUE))
eDataRaw <- aggregateExpData(importToExpData(df, filename = tempfile(), tablename = "ex_t
```

---

applyMapped — *Apply a function over mapped data.*

Description

Apply a function over each element of a list containing data subsets, organized by annotation, with an additional argument for the annotation element associated with the list item.

Usage

```
applyMapped(mapped, annoData, FUN, bindAnno = FALSE)
```

Arguments

mapped	A list of data subsets, typically the return value of a call to splitByAnnotation. Names should correspond to names of annoData object.
annoData	A data frame which must contain the columns chr, start, end and strand which specifies annotation regions of interest.
FUN	A function of two arguments, the first being an element of mapped, the second being the corresponding element of annoData.
bindAnno	Logical indicating whether annotation information should be included in the output. If TRUE it assumes the output of FUN is conformable into a data.frame.

Value

If bindAnno is FALSE, returns a list containing the output of FUN for each element of the original mapped argument. If bindAnno is TRUE, returns a data frame, containing annotation information and output of FUN.

Author(s)

James Bullard <bullard@stat.berkeley.edu>, Kasper Daniel Hansen <khansen@stat.berkeley>

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
s <- splitByAnnotation(ed, yeastAnno[1:100,], what = getColnames(ed,
                       all = FALSE), ignoreStrand = TRUE, addOverStrand = TRUE)

## compute the per-base rate for this dataset.
applyMapped(s, yeastAnno, function(dta, anno) {
   colSums(dta, na.rm = TRUE)/(anno$end - anno$start + 1)
}, bindAnno = TRUE)[1:4,]
```

`collapseExpData` *Combine multiple data sets*

Description

This function takes a dataset with data from multiple experiments, and combines the data across multiple experiments according to a user-specified function.

Usage

```
collapseExpData(expData, tablename = NULL, what = getColnames(expData,
                all = FALSE), groups = "COL", collapse = c("sum", "avg",
                "weighted.avg"), overwrite = FALSE, deleteOriginal = FALSE,
                verbose = getOption("verbose"))
```

Arguments

`expData`	An object of class `ExpData`.
`tablename`	Name of database table to write output data to.
`what`	Data columns to apply `collapse` function to.
`groups`	Vector of length `what` indicating how columns should be grouped when applying `collapse` function.
`collapse`	Function to apply to grouped columns.
`overwrite`	Logical indicating whether database referred to in `tablename` argument should be overwritten.
`deleteOriginal`	Logical indicating whether original database in `ExpData` object should be deleted.
`verbose`	Logical indicating whether details should be printed.

Details

This function can be thought of as similar to `tapply`, operating over the entries in the data set, applying the function specified in the `collapse` argument, grouping the data as indicated in the `groups` argument.

Value

Returns an object of class `ExpData`.

Author(s)

James Bullard <bullard@stat.berkeley.edu>, Kasper Daniel Hansen <khansen@stat.berkeley>

See Also

See `Genominator` vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
nd <- importToExpData(head(ed, -1), filename = tempfile(), tablename = "collapsed")
cd <- collapseExpData(nd, tablename = "bio", groups = c("mut", "mut", "wt", "wt"), overwr
head(cd)
```

`computeCoverage` *Compute effort-coverage values*

Description

Compute fraction coverage obtained for a certain degree of sequencing effort.

Usage

```
computeCoverage(expData, annoData, cutoff = function(x, anno, group) x
> 10, effort = seq(1e+05, 5e+07, length = 20), smooth =
function(probs) probs, groups = rep("ALL", length(what)), what = getColnames(e
```

Arguments

`expData`	An `ExpData` object.
`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest.
`cutoff`	A predicate which determines when a region of annotation has been "sequenced". This function takes three arguments x = number of reads in region, anno = the annotation description of the region, group = the group it is in.
`effort`	Effort is a vector of how much sequencing has been done.
`smooth`	A function which takes as input the vector of probabilities and must return the probabilities.
`groups`	The different groups for which to calculate coverage.
`what`	The different columns, must be the same length as the groups.
`totals`	The lane totals, or some other totals. This allows us to estimate the sampling probability vector.
`ignoreStrand`	Whether or not to add over strands.
`verbose`	Do you want to see output.
`...`	Extra argument passed to cutoff.

Details

This argument is pretty general as different ways of specifying the arguments allows one to compute "coverage" under a lot of different definitions.

Value

Returns an object of class `genominator.coverage`. Pretty much you'll want to call plot on this object.

Author(s)

James Bullard <`bullard@stat.berkeley.edu`>, Kasper Daniel Hansen <`khansen@stat.berkeley`>

See Also

See the `plot.genominator.coverage` `Genominator` vignette for details.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
a <- computeCoverage(ed, yeastAnno, effort = 2^(5:18), cutoff = function(x, ...) x > 1, s
names(a)
```

`ExpData-class` *Class "ExpData"*

Description

A class for representing experimental data organized along a genome.

