Package ‘mdbplyr’

April 22, 2026

Title A Native Lazy Analytical Backend for MongoDB

Version 0.3.0

Description Provides a disciplined, lazy subset of 'dplyr' semantics for MongoDB aggregation pipelines. Queries remain lazy until collect() and compile into MongoDB-native aggregation stages.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.2.0)

Imports cli, dplyr, jsonlite, rlang, tibble

Suggests knitr, mongolite, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Paolo Bosetti [aut, cre]

Maintainer Paolo Bosetti <paolo.bosetti@unitn.it>

Repository CRAN

Date/Publication 2026-04-22 13:20:02 UTC

Contents

`append_stage` *Append a manual MongoDB aggregation stage*

Description

Appends a single raw MongoDB aggregation stage, provided as a JSON string, after the stages generated from the current lazy query.

Usage

```
append_stage(x, json_string)
```

Arguments

`x`	A `tbl_mongo` object.
`json_string`	A JSON string representing a single MongoDB pipeline stage, such as `{"$match":{"status":"paid"}}`

Details

This is an escape hatch for features not yet modeled by the package. The package does not attempt to infer schema changes introduced by manual stages.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

query <- append_stage(tbl, "{\"$limit\": 1}")
show_query(query)
```

arrange.tbl_mongo *Arrange a lazy Mongo query*

Description

Arrange a lazy Mongo query

Usage

```
arrange.tbl_mongo(.data, ..., .by_group = FALSE)
```

Arguments

.data	A tbl_mongo object.
...	Bare field names or desc(field).
.by_group	Whether to prefix the ordering with grouping fields.

Value

A modified tbl_mongo object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::arrange(tbl, dplyr::desc(amount))
```

collect *Collect a lazy Mongo query*

Description

Collect a lazy Mongo query

Usage

```
collect(x, ...)
```

Arguments

x	A tbl_mongo object.
...	Additional arguments forwarded to the executor.

Value

A tibble.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) {
    tibble::tibble(status = "paid", amount = 10)
  }
)

query <- dplyr::filter(tbl, amount > 0)
collect(query)
```

`compile_pipeline` *Compile a lazy Mongo query into an aggregation pipeline*

Description

Compile a lazy Mongo query into an aggregation pipeline

Usage

```
compile_pipeline(x)
```

Arguments

`x`	A `tbl_mongo` object.

Value

A list of MongoDB aggregation stages.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

query <- dplyr::filter(tbl, amount > 0)
compile_pipeline(query)
```

cursor *Open a lazy Mongo query as a mongolite cursor*

Description

Open a lazy Mongo query as a mongolite cursor

Usage

```
cursor(x, ...)
```

Arguments

x	A tbl_mongo object.
...	Additional arguments forwarded to the cursor executor.

Value

A mongolite iterator when backed by a live MongoDB collection, or a compatible cursor-like object supplied by the source.

Examples

```
collection <- list(
  name = "orders",
  aggregate = function(pipeline_json, iterate = FALSE, ...) {
    data <- tibble::tibble(status = "paid", amount = 10)
    if (!iterate) {
      return(data)
    }

    local({
      offset <- 1L
      page <- function(size = 1000) {
        if (offset > nrow(data)) {
          return(data[0, , drop = FALSE])
        }
        out <- data[offset:min(nrow(data), offset + size - 1L), , drop = FALSE]
        offset <<- offset + nrow(out)
        tibble::as_tibble(out)
      }
      structure(environment(), class = "mongo_iter")
    })
  }
)

tbl <- tbl_mongo(collection, schema = c("status", "amount"))
iter <- cursor(dplyr::filter(tbl, amount > 0))
iter$page()
```

`filter.tbl_mongo` *Filter a lazy Mongo query*

Description

Filter a lazy Mongo query

Usage

```
filter.tbl_mongo(.data, ..., .by = NULL, .preserve = FALSE)
```

Arguments

`.data`	A `tbl_mongo` object.
`...`	Predicate expressions.
`.by`	Unsupported.
`.preserve`	Included for dplyr compatibility.

Details

Predicate expressions use schema-first tidy evaluation:

- bare names refer to MongoDB fields when they are known fields of the lazy table
- otherwise, bare names are evaluated in the local R environment and inlined as literals
- `.data$...` always forces a MongoDB field reference
- `.env$...` always forces a local R value
- if a name exists both as a field and as a local variable, the field wins; use `.env$...` to force the local value

Bare field resolution depends on the known schema of `.data`. If a field, including a dotted path such as `` `message.measurements.Fx` ``, is missing from `schema_fields()`, write it explicitly as `.data$...` or supply the schema when creating the `tbl_mongo`.

