Package 'dbSequence'

July 22, 2026

Title DuckDB-Backed Interface for Large-Scale Genomic Data

Version 0.99.1

Description Provides a DuckDB-backed infrastructure for working with large-scale genomic data that exceeds available memory. Supports lazy evaluation of genomic ranges and efficient overlap queries. Integrates with Bioconductor classes (GRanges, BiocIO) and provides plyranges-compatible API.

License MIT + file LICENSE

URL https://github.com/dbverse-org/dbSequence

BugReports https://github.com/dbverse-org/dbSequence/issues

biocViews Software, Infrastructure, DataImport, Sequencing, Coverage, DataRepresentation

BiocType Software

LazyData false

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.6.0)

Imports methods, duckdb, DBI, dbplyr, dplyr, glue, arrow, BiocIO, rtracklayer, Rsamtools, VariantAnnotation, dbProject, GenomicRanges, IRanges, rlang, cli, crayon

Suggests testthat (>= 3.0.0), SingleCellExperiment, scater, knitr, rmarkdown, BiocStyle, plyranges, microbenchmark

Config/testthat/edition 3

VignetteBuilder knitr

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

git_branch devel

git_last_commit 72038b1

git_last_commit_date 2026-06-30

Repository Bioconductor 3.24

Date/Publication 2026-07-22

Author Edward C. Ruiz [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9174-5387>),
Ruben Dries [rev],
National Human Genome Research Institute [fnd] (grant: 1R01HG014487-01)

Maintainer Edward C. Ruiz <ecr7407@gmail.com>

Contents

`dbSequence-package` *dbSequence: DuckDB-Backed Interface for Large-Scale Genomic Data*

Description

dbSequence provides DuckDB-backed infrastructure for genomic interval and sequence-derived data that are too large to work with comfortably in memory. It imports common genomics file formats into DuckDB, keeps range operations lazy, and interoperates with Bioconductor classes such as GenomicRanges::GRanges.

Author(s)

Maintainer: Edward C. Ruiz <ecr7407@gmail.com> (ORCID)

Other contributors:

- Ruben Dries [reviewer]
- National Human Genome Research Institute (1R01HG014487-01) [funder]

See Also

Useful links:

- https://github.com/dbverse-org/dbSequence
- Report bugs at https://github.com/dbverse-org/dbSequence/issues

---

`asRanges` *Extract genomic ranges from dbSequence objects*

Description

Extract genomic ranges from dbSequence objects

Usage

```
asRanges(x, ...)
```

Arguments

`x`	(required) A dbSequence object
`...`	(optional) Additional arguments

Value

A view of the dbSequence with only range columns (seqnames, start, end, strand)

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_path <- tempfile(fileext = ".duckdb")
db_seq <- read_bed(bed, dest = DuckDBFile(db_path), lazy = FALSE)
asRanges(db_seq)
```

as_granges *Convert objects to GRanges*

Description

S3 generic for converting various objects to GRanges. This generic is provided for plyranges compatibility when plyranges is not loaded.

S3 method for plyranges compatibility. Converts a dbSequence object to a GRanges object by collecting data from DuckDB.

Usage

```
as_granges(.data, ..., keep_mcols = TRUE)

## Default S3 method:
as_granges(.data, ..., keep_mcols = TRUE)

## S3 method for class 'dbSequence'
as_granges(.data, ..., keep_mcols = TRUE)
```

Arguments

.data	A dbSequence object
...	Additional arguments (currently unused)
keep_mcols	logical: whether to keep metadata columns (default: TRUE)

Details

This method enables plyranges-style workflows: `db_seq %>% as_granges()`

Note that this collects all data from DuckDB into memory. For large datasets, consider filtering first using dplyr verbs.

Value

A GRanges object

A GRanges object

Examples

```
# Import BED file to dbSequence
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)

# Convert to GRanges (collects data)
gr <- as_granges(db_seq)
```

as_ranges-dbSequence-method *Extract range columns as a view*

Description

Returns a view of the dbSequence with only the core range columns (seqnames, start, end, strand). This is useful for range operations that don't need metadata columns.

Usage

```
## S4 method for signature 'dbSequence'
asRanges(x, ...)
```

Arguments

x	(required) A dbSequence object
...	(optional) Additional arguments (currently unused)

Value

A dbSequence view with only range columns

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_path <- tempfile(fileext = ".duckdb")
db_seq <- read_bed(bed, dest = DuckDBFile(db_path), lazy = FALSE)
asRanges(db_seq)
```

coerce-methods *Coercion methods between dbSequence and GRanges*

Description

Methods for converting between dbSequence and GRanges objects.