Objects from the Class

The preferred way to construct objects of class `ExpData` is to use the constructor function `ExpData(db = "filename.db", tablename = "tablename")`

Slots

`db`: Object of class `"character"` containing the filename of the SQLite database.

`tablename`: Object of class `"character"` containing the tablename of the relevant SQLite table.

`tableSchema`: Object of class `"character"`. The schema for the SQLite table.

`indexColumns`: Object of class `"character"`, listing which columns (and in which order) in the table has been indexed.

`mode`: Object of class `"character"`. Indicates whether the database is in read or write mode. Write mode implies read mode.

`chrMap`: Object of class `"character"`. For now, a placeholder.

`.tmpFile`: Object of class `"character"`. Only for developers..

`.pool`: Object of class `"environment"`. Only for developers.

Details

For all practical purposes, the class may be considered to point to a specific table in an SQLite database. A connection to the database is opened automatically and a pool of connections is maintained.

Methods

`ExpData(db, tablename, mode, indexColumns, pragma)` A constructor function. The last three arguments are for expert users.

`getDB` Returns a connection to the database associated with the ExpData object.

`getDBName` Returns the filename of the database associated with the ExpData object.

`getTablename` Returns the tablename of the ExpData object

`getSchema` Returns the schema of the table associated with the ExpData object.

`getIndexColumns` Returns the indexColumns of the object.

`getColnames` Returns all columns (argument `all = TRUE`) or all columns except the index-Columns (argument `all = FALSE`).

`listTables` Returns all vector of tables in a database.

`getMode` Returns the mode of the ExpData object.

[`signature(x = "ExpData")`: subsetting of the object. ExpData objects do not have row-names.

$ `signature(x = "ExpData")`: selects a column of the table.

head `signature(x = "ExpData")`: prints the first 10 rows of the object.

initialize `signature(.Object = "ExpData")`: The initialize method; use the constructor function `ExpData` instead.

regionGoodnessOfFit `signature(obj = "ExpData")`: FIXME

show `signature(object = "ExpData")`: the show method.

Author(s)

Jim Bullard <bullard@berkeley.edu> and Kasper Daniel Hansen <khansen@stat.berkeley.edu>

See Also

The package vignettes.

Examples

```
showClass("ExpData")
```

`Genominator-package` *Data backend for Genomic data*

Description

This package implements a data backend for genomic data, ie. data mapped to a genome with chromosome, location and possibly strand information. The data is stored in an SQLite database.

We are primarily using the package for analyzing mRNA-Seq data generated from a Solexa machine, but have also used it in part of a larger project incorporating Solexa data, tiling array data from various experiments and cDNA sequencing data.

It interfaces well with the GenomeGraphs package.

Read the package vignettes for extensive use cases.

Author(s)

James Bullard <bullard@stat.berkeley.edu>, Kasper Daniel Hansen <khansen@stat.berkeley>

`getRegion` *Select a region from an list("ExpData") object.*

Description

This function selects a subset of the data that falls into a particular contiguous genomic region.

Usage

```
getRegion(expData, chr, start, end, strand, what = "*", whereClause = "", verbos
```

Arguments

`expData`	An object of class `ExpData`.
`chr`	Chromosome number of desired region.
`start`	Start position of desired region. If omitted, it is set to 0.
`end`	End position of desired region. If omitted, it is set to 1e12.
`strand`	Strand of desired region. Values of 1 or -1 return data from forward or reverse strand. A value of 0 or a missing argument returns data from any strand, including data with missing strand information.
`what`	A vector of column names specifying which columns of the data should be returned. Defaults to all columns.
`whereClause`	Additional filtration criteria, customizable to refer to additional data columns. See Details for more explanation.
`verbose`	Logical indicating whether details should be printed.

