Package 'DuckDBGRanges'

August 24, 2026

Version 0.99.5

Date 2026-08-19

Title DuckDB-Backed GenomicRanges Implementation

Description Provides DuckDB-backed implementations of GenomicRanges data structures for efficient, out-of-memory operations on large genomic interval datasets. The package includes DuckDBGRanges and DuckDBGRangesList classes with optimized range operations including restrict, flank, shift, findOverlaps, coverage, and distance calculations. Ideal for variant analysis, scATAC-seq peak analysis, and other genomic workflows requiring memory-efficient range operations on large datasets.

License MIT + file LICENSE

URL https://github.com/Genentech/DuckDBGRanges

BugReports https://github.com/Genentech/DuckDBGRanges/issues

Depends R (>= 4.6.0), methods, stats, DuckDBDataFrame, GenomicRanges

Imports bit64, BiocGenerics, S4Vectors, IRanges, Seqinfo, DelayedArray, DBI, dplyr, dbplyr

Suggests knitr, rmarkdown, BiocStyle, testthat, microbenchmark, arrow, TxDb.Hsapiens.UCSC.hg38.knownGene

biocViews DataImport, DataRepresentation, Infrastructure, Sequencing, Software

VignetteBuilder knitr

Encoding UTF-8

Collate 'DuckDBGRanges-class.R' 'DuckDBGRanges-package.R' 'DuckDBGRanges-utils.R' 'DuckDBGRangesList-class.R' 'DuckDBGRangesList-utils.R'

Config/roxygen2/version 8.1.0

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

git_branch devel

git_last_commit 8b434a9

git_last_commit_date 2026-08-19

Repository Bioconductor 3.24

Date/Publication 2026-08-24

Author Patrick Aboyoun [aut, cre], Genentech, Inc. [cph]

Maintainer Patrick Aboyoun <aboyounp@gene.com>

Contents

`DuckDBGRanges-package` *DuckDBGRanges: DuckDB-Backed GenomicRanges Implementation*

Description

Provides DuckDB-backed implementations of GenomicRanges data structures for efficient, out-of-memory operations on large genomic interval datasets. The package includes DuckDBGRanges and DuckDBGRangesList classes with optimized range operations including restrict, flank, shift, findOverlaps, coverage, and distance calculations. Ideal for variant analysis, scATAC-seq peak analysis, and other genomic workflows requiring memory-efficient range operations on large datasets.

Author(s)

Maintainer: Patrick Aboyoun <aboyounp@gene.com>

Authors:

- Patrick Aboyoun <aboyounp@gene.com>

Other contributors:

- Genentech, Inc. [copyright holder]

See Also

Useful links:

- https://github.com/Genentech/DuckDBGRanges
- Report bugs at https://github.com/Genentech/DuckDBGRanges/issues

---

`DuckDBGRanges-class` *DuckDBGRanges objects*

Description

The DuckDBGRanges class extends the GenomicRanges virtual class for DuckDB tables.

Details

DuckDBGRanges adds GenomicRanges semantics to a DuckDB table. This includes the ability to define the sequence names, start, end, width, and strand columns, as well as the metadata columns. The `seqinfo` slot is used to define the sequence information for the ranges.

Value

The DuckDBGRanges() constructor returns a DuckDBGRanges object. Accessors return the requested component of the ranges (for example seqnames(), start(), end(), width(), strand(), ranges(), seqinfo(), length(), names(), and elementMetadata()); dbconn() and tblconn() return the backing DuckDB connection. Replacement methods (such as seqlengths<-, genome<-, and dimtbls<-) return the updated DuckDBGRanges. Coercion methods return the corresponding in-memory object (for example a GRanges), subsetting returns a DuckDBGRanges, and the show method returns NULL invisibly.