The same resolution rules apply to expression arguments in `mutate.tbl_mongo()` and `summarise.tbl_mongo()`.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)
```

```
dplyr::filter(tbl, amount > 0)

threshold <- 10
dplyr::filter(tbl, amount > threshold)
dplyr::filter(tbl, .data$amount > .env$threshold)
```

`flatten_fields` *Flatten nested object fields into flat columns*

Description

Flatten nested object fields into flat columns

Usage

```
flatten_fields(.data, ..., names_fn = identity)
```

Arguments

`.data`	A `tbl_mongo` object.
`...`	Optional bare field roots or backticked dotted paths to flatten.
`names_fn`	Optional naming function applied to flattened output names.

Details

`flatten_fields()` relies on known schema fields. If nested dotted paths are not known yet, supply `schema = ...` when creating the table or call `infer_schema()` first.

With no field arguments, all known dotted paths are flattened. Existing already-flat columns are preserved. Arrays are treated as leaf values; use `unwind_array()` first if you need one row per array element.

Value

A modified `tbl_mongo` object.

`group_by.tbl_mongo` *Group a lazy Mongo query*

Description

Group a lazy Mongo query

Usage

```
group_by.tbl_mongo(
  .data,
  ...,
  .add = FALSE,
  .drop = dplyr::group_by_drop_default(.data)
)
```

Arguments

`.data`	A `tbl_mongo` object.
`...`	Bare field names.
`.add`	Whether to add to existing groups.
`.drop`	Included for dplyr compatibility.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::group_by(tbl, status)
```

infer_schema *Infer schema fields from the first source document*

Description

Infer schema fields from the first source document

Usage

```
infer_schema(x)
```

Arguments

x	A tbl_mongo object.

Details

`infer_schema()` inspects the first document of the source collection and flattens nested named subdocuments into dotted paths such as `"message.measurements.Fx"`. Arrays and other non-object values are treated as leaf fields. Because the schema comes from a single document, heterogeneous collections may still require manual schema adjustment.

Value

A tbl_mongo object with its source and IR schema updated from the first document in the underlying collection.

Examples

```
collection <- list(
  name = "orders",
  aggregate = function(pipeline_json, iterate = FALSE, ...) {
    tibble::tibble(
      status = "paid",
      message = list(list(amount = 10, currency = "EUR"))
    )
  }
)
tbl <- tbl_mongo(collection)

schema_fields(infer_schema(tbl))
```

mongo_src *Construct a MongoDB source wrapper*

Description

Construct a MongoDB source wrapper

Usage

```
mongo_src(
  collection,
  name = NULL,
  schema = NULL,
  executor = NULL,
  cursor_executor = NULL
)
```

Arguments

collection	A mongolite::mongo()-like object or a test double.
name	Optional human-readable collection name.
schema	Optional character vector describing the available fields.
executor	Optional function used to execute compiled pipelines.
cursor_executor	Optional function used to open a cursor over compiled pipelines.

Value

A mongo_src object.

Examples

```
collection <- list(
  name = "orders",
  aggregate = function(pipeline_json, iterate = FALSE, ...) {
    if (iterate) {
      data <- tibble::tibble(status = "paid", amount = 10)
      return(local({
        page <- function(size = 1000) data
        structure(environment(), class = "mongo_iter")
      }))
    }
    tibble::tibble(status = "paid", amount = 10)
  }
)

src <- mongo_src(collection, schema = c("status", "amount"))
src
```

mutate.tbl_mongo *Add computed fields to a lazy Mongo query*

Description

Add computed fields to a lazy Mongo query

Usage

```
mutate.tbl_mongo(.data, ...)
```

Arguments

.data	A tbl_mongo object.
...	Named scalar expressions.

Details

Expression arguments follow the same field-vs-local name resolution rules as `filter.tbl_mongo()`.

Value

A modified tbl_mongo object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::mutate(tbl, doubled = amount * 2)
```

rename.tbl_mongo *Rename fields in a lazy Mongo query*

Description

Rename fields in a lazy Mongo query

Usage

```
rename.tbl_mongo(.data, ...)
```

Arguments

.data	A tbl_mongo object.
...	Named bare field renames.

Value

A modified tbl_mongo object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::rename(tbl, total = amount)
```

---

schema_fields *Inspect known fields for a lazy Mongo query*

Description

Inspect known fields for a lazy Mongo query

Usage

```
schema_fields(x)
```

Arguments

x	A tbl_mongo or mongo_src object.

Value

A character vector of known field names.

