Package ‘duckplyr’

December 11, 2023

Type Package
Title A ‘DuckDB’-Backed Version of ‘dplyr’
Version 0.3.0
Description A drop-in replacement for ‘dplyr’, powered by 'DuckDB' for performance. Also defines a set of generics that provide a low-level implementer's interface for the high-level user interface of 'dplyr'.
License MIT + file LICENSE
BugReports https://github.com/duckdblabs/duckplyr/issues
Depends R (>= 4.1.0)
Imports cli, collections, DBI, dplyr (>= 1.1.4), duckdb (>= 0.9.1-1), glue, lifecycle, purrr, rlang (>= 1.0.6), tibble, tidyselect, utils, vctrs (>= 0.6.3)
Suggests arrow, brio, constructive (>= 0.2.0), dbplyr, hms, lobstr, lubridate, palmerpenguins, pillar, prettycode, qs, reprex, rstudioapi, styler, testthat (>= 3.1.5), withr
Config/testthat/edition 3
Config/testthat/parallel false
Config/testthat/start-first as_duckplyr_df, mutate, filter, count-tally
Encoding UTF-8
RoxygenNote 7.2.3
NeedsCompilation no
Author Hannes Mühleisen [aut] (<https://orcid.org/0000-0001-8552-0029>), Kirill Müller [aut, cre] (<https://orcid.org/0000-0002-1416-3412>), Posit Software, PBC [cph, fnd]
Maintainer Kirill Müller <kirill@cynkra.com>
Repository CRAN
Date/Publication 2023-12-11 07:40:10 UTC
**R topics documented:**

- `as_duckplyr_df` ................................................................. 2
- `config` ................................................................................. 3
- `df_from_file` ........................................................................ 4
- `is_duckplyr_df` ................................................................. 5
- `methods_overwrite` .......................................................... 6
- `new_relational` ................................................................. 7
- `new_relexpr` ...................................................................... 11
- `stats_show` ........................................................................ 12

**Index**

---

**as_duckplyr_df**  
Convert to a duckplyr data frame

---

**Description**

For an object of class `duckplyr_df`, `dplyr` verbs such as `mutate()`, `select()` or `filter()` will attempt to use DuckDB. If this is not possible, the original `dplyr` implementation is used.

**Usage**

```r
as_duckplyr_df(.data)
```

**Arguments**

- `.data`  
  data frame or tibble to transform

**Details**

Set the `DUCKPLYR_FALLBACK_INFO` and `DUCKPLYR_FORCE` environment variables for more control over the behavior, see `config` for more details.

**Value**

An object of class "duckplyr_df", inheriting from the classes of the `.data` argument.

**Examples**

```r
tibble(a = 1:3) %>%
  mutate(b = a + 1)
```

```r
tibble(a = 1:3) %>%
  as_duckplyr_df() %>%
  mutate(b = a + 1)
```
### Description

The behavior of duckplyr can be fine-tuned with several environment variables, and one option.

### Options

`duckdb.materialize_message`: Set to FALSE to turn off diagnostic output from duckdb on data frame materialization. Currently set to TRUE when duckplyr is loaded.

### Environment variables

- `DUCKPLYR_OUTPUT_ORDER`: If TRUE, row output order is preserved. The default may change the row order where dplyr would keep it stable.
- `DUCKPLYR_FORCE`: If TRUE, fail if duckdb cannot handle a request.
- `DUCKPLYR_FALLBACK_INFO`: If TRUE, print a message when a fallback to dplyr occurs because duckdb cannot handle a request.
- `DUCKPLYR_CHECK_ROUNDTRIP`: If TRUE, check if all columns are roundtripped perfectly when creating a relational object from a data frame. This is slow, and mostly useful for debugging. The default is to check roundtrip of attributes.
- `DUCKPLYR_EXPERIMENTAL`: If TRUE, pass experimental = TRUE to certain duckdb functions. Currently unused.