`as(dbSequence, "GRanges")` materializes a dbSequence object into an in-memory GRanges object, triggering data collection from the DuckDB database.

`as(GRanges, "dbSequence")` loads a GRanges object into a DuckDB table and wraps it as dbSequence, enabling lazy evaluation of operations.

Arguments

from	Object to convert (dbSequence or GRanges)

Value

Converted object (GRanges or dbSequence respectively)

compute.dbSequence *Materialize a dbSequence in DuckDB*

Description

S3 method for `dplyr::compute()` that materializes the underlying lazy query into a DuckDB table, returning a new dbSequence pointing at the computed table.

Usage

```
compute.dbSequence(
  x,
  name = tableName(x),
  temporary = TRUE,
  overwrite = TRUE,
  ...
)
```

Arguments

`x`	A dbSequence object
`name`	Character table name to create. Defaults to `tableName(x)`.
`temporary`	Logical, create a temporary table (default: TRUE)
`overwrite`	Logical, overwrite existing table (default: TRUE)
`...`	Passed to `dplyr::compute()`.

Value

A dbSequence object backed by the materialized table.

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)
materialized <- compute.dbSequence(db_seq, name = "bed_materialized")
materialized
```

compute_coverage *Compute coverage over a region*

Description

Compute coverage (count of overlapping fragments) in bins across a genomic region. This is useful for visualization and summarizing fragment density.

Usage

```
compute_coverage(x, region, window = 100, ...)

## S3 method for class 'dbSequence'
compute_coverage(x, region, window = 100, ...)
```

Arguments

x	A dbSequence object
region	A GRanges object or string like "chr1:1000-2000"
window	Bin size in base pairs (default: 100)
...	Additional arguments

Value

A data frame with columns: bin_start, bin_end, count

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
fragments <- read_bed(bed)
coverage <- compute_coverage(fragments, "chr1:100-500", window = 100)
coverage
```

---

dbSequence *Constructor for dbSequence objects*

Description

Creates a dbSequence object by wrapping a table name. This is typically called internally by import methods.

Usage

```
dbSequence(table_name, file_source = NULL, .conn = NULL)
```

Arguments

table_name	(required) character: name of the table in the DuckDB database
file_source	(required) character or DuckDBFile: source file or database
.conn	(optional) DBIConnection: existing connection for in-memory databases (default: NULL)

Value

A dbSequence object

Examples

```
db_path <- tempfile(fileext = ".duckdb")
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = db_path)
DBI::dbWriteTable(con, "ranges", data.frame(seqnames = "chr1", start = 1, end = 10))
DBI::dbDisconnect(con, shutdown = TRUE)

db_seq <- dbSequence("ranges", DuckDBFile(db_path))
db_seq
```

---

dbSequence-class *dbSequence objects*

Description

S4 class for genomic sequences stored in DuckDB

Extends dbData to wrap genomic data stored in DuckDB tables.

Slots

`value` dplyr tbl representing the genomic data in the database (inherited from dbData)

`name` character table name in database (inherited from dbData)

`file_source` character source file path or identifier (immutable after creation)

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)
db_seq
```

DuckDBFile *Constructor for DuckDBFile objects*

Description

Creates a DuckDBFile object that wraps a file path for DuckDB databases. This is a simple path wrapper following the BiocFile pattern and supports both file paths and in-memory databases.

Usage

```
DuckDBFile(resource)
```

Arguments

`resource`	(required) character: path to DuckDB file. Can be a file path or ":memory:" for in-memory databases

Value

DuckDBFile object containing the path information

Examples

```
# Create DuckDBFile for existing database
db_file <- DuckDBFile(tempfile(fileext = ".duckdb"))