Details

The argument `whereClause` should be a string indicating a subset of the data to be selected, using SQL syntax. For example, if you have a column called `category`, you could specify `category = 1` to select only those data entries where category has a value of 1. This function operates as a database query, and this argument can include logical combinations of multiple criteria.

Value

Returns a data frame containing the data from the desired region, with the desired columns.

Author(s)

James Bullard <`bullard@stat.berkeley.edu`>, Kasper Daniel Hansen <`khansen@stat.berkeley`>

See Also

See `Genominator` vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
c1 <- getRegion(ed, chr = 1)
dim(c1)
head(c1)
```

`importFromAlignedReads` *Import aligned reads to database*

Description

This function takes a named list of `AlignedRead` objects (from the **ShortRead** package) and creates an `ExpData` object from them, with one column for each list element. Column names are taken from list names, which must be unique.

Usage

```
importFromAlignedReads(alignedReads, chrMap, filename, tablename,
overwrite = TRUE, deleteIntermediates = TRUE, verbose = getOption("verbose"), ..
```

Arguments

`alignedReads`	A list of objects of class `AlignedRead`, where list elements have unique names.
`chrMap`	A vector of chromosome names from the aligned output. On importation to the database, chromosome names will be converted to integers corresponding to position within the `chrMap` vector.
`filename`	The filename of the database to which the data will be imported.
`tablename`	Name of database table to write output data to.
`overwrite`	Logical indicating whether database table referred to in `tablename` argument should be overwritten.

`deleteIntermediates`	Logical indicating whether intermediate database tables constructed in the process should be removed.
`verbose`	Logical indicating whether details should be printed.
`...`	Additional arguments to be passed to lower-level functions.

Value

Outputs an object of class `ExpData` with a column for each element of the `alignedRead` list.

Author(s)

James Bullard `<bullard@stat.berkeley.edu>`, Kasper Daniel Hansen `<khansen@stat.berkeley>`

See Also

See `Genominator` vignette for more information. See also `ExpData-class` and `AlignedRead-class`.

Examples

```
   ## Not run:
    require(ShortRead)
    require(yeastRNASeq)
    data("yeastAligned")
    eData <- importFromAlignedReads(yeastAligned, chrMap = levels(chromosome(yeastAlignec
                                     filename = tempfile(), tablename =
                                     "raw", overwrite = TRUE)

## End(Not run)
```

`importToExpData` *Import data to database*

Description

This function imports data from a data frame to a table in a database.

Usage

```
importToExpData(df, filename, tablename, overwrite = FALSE, verbose = getOption(
```

Arguments

`df`	A data frame containing data to be imported. Must have columns `chr`, `location` and `strand`.
`filename`	The filename of the database to which the data will be imported.
`tablename`	Name of database table to write output data to.
`overwrite`	Logical indicating whether database table referred to in `tablename` argument should be overwritten.
`verbose`	Logical indicating whether details should be printed.
`columns`	Vector of column names of columns to be imported.

Value

Returns an object of class `ExpData`.

Author(s)

James Bullard `<bullard@stat.berkeley.edu>`, Kasper Daniel Hansen `<khansen@stat.berkeley>`

See Also

See `Genominator` vignette for more information. See also `ExpData-class`.

Examples

```
N  <- 10000 # the number of observations.
df <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                 location = sample(1:1000, size = N, replace = TRUE),
                 strand = sample(c(1L,-1L), size = N, replace = TRUE))
eDataRaw <- importToExpData(df, filename = tempfile(), tablename = "ex_tbl", overwrite =
```

`joinExpData` *Merge ExpData objects*

Description

This function merges multiple `ExpData` object into one in an efficient manner.

Usage

```
joinExpData(expDataList, fields = NULL, tablename = "aggtable", overwrite = TRUE
```

Arguments

`expDataList`	List of `ExpData` objects. Must all be contained in the same database.
`fields`	A named list whose names correspond to tables of `ExpData` objects and whose entries indicate the column names to be pulled from each table.
`tablename`	Name of database table to write output data to.
`overwrite`	Logical indicating whether database table referred to in `tablename` argument should be overwritten.
`deleteOriginals`	Logical indicating whether original database tables in `ExpData` objects should be deleted.
`verbose`	Logical indicating whether details should be printed.

Value

An object of class `ExpData` containing data columns from all the original `ExpData` objects.

Author(s)

James Bullard `<bullard@stat.berkeley.edu>`, Kasper Daniel Hansen `<khansen@stat.berkeley>`

See Also

See `Genominator` vignette for more information.

Examples