Constructor

DuckDBGRanges(conn, seqnames, start = NULL, end = NULL, width = NULL, strand = NULL, keycol = NULL, dimtbl

Creates a DuckDBGRanges object.

conn Either a character vector containing the paths to parquet, csv, or gzipped csv data files; a string that defines a duckdb read_* data source; a DuckDBDataFrame object; or a tbl_duckdb_connection object.

seqnames Either NULL or a string specifying the column from conn that defines the sequence names.

start Either NULL or a string specifying the column from conn that defines the start of the range.

end Either NULL or a string specifying the column from conn that defines the end of the range.

width Either NULL or a string specifying the column from conn that defines the width of the range.

strand Either NULL or a string specifying the column from conn that defines the width of the range.

keycol An optional string specifying the column name from conn that will define the foreign key in the underlying table, or a named list containing a character vector where the name of the list element defines the foreign key and the character vector set the distinct values for that key. If missing, a row_number column is created as an identifier.

dimtbl A optional named DataFrameList that specifies the dimension table associated with the keycol. The name of the list element must match the name of the keycol list. Additionally, the DataFrame object must have row names that match the distinct values of the keycol list element and columns that define partitions in the data table for efficient querying.

mcols Optional character vector specifying the columns that define the metadata columns.

seqinfo Either NULL, or a Seqinfo object, or a character vector of unique sequence names (a.k.a. seqlevels), or a named numeric vector of sequence lengths.

seqlengths Either NULL, or an integer vector named with levels(seqnames) and containing the lengths (or NA) for each level in levels(seqnames).

Accessors

In the code snippets below, x is a DuckDBGRanges object:

length(x): Get the number of elements.

names(x): Get the names of the elements.

seqnames(x): Get the sequence names.

ranges(x): Get the ranges as a DuckDBDataFrame.

start(x): Get the start values as a DuckDBColumn.

`end(x)`: Get the end values as a DuckDBColumn.

`width(x)`: Get the width values as a DuckDBColumn.

`strand(x)`: Get the strand values as a DuckDBColumn.

`mcols(x)`, `mcols(x) <- value`: Get or set the metadata columns.

`seqinfo(x)`: Get the information about the underlying sequences.

`seqlevels(x)`: Get the sequence levels; equivalent to `seqlevels(seqinfo(x))`.

`seqlengths(x)`: Get the sequence lengths; equivalent to `seqlengths(seqinfo(x))`.

`isCircular(x)`: Get or set the circularity flags.

`genome(x)`: Get the genome identifier or assembly name for each sequence; equivalent to `genome(seqinfo(x))`.

Coercion

`as(from, "DuckDBDataFrame")`: Creates a DuckDBDataFrame object.

`as.data.frame(x)`: Coerces `x` to a data.frame.

`as(from, "GRanges")`: Converts a DuckDBGRanges object to a GRanges object. This conversion begins by transforming the DuckDBGRanges into memory using `as.data.frame`. A GRanges object is then instantiated using the seqnames, start, width, and strand columns from the data.frame. Lastly, the seqinfo, metadata, and mcols (metadata columns) are copied over.

`realize(x, BACKEND = getAutoRealizationBackend())`: Realize an object into memory or on disk using the equivalent of `realize(as(x, "GRanges"), BACKEND)`.

Subsetting

In the code snippets below, `x` is a DuckDBGRanges object:

`x[i]`: Returns a DuckDBGRanges object containing the selected elements.

`head(x, n = 6L)`: If `n` is non-negative, returns the first `n` elements of `x`. If `n` is negative, returns all but the last `abs(n)` elements of `x`.

`tail(x, n = 6L)`: If `n` is non-negative, returns the last `n` elements of `x`. If `n` is negative, returns all but the first `abs(n)` elements of `x`.

Displaying

The `show()` method for DuckDBGRanges objects obeys global options `showHeadLines` and `showTailLines` for controlling the number of head and tail rows to display.