# Create DuckDBFile for in-memory database
mem_db <- DuckDBFile(":memory:")
```

`DuckDBFile-class` *DuckDBFile objects*

Description

File class for DuckDB database files, extending BiocFile

This mirrors the *File* classes in `rtracklayer`, giving us a concrete object to dispatch `import()` / `export()` methods **to**. Extends BiocFile to be compatible with BiocIO import/export framework.

Slots

`path` Character string specifying the path to the DuckDB database file. Can be a file path or ":memory:" for in-memory databases.

Examples

```
db_file <- DuckDBFile(tempfile(fileext = ".duckdb"))
db_file
```

`fileSource` *Accessor for file_source slot*

Description

This method prevents modification of the file_source slot after object creation. The file_source is set during import and should not be changed to maintain data integrity.

Usage

```
fileSource(object)

## S4 method for signature 'dbSequence'
fileSource(object)

fileSource(object) <- value

## S4 replacement method for signature 'dbSequence'
fileSource(object) <- value
```

Arguments

object	A dbSequence object
value	New value (will be rejected)

Value

The file_source value

Always errors because file sources are immutable after creation.

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)
fileSource(db_seq)
try(fileSource(db_seq) <- "other.bed")

bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)
try(fileSource(db_seq) <- "other.bed")
```

---

filter_by_overlaps *Filter ranges by overlap*

Description

Filter a dbSequence object to only include ranges that overlap with a set of query ranges. This operation is performed in DuckDB using efficient SQL filtering.

Usage

```
filter_by_overlaps(x, y, ...)

## S3 method for class 'dbSequence'
filter_by_overlaps(x, y, ...)
```

Arguments

x	A dbSequence object
y	A GRanges object or dbSequence object representing query ranges
...	Additional arguments (currently unused)

Details

Two intervals overlap if: start1 <= end2 AND start2 <= end1 AND chr1 == chr2

The operation is pushed down to DuckDB, so only matching rows are retrieved.

Value

A dbSequence object filtered to overlapping ranges

Examples

```
# Filter fragments to a specific region
bed <- system.file("extdata", "example.bed", package = "dbSequence")
fragments <- read_bed(bed)
region <- GenomicRanges::GRanges("chr1:100-500")
filtered <- filter_by_overlaps(fragments, region)
filtered
```

---

import,BamFile,missing,ANY-method

Import BAM files into DuckDB

Description

Import BAM files into DuckDB

Usage

```
## S4 method for signature 'BamFile,missing,ANY'
import(con, format, text, dest, table_name = "bam_data", .conn = NULL, ...)
```

Arguments

con	A file connection object.
format	Import format; unused for these methods.
text	Text input; unused for these methods.
dest	A DuckDBFile destination.
table_name	Table name to create.
.conn	Optional existing DBI connection.
...	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

---

import,BEDFile,missing,ANY-method

Import BED files into DuckDB

Description

Import BED files into DuckDB

Usage

```
## S4 method for signature 'BEDFile,missing,ANY'
import(con, format, text, dest, table_name = "bed_data", ...)
```

Arguments

con	A file connection object.
format	Import format; unused for these methods.
text	Text input; unused for these methods.
dest	A DuckDBFile destination.
table_name	Table name to create.
...	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

---

import,character,missing,ANY-method

Import character file paths into DuckDB (auto-detect format)

Description

Import character file paths into DuckDB (auto-detect format)

Usage

```
## S4 method for signature 'character,missing,ANY'
import(con, format, text, dest, table_name = NULL, ...)
```

Arguments

con	A file path.
format	Import format; unused for these methods.
text	Text input; unused for these methods.
dest	A DuckDBFile destination.
table_name	Table name to create. If NULL, the file extension is used.
...	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

import,FastaFile,missing,ANY-method

Import FASTA files into DuckDB

Description

Import FASTA files into DuckDB

Usage

```
## S4 method for signature 'FastaFile,missing,ANY'
import(con, format, text, dest, table_name = "fasta_data", ...)
```

Arguments

con	A file connection object.
format	Import format; unused for these methods.
text	Text input; unused for these methods.
dest	A DuckDBFile destination.
table_name	Table name to create.
...	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

import,GFFFile,missing,ANY-method

Import GFF files into DuckDB

Description

Import GFF files into DuckDB

Usage

```
## S4 method for signature 'GFFFile,missing,ANY'
import(con, format, text, dest, table_name = "gff_data", ...)
```