```
N  <- 10000 # the number of observations.
df1 <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                  location = sample(1:1000, size = N, replace = TRUE),
                  strand = sample(c(1L,-1L), size = N, replace = TRUE))
df2 <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                  location = sample(1:1000, size = N, replace = TRUE),
                  strand = sample(c(1L,-1L), size = N, replace = TRUE))

 eDataRaw1 <- aggregateExpData(importToExpData(df1, filename = "my.db", tablename = "
 eDataRaw2 <- aggregateExpData(importToExpData(df1, filename = "my.db", tablename = "
 jd <- joinExpData(list(eDataRaw1, eDataRaw2), tablename = "combined",
                  fields = list("ex_tbl_1" = c("counts" = "e1"), "ex_tbl_2" = c("cour
 head(jd)
```

`mergeWithAnnotation` *Combine data with annotation*

Description

This function creates a data frame containing the data and the corresponding annotation information for each data row included in the annotation.

Usage

```
mergeWithAnnotation(expData, annoData, what = "*", ignoreStrand = FALSE, splitBy
```

Arguments

`expData`	An object of class `ExpData`.
`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest.
`what`	Which columns of `expData` to include.
`ignoreStrand`	Logical indicating whether strand should be ignored. If `TRUE`, data from either strand that falls into an annotation region is included.
`splitBy`	Field on which merged data frame should be split before returning.
`verbose`	Logical indicating whether details should be printed.

Details

Generally this function is good for creating a list of data split by some annotation feature, which can then be applied across.

Value

If `splitBy` is `NULL`, returns a data frame containing the data from `expData` that fall into regions defined by `annoData`, and which includes the annotation information, with columns as specified by `what`. If `splitBy` is non-`NULL`, returns a list of data frames with an element for each unique value of `splitBy` field.

Author(s)

James Bullard `<bullard@stat.berkeley.edu>`, Kasper Daniel Hansen `<khansen@stat.berkeley>`

See Also

See `Genominator` vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
mergeWithAnnotation(ed, yeastAnno[1:5,])
```

`plot.genominator.coverage`

Create coverage plot

Description

S3 method to plot `genominator.coverage` object. Shows coverage as a function of plotting effort.

Usage

```
plot.genominator.coverage(x, type = "l", col = NULL, draw.totals = TRUE, draw.le
```

Arguments

`x`	An object of class `genominator.coverage`, as returned by `computeCoverage`.
`type`	Plot type. See `plot`.
`col`	Vector of plotting colors.
`draw.totals`	Logical indicating whether totals should be drawn.
`draw.legend`	Logical indicating whether legend should be drawn.
`legend.location`	Vector giving x and y coordinates of legend position.
`...`	Additional arguments for lower-level functions.

Value

This method is used for its side effect.

Author(s)

James Bullard `<bullard@stat.berkeley.edu>`, Kasper Daniel Hansen `<khansen@stat.berkeley>`

See Also

See `Genominator` vignette for more information. See also `computeCoverage`.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
a <- computeCoverage(ed, yeastAnno, effort = 2^(5:18), cutoff = function(x, ...) x > 1)
plot(a, lwd = 5, col = "grey")
plot(a, draw.totals = FALSE)
b <- computeCoverage(ed, yeastAnno, groups = c("mut", "mut", "wt",
  "wt"), effort = 2^(5:18), cutoff = function(x, ...) x > 1)
plot(b)
b <- computeCoverage(ed, yeastAnno, groups = c("mut", "mut", "wt",
  "wt"), effort = 2^(5:18),
  cutoff = function(x, ...) x > 3, smooth = function(probs) { probs =
  probs + min(probs[probs!=0]); probs = probs/sum(probs)})
plot(b)
```

`plot.genominator.goodness.of.fit` *Create goodness-of-fit quantile-quantile plot*

Description

S3 method to plot `genominator.goodness.of.fit` object. Creates a quantile-quantile plot of the observed versus theoretical quantiles of goodness-of-fit statistics based on a chi-squared distribution.

Usage