Author(s)

Patrick Aboyoun

See Also

- `DuckDBGRangesList-class` for the split GRanges class
- `GenomicRanges` for the base class

Examples

```
# Create an example data set with start and width columns:
df <- data.frame(id = head(letters, 10),
                 seqnames = rep.int(c("chr2", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)),
                 start = 1:10, width = 10:1,
                 strand = strand(rep.int(c("-", "+", "*", "+", "-"), c(1, 2, 2, 3, 2))),
                 score = 1:10,
                 GC = seq(1, 0, length = 10))
tf <- tempfile(fileext = ".parquet")
on.exit(unlink(tf))
arrow::write_parquet(df, tf)

# Create the DuckDBGRanges object
seqinfo <- Seqinfo(paste0("chr", 1:3), c(1000, 2000, 1500), NA, "mock1")
gr <- DuckDBGRanges(tf, seqnames = "seqnames", start = "start", width = "width",
                    strand = "strand", mcols = c("score", "GC"), seqinfo = seqinfo,
                    keycol = "id")
gr
```

`DuckDBGRanges-utils` *Utilities for DuckDBGRanges objects*

Description

Various utility methods for DuckDBGRanges objects including overlap detection, intra-range transformations, inter-range operations, and comparison methods. These methods translate R operations into efficient SQL queries executed directly in DuckDB.

Arguments

`query`, `x`	A GRanges or DuckDBGRanges object.
`subject`, `ranges`	A GRanges or DuckDBGRanges object.
`y`	For parallel set operations, a GRanges or DuckDBGRanges object of the same length as `x`.
`table`	For `match`, a DuckDBGRanges object to match against.
`maxgap`	A non-negative integer. Intervals with a separation of `maxgap` or less are considered to be overlapping. Default is -1L (no gap allowed).
`minoverlap`	A non-negative integer. The minimum number of base pairs that must be overlapping for two intervals to be considered overlapping. Default is 0L.
`type`	The type of overlap. Currently only `"any"` is supported for DuckDBGRanges.
`select`	When `"all"` (default), returns all overlapping pairs. When `"first"`, `"last"`, or `"arbitrary"`, returns a single match per query element.
`ignore.strand`	If `TRUE`, strand information is ignored when computing operations. Default is `FALSE`.
`invert`	For `subsetByOverlaps`, if `TRUE`, returns elements that do NOT overlap. Default is `FALSE`.
`shift`	For `shift`, the amount to shift ranges (can be negative).

width	For resize and flank, the target width.
fix	For resize, where to anchor: "start", "end", or "center".
start, end	For narrow, the new start/end positions relative to current ranges; a negative value counts back from the range end (-1 is the last base), matching base IRanges::narrow. For flank, if TRUE get upstream flanks.
both	For flank, if TRUE get flanks on both sides.
upstream, downstream	For promoters/terminators, distances.
use.names	If TRUE, preserve names in output.
fill.gap	For punion, if TRUE fill gaps between ranges.
drop.nohit.ranges	For pintersect, if TRUE drop non-intersecting pairs.
with.revmap	For range, if TRUE include reverse mapping.
na.rm, incomparables, fromLast, nomatch	Standard R arguments for comparison functions.
decreasing	For sort and order, if TRUE sort in decreasing order. Default is FALSE.
na.last	For order and rank, handling of NA values.
strictly	For is.unsorted, if TRUE check for strict ordering (no ties). Default is FALSE.
ties.method	For rank, method for handling ties. Only "first" and "min" are supported for DuckDBGRanges.
keep.all.ranges	For restrict, if TRUE keep ranges that become invalid after restriction. Default is FALSE.
method	For order, sorting method (ignored, included for compatibility).
...	Additional arguments passed to methods.

Details

The DuckDBGRanges methods translate genomic range operations into SQL queries, enabling efficient manipulation of large disk-backed genomic annotations without loading them into memory.

Value

Overlap methods return a Hits object (findOverlaps()), an integer vector (countOverlaps()), a logical vector (overlapsAny()), or a DuckDBGRanges (subsetByOverlaps()). Intra-range methods (shift(), narrow(), resize(), flank(), promoters(), terminators()) return a GRanges. Inter-range, set-operation, range-restriction, and tiling methods (such as range() and reduce()) return a DuckDBGRanges (or DuckDBGRangesList where the result is grouped). Nearest-neighbor, distance, comparison, and ordering methods return the corresponding integer or logical vectors (or a Hits).