Examples

```
src <- mongo_src(
  list(name = "orders", aggregate = function(...) tibble::tibble()),
  schema = c("status", "amount")
)
tbl <- tbl_mongo(src)

schema_fields(src)
schema_fields(tbl)
```

select.tbl_mongo *Select fields from a lazy Mongo query*

Description

Select fields from a lazy Mongo query

Usage

```
select.tbl_mongo(.data, ...)
```

Arguments

`.data`	A `tbl_mongo` object.
`...`	Bare field names or `new_name = old_name` renames.

Details

Selecting a dotted field path such as `` `message.measurements.Fx` `` does not flatten nested documents by default. The collected result preserves the native nested structure unless you explicitly rename the field in `select()` or call `flatten_fields()`.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::select(tbl, amount)
```

show_query *Show the MongoDB aggregation pipeline for a lazy query*

Description

Show the MongoDB aggregation pipeline for a lazy query

Usage

```
show_query(x, ...)
```

Arguments

x	A tbl_mongo object.
...	Unused.

Value

The pipeline JSON string, invisibly.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

query <- dplyr::select(tbl, amount)
show_query(query)
```

---

slice_head.tbl_mongo *Slice a lazy Mongo query*

Description

Slice a lazy Mongo query

Usage

```
slice_head.tbl_mongo(.data, ..., n = NULL, prop = NULL, by = NULL)

slice_tail.tbl_mongo(.data, ..., n = NULL, prop = NULL, by = NULL)

## S3 method for class 'tbl_mongo'
head(x, n = 6L, ...)
```

Arguments

.data	A tbl_mongo object.
...	Must be empty.
n	Number of rows to keep. Negative values drop rows from the opposite end, matching dplyr.
prop	Unsupported.
by	Unsupported.
x	A tbl_mongo object.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::slice_head(tbl, n = 2)
dplyr::slice_tail(tbl, n = 2)
head(tbl, 2)
```

---

summarise.tbl_mongo *Summarise a lazy Mongo query*

Description

Summarise a lazy Mongo query

Usage

```
summarise.tbl_mongo(.data, ..., .by = NULL, .groups = NULL)
```

Arguments

`.data`	A `tbl_mongo` object.
`...`	Named summary expressions.
`.by`	Unsupported.
`.groups`	Included for dplyr compatibility.

Details

Summary expressions follow the same field-vs-local name resolution rules as `filter.tbl_mongo()`.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

query <- tbl |>
  dplyr::group_by(status) |>
  dplyr::summarise(total = sum(amount))

show_query(query)
```

---

tbl_mongo *Create a lazy MongoDB table*

Description

Create a lazy MongoDB table

Usage

```
tbl_mongo(collection, name = NULL, schema = NULL, executor = NULL)
```

Arguments

collection	A mongo_src object or a mongolite::mongo()-like object.
name	Optional collection name when collection is not already a mongo_src.
schema	Optional character vector describing known fields.
executor	Optional executor function for compiled pipelines.

Details

Supplying schema = ... is the most reliable way to make field references explicit, especially for dotted paths in nested documents. If you do not want to write the schema manually, infer_schema() can populate it from the first document in the collection:

```
tbl_mongo(collection) |> infer_schema()
```

This is convenient, but it only reflects one document. If the collection is heterogeneous, fields that do not appear in the first document may still need to be added manually.

Value

A tbl_mongo object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("status", "amount"),
  executor = function(pipeline, ...) {
    tibble::tibble(status = "paid", amount = 10)
  }
)

tbl
```

`transmute.tbl_mongo` *Compute and keep only derived fields*

Description

Compute and keep only derived fields

Usage

```
transmute.tbl_mongo(.data, ...)
```

Arguments

`.data`	A `tbl_mongo` object.
`...`	Named scalar expressions.

Details

Expression arguments follow the same field-vs-local name resolution rules as `filter.tbl_mongo()`.

Value

A modified `tbl_mongo` object.

Examples

```
tbl <- tbl_mongo(
  list(name = "orders"),
  schema = c("amount"),
  executor = function(pipeline, ...) tibble::tibble()
)

dplyr::transmute(tbl, doubled = amount * 2)
```

unwind_array *Unwind one array field lazily*

Description

Unwind one array field lazily

Usage

```
unwind_array(.data, field, preserve_empty = FALSE)
```

Arguments

.data	A tbl_mongo object.
field	A single bare field name or backticked dotted path.
preserve_empty	Whether to preserve rows with missing or empty arrays.

Details

unwind_array() compiles to MongoDB $unwind and repeats each row once per array element, replacing the original array field with that element. Only one field can be unwound per call; chain multiple calls if needed.

Value

A modified tbl_mongo object.

Index