### Examples

```r
# options(duckdb.materialize_message = FALSE)
data.frame(a = 3:1) %>%
as_duckplyr_df() %>%
inner_join(data.frame(a = 1:4), by = "a")

rlang::with_options(duckdb.materialize_message = FALSE, {
data.frame(a = 3:1) %>%
as_duckplyr_df() %>%
inner_join(data.frame(a = 1:4), by = "a") %>%
print()
})

# Sys.setenv(DUCKPLYR_OUTPUT_ORDER = TRUE)
data.frame(a = 3:1) %>%
as_duckplyr_df() %>%
inner_join(data.frame(a = 1:4), by = "a")

withr::with_envvar(c(DUCKPLYR_OUTPUT_ORDER = "TRUE"), {
data.frame(a = 3:1) %>%
as_duckplyr_df() %>%
inner_join(data.frame(a = 1:4), by = "a")
})
```
df_from_file

Read Parquet, CSV, and other files using DuckDB

Description

This function ingests data from files. Internally, a DuckDB table-valued function is called, the results are transparently converted to a data frame. The data is only read when the data is actually accessed. See https://duckdb.org/docs/data/overview for a documentation of the available functions and their options.

duckplyr_df_from_file() is a thin wrapper around df_from_file() that calls as_duckplyr_df() on the output.

Usage

df_from_file(path, table_function, options = list(), class = NULL)

duckplyr_df_from_file(path, table_function, options = list(), class = NULL)

Arguments

path         Path to file or directory

table_function  The name of a table-valued DuckDB function such as "read_parquet", "read_csv", "read_csv_auto" or "read_json".

options  Arguments to the DuckDB function indicated by table_function.

class  An optional class to add to the data frame. The returned object will always be a data frame. Pass class(tibble()) to create a tibble.
is_duckplyr_df

Value

A data frame for df_from_file(), or a duckplyr_df for duckplyr_df_from_file(), extended by the provided class.

Examples

# Create simple CSV file
path <- tempfile(fileext = ".csv")
write.csv(data.frame(a = 1:3, b = letters[4:6]), path, row.names = FALSE)

# Reading is immediate
df <- df_from_file(path, "read_csv_auto")

# Materialization only upon access
names(df)
df$a

# Return as tibble:
df_from_file(
  path,
  "read_csv",
  options = list(delim = ",", auto_detect = TRUE),
  class = class(tibble())
)

unlink(path)

---

is_duckplyr_df

Class predicate for duckplyr data frames

Description

Tests if the input object is of class "duckplyr_df".

Usage

is_duckplyr_df(.data)

Arguments

.data The object to test

Value

TRUE if the input object is of class "duckplyr_df", otherwise FALSE.
### methods_overwrite

**Forward all dplyr methods to duckplyr**

#### Description

After calling `methods_overwrite()`, all dplyr methods are redirected to duckplyr for the duration of the session, or until a call to `methods_restore()`.

#### Usage

```r
methods_overwrite()
methods_restore()
```

#### Value

Called for their side effects.