Overlap Methods

In the code snippets below, query and subject can be GRanges or DuckDBGRanges objects, with at least one being a DuckDBGRanges:

`findOverlaps(query, subject, maxgap=-1L, minoverlap=0L, type=c("any", "start", "end", "within", "equ`
Find overlapping intervals between query and subject via SQL JOIN. Returns a Hits object.

`countOverlaps(query, subject, maxgap=-1L, minoverlap=0L, type=c("any", "start", "end", "within", "eq`
Count overlapping intervals for each query element. Returns an integer vector.

`overlapsAny(query, subject, maxgap=-1L, minoverlap=0L, type=c("any", "start", "end", "within", "equa`
Test if each query element overlaps any element in subject. Returns a logical vector.

`subsetByOverlaps(x, ranges, maxgap=-1L, minoverlap=0L, type=c("any", "start", "end", "within", "equa`
Subset x to elements that overlap ranges. When x is a DuckDBGRanges, returns a DuckDBGRanges object. When x is a GRanges, returns a GRanges object. Implemented using `overlapsAny` internally.

Intra-range Methods

Intra-range methods transform individual ranges. The `x` parameter is a DuckDBGRanges object, and these methods return a GRanges object:

`shift(x, shift=0L, use.names=TRUE)`**:** Shift all ranges by the specified amount. Returns a GRanges object.

`narrow(x, start=NA, end=NA, width=NA, use.names=TRUE)`**:** Narrow ranges by specifying new start/end/width relative to current ranges. Returns a GRanges object.

`resize(x, width, fix="start", use.names=TRUE, ignore.strand=FALSE)`**:** Resize ranges to specified width, anchored by fix position. Returns a GRanges object.

`flank(x, width, start=TRUE, both=FALSE, use.names=TRUE, ignore.strand=FALSE)`**:** Get flanking regions of specified width. Returns a GRanges object.

`promoters(x, upstream=2000, downstream=200, use.names=TRUE)`**:** Get promoter regions (upstream and downstream of TSS). Returns a GRanges object.

`terminators(x, upstream=2000, downstream=200, use.names=TRUE)`**:** Get terminator regions (upstream and downstream of TES). Returns a GRanges object.

Inter-range Methods

Inter-range methods operate across multiple ranges:

`range(x, ..., with.revmap=FALSE, ignore.strand=FALSE, na.rm=FALSE)`**:** Returns the range (min start to max end) per seqname/strand combination. Computed via SQL MIN/MAX aggregation. Returns a DuckDBGRanges object.

`reduce(x, drop.empty.ranges=FALSE, min.gapwidth=1L, with.revmap=FALSE, with.inframe.attrib=FALSE, `
Merge overlapping and adjacent ranges into single ranges per seqname/strand. Uses SQL window functions to identify merge groups. Returns a DuckDBGRanges object. Note: `with.revmap` and `with.inframe.attrib` are not supported.

`gaps(x, start=1L, end=seqlengths(x), ignore.strand=FALSE)`**:** Find gaps (uncovered regions) between ranges per seqname/strand. Uses `reduce()` internally then computes gaps between consecutive ranges. Returns a DuckDBGRanges object. Note: When `ignore.strand=FALSE`, only returns gaps for seqname/strand combinations present in the input data, unlike GRanges which returns gaps for all possible combinations. Use `ignore.strand=TRUE` for exact matching behavior.

`disjoin(x, with.revmap=FALSE, ignore.strand=FALSE)`**:** Break ranges into non-overlapping pieces at all breakpoints. Creates intervals between consecutive unique start/(end+1) positions. Returns a DuckDBGRanges object. Note: `with.revmap` is not supported.