```
plot.genominator.goodness.of.fit(x, chisq = FALSE, plotCol = TRUE,
                                    sample = FALSE, nsamples = 5000, xlab =
                                    "theoretical quantiles",
                                    ylab = "observed quantiles", main = names(x), p
```

Arguments

`x`	An object of class `genominator.goodness.of.fit`, as returned by `regionGoodnessOfFi`
`chisq`	Logical indicating whether chi-squared statistics should be plotted (as opposed to p-values from a chi-squared distribution).
`plotCol`	Logical indicating whether points at extreme quantiles should be colored.
`sample`	Logical indicating whether only a sample of statistics should be included in the plot. For large data sets, this may be useful for reducing the plot size.
`nsamples`	Integer indicating number of samples to be selected if `sample` is `TRUE`.
`xlab`	X-axis label for plot.
`ylab`	Y-axis label for plot.
`main`	Main label for plot.
`pch`	Plotting character type for plot.

`cex`	A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. See `par`.
`...`	Additional arguments for lower-level functions, namely `plot`.

Details

This function constructs a quantile-quantile plot comparing the distribution of observed statistics to either the uniform 0,1 distribution or the appropriate chi-squared distribution. This plotting function provides a tool to assess whether replicate lanes, flow cells, sample preparations, etc. fit the model described in `regionGoodnessOfFit`.

Value

This method is used for its side effect.

Author(s)

James Bullard <`bullard@stat.berkeley.edu`>, Kasper Daniel Hansen <`khansen@stat.berkeley`>

See Also

See `Genominator` vignette for more information. See also `regionGoodnessOfFit`.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
plot(regionGoodnessOfFit(ed, yeastAnno), chisq = TRUE)
```

`regionGoodnessOfFit-methods`
Calculate goodness-of-fit statistics

Description

A generic method for calculating chi-squared goodness-of-fit statistics (See details). Dispatches on either a `data.frame` or and `ExpData` object.

Usage

```
## S4 method for signature 'data.frame':
regionGoodnessOfFit(obj, denominator =
colSums(obj), groups = rep("A", ncol(obj)))

## S4 method for signature 'ExpData':
regionGoodnessOfFit(obj, annoData, groups = rep("A",
length(what)), what = getColnames(obj, all = FALSE), denominator =
c("regions", "lanes"), verbose = getOption("verbose"))
```

Arguments

`obj`	`data.frame` or `ExpData`
`annoData`	A data.frame of annotation.
`groups`	A factor or character vector describing which are the replicates.
`denominator`	How to scale the columns to take into account sequencing depth.
`what`	Which columns to choose from the database. Default is all data columns.
`verbose`	Whether or not debugging / timing info should be printed.

Details

This function implements the homogenous Poisson model across lanes as described in the article cited below. This model corresponds to common expression parameter across lanes scaled by a lane-specific offset. Goodness of fit to this model across replicates is a good indication of Poisson variation across lanes. Deviation from this is an indication of overdispersion between replicate lanes.

James H. Bullard, Elizabeth A. Purdom, Kasper D. Hansen, Steffen Durinck, and Sandrine Dudoit, "Statistical Inference in mRNA-Seq: Exploratory Data Analysis and Differential Expression" (April 2009). U.C. Berkeley Division of Biostatistics Working Paper Series.Working Paper 247.

Value

An list containing the statistics and degrees of freedom. See details. Technically, an S3 object with class genominator.goodness.of.fit

Methods

`signature(obj = "ExpData")` Here `obj` represents the results of a call to `summarizeByAnnotation` or a data.frame with columns representing samples and rows representing regions, i.e. genes. Denominator is how we scale each column, therefore it this must be true: `length(denominator) == ncol(obj)`. Finally, groups determines how columns are aggregated across one another, i.e. which columns are replicates.

`signature(obj = "data.frame")` Here `annoData` is an annotation data frame. `groups` is as above. `what` represents the columns to select choose. `denominator` is either the total lane counts, or the lane counts restricted to `annoData`, or a vector of length `length(groups)`

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
names(regionGoodnessOfFit(ed, yeastAnno))
```

`splitByAnnotation` *Split data into a list by annotation element.*

Description

This function splits the data into a list of matrices, by annotation element.

Usage