#### Examples

```r
tibble(a = 1:3) %>%
  is_duckplyr_df()

tibble(a = 1:3) %>%
  as_duckplyr_df() %>%
  is_duckplyr_df()
```

```r
tibble(a = 1:3) %>%
  mutate(b = a + 1)
methods_overwrite()

tibble(a = 1:3) %>%
  mutate(b = a + 1)
methods_restore()

tibble(a = 1:3) %>%
  mutate(b = a + 1)```
new_relatio\nal

new relational Relational implementer’s interface

Description
The constructor and generics described here define a class that helps separating dplyr’s user interface from the actual underlying operations. In the longer term, this will help packages that implement the dplyr interface (such as dbplyr, dplyr, arrow and similar) to focus on the core details of their functionality, rather than on the intricacies of dplyr’s user interface.

new_relatio\nal() constructs an object of class “relational”. Users are encouraged to provide the class argument. The typical use case will be to create a wrapper function.

rel_to_df() extracts a data frame representation from a relational object, to be used by dplyr::collect().
rel_filter() keeps rows that match a predicate, to be used by dplyr::filter().
rel_project() selects columns or creates new columns, to be used by dplyr::select(), dplyr::rename(), dplyr::mutate(), dplyr::relocate(), and others.
rel_aggregate() combines several rows into one, to be used by dplyr::summarize().
rel_order() reorders rows by columns or expressions, to be used by dplyr::arrange().
rel_join() joins or merges two tables, to be used by dplyr::left_join(), dplyr::right_join(), dplyr::inner_join(), dplyr::full_join(), dplyr::cross_join(), dplyr::semi_join(), and dplyr::anti_join().
rel_limit() limits the number of rows in a table, to be used by utils::head().
rel_distinct() only keeps the distinct rows in a table, to be used by dplyr::distinct().
rel_set_intersect() returns rows present in both tables, to be used by intersect().
rel_set_diff() returns rows present in any of both tables, to be used by setdiff().
rel_set_symdiff() returns rows present in any of both tables, to be used by dplyr::symdiff().
rel_union_all() returns rows present in any of both tables, to be used by dplyr::union_all().
rel_explain() prints an explanation of the plan executed by the relational object.
rel_alias() returns the alias name for a relational object.
rel_set_alias() sets the alias name for a relational object.
rel_names() returns the column names as character vector, to be used by colnames().

Usage
new_relatio\nal(..., class = NULL)
rel_to_df(rel, ...)
rel_filter(rel, exprs, ...)
rel_project(rel, exprs, ...)
new_rendonic

rel_aggregate(rel, groups, aggregates, ...)
rel_order(rel, orders, ...)

rel_join(
  left,
  right,
  conds,
  join = c("inner", "left", "right", "outer", "cross", "semi", "anti"),
  join_ref_type = c("regular", "natural", "cross", "positional", "asof"),
  ...
)
rel_limit(rel, n, ...)
rel_distinct(rel, ...)
rel_set_intersect(rel_a, rel_b, ...)
rel_set_diff(rel_a, rel_b, ...)
rel_set_symdiff(rel_a, rel_b, ...)
rel_union_all(rel_a, rel_b, ...)
rel_explain(rel, ...)
rel_alias(rel, ...)
rel_set_alias(rel, alias, ...)
rel_names(rel, ...)

Arguments
... Reserved for future extensions, must be empty.
class Classes added in front of the "relational" base class.
rel, rel_a, rel_b, left, right
  A relational object.
exprs A list of expr objects to filter by.
groups A list of expressions to group by.
aggregates A list of expressions with aggregates to compute.
orders A list of expressions to order by.
conds A list of expressions to use for the join.
join The type of join.
join_ref_type The ref type of join.
The number of rows.

the new alias

Value

- new_reational() returns a new relational object.
- rel_to_df() returns a data frame.
- rel_names() returns a character vector.
- All other generics return a modified relational object.

Examples

```r
new_dfrel <- function(x) {
  stopifnot(is.data.frame(x))
  new_relational(list(x), class = "dfrel")
}
mtcars_rel <- new_dfrel(mtcars[1:5, 1:4])

rel_to_df.dfrel <- function(rel, ...) {
  unclass(rel)[[1]]
}
rel_to_df(mtcars_rel)

rel_filter.dfrel <- function(rel, exprs, ...) {
  df <- unclass(rel)[[1]]
  # A real implementation would evaluate the predicates defined
  # by the exprs argument
  new_dfrel(df[seq_len(min(3, nrow(df))), ])
}
rel_filter(
  mtcars_rel,
  list(
    relexpr_function(  
      "gt",
      list(relexpr_reference("cyl"), relexpr_constant("6"))
    )
  )
)