`coverage(x, shift=0L, width=NULL, weight=1L, method=c("auto", "sort", "hash"), ...)`**:** Compute per-base coverage depth per seqname (strand is ignored, as in `GenomicRanges::coverage`). Uses the same delta-event and window-function `cumsum()` SQL as `disjoin()`/`gaps()`, collecting only the compact per-seqname breakpoint table before building a `SimpleRleList`.

shift and width are recycled per seqlevel; a range shifted so that it falls (partly or entirely) before position 1 is clipped to position 1, matching GenomicRanges::coverage rather than erroring. method is accepted for API compatibility and ignored. Note: weight must be a single number; a per-range vector or an mcols column name is not supported.

Set Operations

Set operations combine two DuckDBGRanges objects:

`union(x, y, ignore.strand=FALSE)`: Union of ranges from x and y. Combines ranges then reduces overlapping ones. Returns a DuckDBGRanges object.

`intersect(x, y, ignore.strand=FALSE)`: Intersection of ranges. Returns regions covered by both x and y. Uses SQL JOIN with overlap detection and GREATEST/LEAST for coordinates. Returns a DuckDBGRanges object.

`setdiff(x, y, ignore.strand=FALSE)`: Set difference. Returns regions in x that are not covered by y. Computes fragments by subtracting overlapping y ranges from x. Returns a DuckDBGRanges object.

Nearest Neighbor Methods

Methods for finding the nearest ranges in a subject:

`precede(x, subject, select=c("first", "all"), ignore.strand=FALSE)`: For each range in x, find the index of the first range in subject that it precedes (i.e., the first subject range starting after x ends). Overlapping ranges are excluded. Returns an integer vector with NAs for ranges with no qualifying match, or a Hits object if select="all".

`follow(x, subject, select=c("last", "all"), ignore.strand=FALSE)`: For each range in x, find the index of the last range in subject that x follows (i.e., the last subject range ending before x starts). Overlapping ranges are excluded. Returns an integer vector with NAs for ranges with no qualifying match, or a Hits object if select="all".

`nearest(x, subject, select=c("arbitrary", "all"), ignore.strand=FALSE)`: For each range in x, find the index of the nearest range in subject. Distance is 0 for overlapping ranges. For ties, one match is selected arbitrarily (minimum index). Returns an integer vector or Hits object. When subject is omitted (`nearest(x)`), each range's nearest neighbour is found among the *other* ranges of x – a range is never its own nearest neighbour – matching base GenomicRanges' drop.self.

`distanceToNearest(x, subject, ignore.strand=FALSE, select=c("arbitrary", "all"))`: Find the nearest range and compute the distance. Returns a Hits object with a "distance" metadata column containing the distances. As with nearest, the subject-omitted form excludes self-hits.

Comparison Methods

Methods for comparing DuckDBGRanges elements:

`duplicated(x, incomparables=FALSE, fromLast=FALSE)`: Find duplicate ranges using SQL window functions. Returns a DuckDBColumn.

`unique(x, incomparables=FALSE, fromLast=FALSE)`: Get unique ranges. Returns a DuckDBGRanges object.

`match(x, table, nomatch=NA_integer_, incomparables=NULL)`: Find matches between DuckDBGRanges objects using SQL JOIN. Returns a DuckDBColumn.

Parallel Set Operations

Parallel (element-wise) set operations between two DuckDBGRanges or between DuckDBGRanges and GRanges objects. When one argument is a GRanges, it is automatically converted to a DuckDBGRanges for optimized computation:

`punion(x, y, fill.gap=FALSE, ignore.strand=FALSE)`**:** Parallel union of ranges. Returns a DuckDBGRanges object.

`pintersect(x, y, drop.nohit.ranges=FALSE, ignore.strand=FALSE)`**:** Parallel intersection of ranges. Returns a DuckDBGRanges object.

`psetdiff(x, y, ignore.strand=FALSE)`**:** Parallel set difference of ranges. Returns a DuckDBGRanges object.