Arguments

con	A file connection object.
format	Import format; unused for these methods.
text	Text input; unused for these methods.
dest	A DuckDBFile destination.
table_name	Table name to create.
...	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

`import,GTFFile,missing,ANY-method`
Import GTF files into DuckDB

Description

Import GTF files into DuckDB

Usage

```
## S4 method for signature 'GTFFile,missing,ANY'
import(con, format, text, dest, table_name = "gtf_data", ...)
```

Arguments

`con`	A file connection object.
`format`	Import format; unused for these methods.
`text`	Text input; unused for these methods.
`dest`	A `DuckDBFile` destination.
`table_name`	Table name to create.
`...`	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

`import,VcfFile,missing,ANY-method`
Import VCF files into DuckDB

Description

Import VCF files into DuckDB

Usage

```
## S4 method for signature 'VcfFile,missing,ANY'
import(con, format, text, dest, table_name = "vcf_data", ...)
```

Arguments

con	A file connection object.
format	Import format; unused for these methods.
text	Text input; unused for these methods.
dest	A DuckDBFile destination.
table_name	Table name to create.
...	Additional arguments.

Value

A dbSequence object backed by a DuckDB table.

---

pool *Pool (aggregate) values by grouping columns*

Description

Aggregates values in a lazy database table by grouping columns. This is the database-native equivalent of dplyr::group_by() + summarise() but with a simpler interface optimized for common pooling operations.

Usage

```
pool(
  x,
  group_by,
  value_col = NULL,
  fun = "sum",
  name = NULL,
  filter_zero = TRUE,
  temporary = TRUE,
  overwrite = TRUE,
  ...
)

## Default S3 method:
pool(
  x,
  group_by,
  value_col = NULL,
  fun = "sum",
  name = NULL,
  filter_zero = TRUE,
  temporary = TRUE,
  overwrite = TRUE,
  ...
)

## S3 method for class 'tbl_duckdb_connection'
```

```
pool(
  x,
  group_by,
  value_col = "x",
  fun = "sum",
  name = NULL,
  filter_zero = TRUE,
  temporary = TRUE,
  overwrite = TRUE,
  ...
)

## S3 method for class 'tbl_dbi'
pool(
  x,
  group_by,
  value_col = "x",
  fun = "sum",
  name = NULL,
  filter_zero = TRUE,
  temporary = TRUE,
  overwrite = TRUE,
  ...
)

## S3 method for class 'dbSequence'
pool(
  x,
  group_by,
  value_col = "score",
  fun = "sum",
  name = NULL,
  filter_zero = TRUE,
  temporary = TRUE,
  overwrite = TRUE,
  ...
)
```

Arguments

x	A dbSequence object, tbl_duckdb_connection, or other lazy table
group_by	Character vector of column names to group by
value_col	Character, the column to aggregate. Default is "score" for dbSequence, "x" for tbl objects.
fun	Character, aggregation function: "sum" (default), "mean", "count", "min", "max"
name	Character, optional name for the resulting computed table. If NULL, returns a lazy query without materializing.
filter_zero	Logical, if TRUE (default) filter out rows where aggregated value == 0
temporary	Logical, if TRUE (default) create a temporary table, otherwise create a permanent table
overwrite	Logical, if TRUE (default) overwrite existing table

...	Additional arguments passed to dplyr::compute()

Value

Same type as input (lazy, still in database). For dbSequence, returns dbSequence if range columns are preserved, otherwise returns tbl.

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)

# Pool feature scores by feature name
pooled <- pool(db_seq, group_by = "name", value_col = "score")
pooled
```

---

read_bam *Read a BAM file into DuckDB*

Description

plyranges-style function to read BAM files. Returns a dbSequence object backed by DuckDB.

Usage

```
read_bam(file, dest = DuckDBFile(":memory:"), table_name = "alignments", ...)
```

Arguments

file	Path to BAM file
dest	DuckDBFile or path: destination database (default: in-memory)
table_name	character: name for the table (default: "alignments")
...	Additional arguments passed to import()

Value

A dbSequence object

Note

Unlike plyranges::read_bam which returns a DeferredGenomicRanges, this function immediately imports the BAM data into DuckDB.

See Also

import, BamFile

Examples