rel_project.dfrel <- function(rel, exprs, ...) {
  df <- unclass(rel)[[1]]
  # A real implementation would evaluate the expressions defined
  # by the exprs argument
  new_dfrel(df[seq_len(min(3, ncol(df))))])
}
rel_project(
  mtcars_rel,
  list(  
    relexpr_function(  
      "gt",
      list(relexpr_reference("cyl"), relexpr_constant("6"))
    )
  )
)
```
list(relexpr_reference("cyl"), relexpr_reference("disp"))
)

rel_order.dfrel <- function(rel, exprs, ...) {
  df <- unclass(rel)[[1]]

  # A real implementation would evaluate the expressions defined
  # by the exprs argument
  new_dfrel(df[order(df[[1]]), ])
}

rel_order(
  mtcars_rel,
  list(relexpr_reference("mpg"))
)

rel_join.dfrel <- function(left, right, conds, join, ...) {
  left_df <- unclass(left)[[1]]
  right_df <- unclass(right)[[1]]

  # A real implementation would evaluate the expressions
  # defined by the conds argument,
  # use different join types based on the join argument,
  # and implement the join itself instead of relaying to left_join().
  new_dfrel(dplyr::left_join(left_df, right_df))
}

rel_join(new_dfrel(data.frame(mpg = 21)), mtcars_rel)

rel_limit.dfrel <- function(rel, n, ...) {
  df <- unclass(rel)[[1]]
  new_dfrel(df[seq_len(n), ])
}

rel_limit(mtcars_rel, 3)

rel_distinct.dfrel <- function(rel, ...) {
  df <- unclass(rel)[[1]]
  new_dfrel(df[!duplicated(df), ])
}

rel_distinct(new_dfrel(mtcars[1:3, 1:4]))

rel_names.dfrel <- function(rel, ...) {
  df <- unclass(rel)[[1]]
  names(df)
}

rel_names(mtcars_rel)
new_relexpr

---

### Description

These functions provide a backend-agnostic way to construct expression trees built of column references, constants, and functions. All subexpressions in an expression tree can have an alias.

- `new_relexpr()` constructs an object of class "relational_relexpr". It is used by the higher-level constructors, users should rarely need to call it directly.
- `relexpr_reference()` constructs a reference to a column.
- `relexpr_constant()` wraps a constant value.
- `relexpr_function()` applies a function. The arguments to this function are a list of other expression objects.
- `relexpr_window()` applies a function over a window, similarly to the SQL `OVER` clause.
- `relexpr_set_alias()` assigns an alias to an expression.

### Usage

```r
new_relexpr(x, class = NULL)

relexpr_reference(name, rel = NULL, alias = NULL)

relexpr_constant(val, alias = NULL)

relexpr_function(name, args, alias = NULL)

relexpr_window(
    expr,
    partitions,
    order_bys = list(),
    offset_expr = NULL,
    default_expr = NULL,
    alias = NULL
)

relexpr_set_alias(expr, alias = NULL)
```

### Arguments

- `x` An object.
- `class` Classes added in front of the "relational_relexpr" base class.
- `name` The name of the column or function to reference.
- `rel` The name of the relation to reference.
- `alias` An alias for the new expression.
The value to use in the constant expression.

Function arguments, a list of expr objects.

An expr object.

Partitions, a list of expr objects.

which variables to order results by (list).

offset relational expression.

default relational expression.

Value

an object of class "relational_relexpr"
an object of class "relational_relexpr"
an object of class "relational_relexpr"
an object of class "relational_relexpr"
an object of class "relational_relexpr"
an object of class "relational_relexpr"

Examples

relexpr_set_alias(
 alias = "my_predicate",
 relexpr_function("<",
 list( relexpr_reference("my_number"), relexpr_constant(42) ) )
)
)

stats_show

Show stats

Description

Prints statistics on how many calls were handled by DuckDB. The output shows the total number of requests in the current session, split by fallbacks to dplyr and requests handled by duckdb.

Usage

stats_show()

Value

Called for its side effect.
Examples

```r
stats_show()

tibble(a = 1:3) %>%
  as_duckplyr_df() %>%
  mutate(b = a + 1)

stats_show()
```