Distance Methods

Methods for computing distances between ranges:

`distance(x, y, ignore.strand=FALSE)`**:** Compute pairwise distances between ranges. Returns an integer vector.

Ordering Methods

Methods for ordering and sorting DuckDBGRanges objects:

`sort(x, decreasing=FALSE, ignore.strand=FALSE)`**:** Sort ranges by seqnames, strand, start, and width using SQL ORDER BY. Returns a DuckDBGRanges object.

`order(..., na.last=TRUE, decreasing=FALSE)`**:** Get the ordering permutation using SQL window functions. Returns an integer vector.

`is.unsorted(x, na.rm=FALSE, strictly=FALSE, ignore.strand=FALSE)`**:** Check if ranges are unsorted with respect to genomic order. Returns a logical scalar.

`rank(x, na.last=TRUE, ties.method=c("first", "min"), ignore.strand=FALSE)`**:** Get ranks of ranges using SQL window functions. Only `ties.method="first"` and `ties.method="min"` are supported for DuckDBGRanges (other methods like "average", "last", "random", "max" are not implemented). Returns an integer vector.

Range Restriction Methods

Methods for restricting ranges to bounds:

`trim(x, use.names=TRUE)`**:** Trim out-of-bound ranges on non-circular sequences to their seqlengths. Returns a DuckDBGRanges object.

`restrict(x, start=NA, end=NA, keep.all.ranges=FALSE, use.names=TRUE)`**:** Restrict ranges to specified start/end bounds using SQL GREATEST/LEAST. Returns a DuckDBGRanges object.

Tiling Methods

Methods for dividing ranges into sub-ranges:

`tile(x, n, width)`**:** Divide each range into tiles. Specify either 'n' (number of tiles per range) or 'width' (tile width). Returns a GRangesList with one element per input range. Note: materializes data for processing.

`slidingWindows(x, width, step=1L)`**:** Generate sliding windows of specified 'width' moving by 'step' positions. Returns a GRangesList. Note: materializes data for processing.

`pgap(x, y, ignore.strand=FALSE)`: Compute pairwise gaps between ranges in x and y, entirely in SQL. For each pair, returns the gap region between the two ranges, or a zero-width range at the boundary if they overlap or are adjacent. Errors if a pair has incompatible seqnames, or (unless `ignore.strand`) incompatible strand. Returns a DuckDBGRanges object.

Author(s)

Patrick Aboyoun

See Also

- `DuckDBGRanges-class` for the main class
- `findOverlaps` for the generic function
- `findOverlaps-methods` for GRanges methods

Examples

```
# Load required packages
library(GenomicRanges)

# Create example DuckDBGRanges
df <- data.frame(
    seqnames = c("chr1", "chr1", "chr2", "chr2"),
    start = c(100, 200, 150, 300),
    end = c(150, 250, 200, 350),
    strand = c("+", "-", "+", "-"),
    score = 1:4
)
tf <- tempfile(fileext = ".parquet")
arrow::write_parquet(df, tf)
subject <- DuckDBGRanges(tf, seqnames = "seqnames", start = "start",
                         end = "end", strand = "strand", mcols = "score")

# Create query GRanges
query <- GRanges(c("chr1:120-180", "chr2:100-400"))

# Find overlaps
hits <- findOverlaps(query, subject)

# Count overlaps
counts <- countOverlaps(query, subject)

# Test for any overlaps
any_ov <- overlapsAny(query, subject)

# Subset by overlaps (uses overlapsAny internally)
subset <- subsetByOverlaps(query, subject)
```

DuckDBGRangesList-class *DuckDBGRangesList objects*

Description

The DuckDBGRangesList class extends GRangesList to represent a DuckDB table with LIST[] columns as a GRangesList object.

Details

The DuckDBGRangesList class extends the GRangesList instead of GenomicRangesList class because the `rowRanges` slot accepts either a GenomicRanges object or a GRangesList object.