```
if (nzchar(system.file(package = "exonr"))) {
  bam <- system.file("extdata", "example.bam", package = "dbSequence")
  db_seq <- read_bam(bam, dest = DuckDBFile(tempfile(fileext = ".duckdb")))
  db_seq
}
```

`read_bed` *Read a BED file into DuckDB*

Description

plyranges-style function to read BED files. Unlike the plyranges version which returns GRanges, this returns a dbSequence object backed by DuckDB for lazy evaluation.

Usage

```
read_bed(
  file,
  dest = DuckDBFile(":memory:"),
  table_name = "bed_data",
  lazy = TRUE,
  ...
)
```

Arguments

`file`	Path to BED file
`dest`	DuckDBFile or path: destination database (default: in-memory)
`table_name`	character: name for the table (default: "bed_data")
`lazy`	logical: if TRUE (default), do not import/copy the file into a DuckDB table. Instead, return a dbSequence backed by a DuckDB scan query (an inline SQL SELECT over the file). Call `dplyr::compute()` on the result to materialize when needed.
`...`	Additional arguments passed to import()

Value

A dbSequence object

See Also

`import`, `BEDFile`

Examples

```
# Read BED file (lazy, stays in DuckDB)
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)

# Collect to GRanges when needed
gr <- as_granges(db_seq)
```

`read_gff` *Read a GFF/GTF file into DuckDB*

Description

plyranges-style function to read GFF files. Returns a dbSequence object backed by DuckDB.

Usage

```
read_gff(
  file,
  dest = DuckDBFile(":memory:"),
  table_name = "gff_data",
  lazy = TRUE,
  ...
)

read_gff3(
  file,
  dest = DuckDBFile(":memory:"),
  table_name = "gff_data",
  lazy = TRUE,
  ...
)
```

Arguments

`file`	Path to GFF file
`dest`	DuckDBFile or path: destination database (default: in-memory)
`table_name`	character: name for the table (default: "gff_data")
`lazy`	logical: if TRUE (default), do not import/copy the file into a DuckDB table. Instead, return a dbSequence backed by a DuckDB scan query (an inline SQL SELECT over the file). Call `dplyr::compute()` on the result to materialize when needed.
`...`	Additional arguments passed to import()

Value

A dbSequence object

See Also

import, GFFFile

Examples

```
gff <- system.file("extdata", "example.gff3", package = "dbSequence")
db_seq <- read_gff(gff)
db_seq
```

`read_vcf` *Read a VCF file into DuckDB*

Description

Read VCF variant files into DuckDB. Returns a dbSequence object.

Usage

```
read_vcf(
  file,
  dest = DuckDBFile(":memory:"),
  table_name = "variants",
  lazy = TRUE,
  ...
)
```

Arguments

`file`	Path to VCF file
`dest`	DuckDBFile or path: destination database (default: in-memory)
`table_name`	character: name for the table (default: "variants")
`lazy`	logical: if TRUE (default), do not import/copy the file into a DuckDB table. Instead, return a dbSequence backed by a DuckDB scan query (an inline SQL SELECT over the file). Call `dplyr::compute()` on the result to materialize when needed.
`...`	Additional arguments passed to import()

Value

A dbSequence object

See Also

import, VcfFile

Examples

```
vcf <- system.file("extdata", "example.vcf", package = "dbSequence")
db_seq <- read_vcf(vcf)
db_seq
```

show,dbSequence-method *Show method for dbSequence objects*

Description

Display a summary of the dbSequence object and preview the table data

Usage

```
## S4 method for signature 'dbSequence'
show(object)
```

Arguments

object	A dbSequence object

Value

Invisibly returns NULL.

show,DuckDBFile-method *Show method for DuckDBFile objects*

Description

Display information about the DuckDBFile object.

Usage

```
## S4 method for signature 'DuckDBFile'
show(object)
```

Arguments

object	A DuckDBFile object

Value

Invisibly returns NULL.

tableName *Get table name from dbSequence*

Description

Extract the table name from a dbSequence object.

Usage

```
tableName(x)

## S4 method for signature 'dbSequence'
tableName(x)
```

Arguments

x	(required) A dbSequence object

Value

character: the table name

Examples

```
bed <- system.file("extdata", "example.bed", package = "dbSequence")
db_seq <- read_bed(bed)
tableName(db_seq)
```

Index