```
splitByAnnotation(expData, annoData, what = "*", ignoreStrand = FALSE, expand =
                    addOverStrands = FALSE, verbose = getOption("verbose"))
```

Arguments

`expData`	An object of class `ExpData`.
`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest.
`what`	Vector of names of columns of `expData` to be included in output.
`ignoreStrand`	Logical indicating whether strand should be ignored. If `TRUE`, data that falls into the annotation region, regardless of strand, is included.
`expand`	Logical indicating whether positions with no data should be included in output. If `TRUE`, lines are added to the output to give a value for each position, even if this value is 0.
`addOverStrands`	Logical indicating whether data should be added across strands. Only applies when `expand` is `TRUE`.
`verbose`	Logical indicating whether details should be printed.

Details

This function retrieves the data contained in the regions of the `annoData` object. The return object may be significant in size.

Value

Returns a list of length equal to the number of annotation entries split upon. Each list element is either a matrix of data, or a list with data matrices for each strand included (if `expand` is `TRUE` and `addOverStrands` is `FALSE`).

Author(s)

James Bullard <`bullard@stat.berkeley.edu`>, Kasper Daniel Hansen <`khansen@stat.berkeley`>

See Also

See `Genominator` vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
splitByAnnotation(ed, yeastAnno[1:30,])
```

`summarizeByAnnotation` *Summarize data based on genome annotation.*

Description

This function creates a summarization of columns of the data using specified SQLite functions, applying these summarization function to regions defined in an annotation data frame.

Usage

```
summarizeByAnnotation(expData, annoData, what = getColnames(expData, all = FALSE
```

Arguments

`expData`	An object of class `ExpData`.
`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest.
`what`	Vector of names of data columns to be summarized.
`fxs`	Vector of strings giving the names of SQLite functions to call on the data column(s).
`ignoreStrand`	Logical indicating whether strand should be taken into account in aggregation. If `TRUE` strand will be ignored.
`splitBy`	String indicating column of `annoData` object on which to split results.
`bindAnno`	Logical indicating whether annotation information should be included in the output.
`preserveColnames`	Logical indicating whether column names should be preserved. Only possible when a single function is being applied.
`verbose`	Logical indicating whether details should be printed.

Details

Most of the computation is done using SQLite. Depending on the use case, this approach may be significantly faster and use much less memory than the alternative: use `splitByAnnotation` to retrieve a list with all the data and then use R to summarize over each element of the list. It is (naturally) constrained to the use of operations expressible in (SQLite) SQL.

Value

If `splitBy` is not specified, returns a data frame containing results of aggregation functions performed on each region defined in `annoData`. If `splitBy` is specified, returns a list of data frames with one entry for each unique value of the column which was split on.

Author(s)

James Bullard `<bullard@stat.berkeley.edu>`, Kasper Daniel Hansen `<khansen@stat.berkeley>`

References

The SQLite website http://www.sqlite.org/lang_aggfunc.html has details on what mathematical functions are implemented.

See Also

See `Genominator` vignette for more information, as well as the `ExpData-class`.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
data("yeastAnno")
summarizeByAnnotation(ed, yeastAnno[1:50,])
```

`summarizeExpData` *Summarize a data column*

Description

This function returns a summary of one or more data columns, as indicated by a particular SQLite query function.

Usage

```
summarizeExpData(expData, what = getColnames(expData, all = FALSE), fxs = c("TOT
                 preserveColnames = TRUE, whereClause = "", verbose = getOption(
```

Arguments

`expData`	An object of class `ExpData`.
`what`	Vector of names of data columns to be summarized.
`fxs`	Vector of strings giving the names of SQLite functions to call on the data column.
`preserveColnames`	Logical indicating whether column names should be preserved.
`whereClause`	Additional filtration criteria, customizable to refer to additional data columns. See Details for more explanation.
`verbose`	Logical indicating whether details should be printed.

Details

The argument `whereClause` should be a string indicating a subset of the data to be selected. For example, if you have a column called `category`, you could specify `"category = 1"` to select only those data entries where category has a value of 1. This function operates as a database query, and thus the argument can include logical combinations of multiple criteria using SQL boolean operators.

Value

A vector with results of summarization.

Author(s)

James Bullard <bullard@stat.berkeley.edu>, Kasper Daniel Hansen <khansen@stat.berkeley>

References

The available SQLite functions are listed here: http://www.sqlite.org/lang_aggfunc.html

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"), tablename = "raw")
summarizeExpData(ed)
summarizeExpData(ed, fxs = c("MIN", "MAX", "AVG"))
```

Index