Value

The `DuckDBGRangesList()` constructor returns a DuckDBGRangesList object. Accessors return the requested component (for example `seqnames()`, `start()`, `end()`, `width()`, `strand()`, `length()`, `names()`, and `seqinfo()`); `dbconn()` and `tblconn()` return the backing DuckDB connection. Replacement methods (such as `names<-`, `seqlengths<-`, `genome<-`, and `dimtbls<-`) return the updated DuckDBGRangesList. Coercion methods return the corresponding in-memory object (for example a GRangesList), and subsetting returns a DuckDBGRangesList.

Constructor

`DuckDBGRangesList(conn, seqnames, start, width, strand = NULL, keycol = NULL, dimtbl = NULL, mcols = NU`

Creates a DuckDBGRangesList object from a data source with LIST[] columns.

`conn` Either a character vector containing the paths to parquet, csv, or gzipped csv data files; a string that defines a duckdb `read_*` data source; a DuckDBDataFrame object; or a tbl_duckdb_connection object.

`seqnames` A string specifying the column from `conn` that contains the LIST of sequence names for each element.

`start` A string specifying the column from `conn` that contains the LIST of start positions for each element.

`width` A string specifying the column from `conn` that contains the LIST of widths for each element.

`strand` Either `NULL` or a string specifying the column from `conn` that contains the LIST of strand values for each element.

`keycol` An optional string specifying the column name from `conn` that will define the foreign key in the underlying table, or a named list containing a character vector where the name of the list element defines the foreign key and the character vector set the distinct values for that key. If missing, a `row_number` column is created as an identifier.

`dimtbl` An optional named `DataFrameList` that specifies the dimension table associated with the `keycol`.

`mcols` Optional character vector specifying the columns that define the element-level metadata columns.

`seqinfo` Either `NULL`, or a `Seqinfo` object, or a character vector of unique sequence names (a.k.a. seqlevels), or a named numeric vector of sequence lengths.

`seqlengths` Either `NULL`, or an integer vector named with `levels(seqnames)` and containing the `lengths` (or `NA`) for each level in `levels(seqnames)`.

Accessors

In the code snippets below, x is a DuckDBGRangesList object:

`length(x)`: Get the number of elements in x.

`names(x)`, `names(x) <- value`: Get or set the names of the elements of x.

`seqnames(x)`: Get the sequence names as a DuckDBAtomicList.

`start(x)`: Get the start values as a DuckDBAtomicList.

`end(x)`: Get the end values as a DuckDBAtomicList.

`width(x)`: Get the width values as a DuckDBAtomicList.

`strand(x)`: Get the strand values as a DuckDBAtomicList.

`mcols(x)`, `mcols(x) <- value`: Get or set the element-level metadata columns.

`seqinfo(x)`: Get the information about the underlying sequences.

`elementNROWS(x)`: Get the number of ranges in each element.

Coercion

In the code snippets below, x is a DuckDBGRangesList object:

`as(from, "DuckDBDataFrame")`: Creates a DuckDBDataFrame object.

`as(from, "CompressedGRangesList")`: Converts a DuckDBGRangesList object to a CompressedGRangesList object. This conversion uses SQL UNNEST to flatten the LIST[] columns into a flat table, creates a GRanges object, then splits it by element to reconstruct the CompressedGRangesList.

`realize(x, BACKEND = getAutoRealizationBackend())`: Realize an object into memory or on disk using the equivalent of `realize(as(x, "CompressedGRangesList"), BACKEND)`.

Subsetting

In the code snippets below, x is a DuckDBGRangesList object:

`x[i]`: Returns a DuckDBGRangesList object containing the selected elements.

`x[[i]]`: Return the selected DuckDBGRanges by i, where i is an numeric or character vector of length 1.

`x$name`: Similar to `x[[name]]`, but name is taken literally as an element name.

`head(x, n = 6L)`: If n is non-negative, returns the first n elements of x. If n is negative, returns all but the last `abs(n)` elements of x.

`tail(x, n = 6L)`: If n is non-negative, returns the last n elements of x. If n is negative, returns all but the first `abs(n)` elements of x.

Author(s)

Patrick Aboyoun

See Also

- DuckDBGRanges-class for the single GRanges class
- GRangesList for the base class

Examples

```
# Create an example data set with LIST[] columns:
df <- data.frame(group = c("gr1", "gr2", "gr3", "gr4"),
                 seqnames = I(list("chr2", c("chr2", "chr2", "chr2"),
                                   c("chr1", "chr1"),
                                   c("chr3", "chr3", "chr3", "chr3"))),
                 start = I(list(1L, 2:4, 5:6, 7:10)),
                 width = I(list(10L, 9:7, 6:5, 4:1)),
                 strand = I(list("-", c("+", "+", "*"), c("+", "+"),
                                 c("+", "-", "-", "-"))),
                 score = c(1.0, 2.5, 5.5, 8.5),
                 GC = c(1.0, 0.7, 0.4, 0.15))
tf <- tempfile(fileext = ".parquet")
on.exit(unlink(tf))
arrow::write_parquet(df, tf)

# Create the DuckDBGRangesList object
seqinfo <- Seqinfo(paste0("chr", 1:3), c(1000, 2000, 1500), NA, "mock1")
grlist <- DuckDBGRangesList(tf, seqnames = "seqnames", start = "start",
                            width = "width", strand = "strand",
                            mcols = c("score", "GC"), seqinfo = seqinfo,
                            keycol = list(group = c("gr1", "gr2", "gr3", "gr4")))
grlist
```

DuckDBGRangesList-utils

Utilities for DuckDBGRangesList objects

Description

Various utility methods for DuckDBGRangesList objects including inter-range operations. These methods translate R operations into efficient SQL queries executed directly in DuckDB.

Arguments

x	A DuckDBGRangesList object.
with.revmap	If TRUE, include reverse mapping in output (not supported, will warn).
ignore.strand	If TRUE, strand information is ignored when computing operations. Default is FALSE.
na.rm	Ignored.
...	Additional arguments passed to methods.

Details

The DuckDBGRangesList methods translate genomic range operations into SQL queries, enabling efficient manipulation of large disk-backed genomic annotations without loading them into memory.

Value

range() returns a DuckDBGRangesList giving the range (minimum start to maximum end) spanned within each list element, per seqname/strand.

Inter-range Methods

Inter-range methods operate across multiple ranges within each list element:

`range(x, ..., with.revmap=FALSE, ignore.strand=FALSE, na.rm=FALSE)`: Returns the range (min start to max end) per list element and seqname/strand combination. Computed via SQL MIN/MAX aggregation with grouping by the partitioning column. Returns a DuckDBGRangesList object.

Author(s)

Patrick Aboyoun

See Also

- `DuckDBGRangesList-class` for the main class
- `DuckDBGRanges-utils` for DuckDBGRanges utilities
- `GRangesList-class` for the base class

Examples

```
# Create example DuckDBGRangesList
df <- data.frame(
    group = c("A", "B"),
    seqnames = I(list(
        rep(c("chr1", "chr2"), c(2, 3)),
        rep(c("chr1", "chr2"), c(3, 2))
    )),
    start = I(list(
        c(100L, 200L, 150L, 300L, 400L),
        c(50L, 100L, 200L, 250L, 300L)
    )),
    end = I(list(
        c(150L, 250L, 180L, 350L, 450L),
        c(80L, 150L, 250L, 300L, 350L)
    )),
    strand = I(list(
        rep(c("+", "-"), c(3, 2)),
        rep(c("+", "-"), c(2, 3))
    ))
)
tf <- tempfile(fileext = ".parquet")
arrow::write_parquet(df, tf)
grlist <- DuckDBGRangesList(tf, seqnames = "seqnames", start = "start",
                            end = "end", strand = "strand",
                            keycol = list(group = c("A", "B")))

# Get range per list element
range(grlist)
```

Index