Package 'mlr3db'

September 28, 2020

Title Data Base Backend for 'mlr3'

Version 0.2.0

Description Extends the 'mlr3' package with a backend to transparently work with data bases. Internally relies on the abstraction of package 'dbplyr' to interact with one of the many supported data base management systems (DBMS).

License LGPL-3

URL https:///mlr3db.mlr-org.com, https://github.com/mlr-org/mlr3db

BugReports https://github.com/mlr-org/mlr3db/issues

Depends R (>= 3.1.0)

Imports backports, checkmate, data.table, digest, dplyr, mlr3 (>= 0.3.0), mlr3misc, R6

Suggests DBI, RSQLite, dbplyr, future, future.apply, future.callr, lgr, testthat, tibble

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Author Michel Lang [cre, aut] (<https://orcid.org/0000-0001-9754-0393>)

Maintainer Michel Lang <michellang@gmail.com>

Repository CRAN

Date/Publication 2020-09-28 08:10:02 UTC

R topics documented:

mlr3db-package *mlr3db: Data Base Backend for 'mlr3'*

Description

Extends the 'mlr3' package with a backend to transparently work with data bases. Internally relies on the abstraction of package 'dbplyr' to interact with one of the many supported data base management systems (DBMS).

Author(s)

Maintainer: Michel Lang <michellang@gmail.com> (ORCID)

See Also

Useful links:

- https:///mlr3db.mlr-org.com
- https://github.com/mlr-org/mlr3db
- Report bugs at https://github.com/mlr-org/mlr3db/issues

as_sqlite_backend *Convert to use a SQLite Backend*

Description

Converts to a DataBackendDplyr using a **RSQLite** data base, depending on the input type:

- `data.frame`: Converts to a DataBackendDplyr.
- [mlr3::DataBackend]: Creates a new DataBackendDplyr using the data of the provided mlr3::DataBackend.
- [mlr3::Task]: Replaces the DataBackend in slot $task with a new backend. Only active columns and rows are considered.

Usage

```
as_sqlite_backend(data, path = NULL, ...)
```

Arguments

`data`	(`data.frame()` | mlr3::DataBackend | mlr3::Task) See description.
`path`	(`NULL` | `character(1)`) Path for the SQLite data base. Defaults to a file in the temporary directory of the R session, see `tempfile()`.
`...`	(any) Additional arguments, currently ignored.

Value

DataBackendDplyr.

`DataBackendDplyr` *DataBackend for dplyr/dbplyr*

Description

A mlr3::DataBackend using `dplyr::tbl()` from packages **dplyr/dbplyr**. This includes `tibbles` and abstract data base connections interfaced by **dbplyr**. The latter allows mlr3::Tasks to interface an out-of-memory data base.

Super class

`mlr3::DataBackend` -> `DataBackendDplyr`

Public fields

`levels` (named `list()`)
: List (named with column names) of factor levels as `character()`. Used to auto-convert character columns to factor variables.

`connector` (function())
: Function which is called to re-connect in case the connection became invalid.

Active bindings

`rownames` (`integer()`)
: Returns vector of all distinct row identifiers, i.e. the contents of the primary key column.

`colnames` (`character()`)
: Returns vector of all column names, including the primary key column.

`nrow` (`integer(1)`)
: Number of rows (observations).

`ncol` (`integer(1)`)
: Number of columns (variables), including the primary key column.

`valid` (`logical(1)`)
: Returns `NA` if the data does not inherits from `"tbl_sql"` (i.e., it is not a real SQL data base). Returns the result of `DBI::dbIsValid()` otherwise.

Methods

Public methods:

- `DataBackendDplyr$new()`
- `DataBackendDplyr$finalize()`
- `DataBackendDplyr$data()`
- `DataBackendDplyr$head()`

- DataBackendDplyr$distinct()
- DataBackendDplyr$missings()

Method `new()`**:** Creates a connection for a `dplyr::tbl()` object.

Usage:

```
DataBackendDplyr$new(
  data,
  primary_key,
  strings_as_factors = TRUE,
  connector = NULL
)
```

Arguments:

`data` (`dplyr::tbl()`)
: The data object.

`primary_key` (`character(1)`)
: Name of the primary key column.

`strings_as_factors` (`logical(1)` || `character()`)
: Either a character vector of column names to convert to factors, or a single logical flag: if `FALSE`, no column will be converted, if `TRUE` all string columns (except the primary key). For conversion, the backend is queried for distinct values of the respective columns on construction and their levels are stored in $levels.

`connector` (function())
: If not `NULL`, a function which re-connects to the data base in case the connection has become invalid. Database connections can become invalid due to timeouts or if the backend is serialized to the file system and then de-serialized again. This round trip is often performed for parallelization, e.g. to send the objects to remote workers. `DBI::dbIsValid()` is called to validate the connection. The function must return just the connection, not a `dplyr::tbl()` object!
  Note that this this function is serialized together with the backend, including possible sensitive information such as login credentials. These can be retrieved from the stored mlr3::DataBackend/mlr3::Task. To protect your credentials, it is recommended to use the **secret** package.
  Instead of calling the constructor yourself, you can call `mlr3::as_data_backend()` on a `dplyr::tbl()`. Note that only objects of class `"tbl_lazy"` will be converted to a DataBackendDplyr (this includes all connectors from **dbplyr**). Local `"tbl"` objects such as `tibbles` will converted to a DataBackendDataTable.

Method `finalize()`**:** Finalizer which disconnects from the data base. This is called during garbage collection of the instance.

Usage:

```
DataBackendDplyr$finalize()
```

Returns: `logical(1)`, the return value of `DBI::dbDisconnect()`.

Method `data()`**:** Returns a slice of the data. Calls `dplyr::filter()` and `dplyr::select()` on the table and converts it to a `data.table::data.table()`.
The rows must be addressed as vector of primary key values, columns must be referred to via column names. Queries for rows with no matching row id and queries for columns with no

matching column name are silently ignored. Rows are guaranteed to be returned in the same order as `rows`, columns may be returned in an arbitrary order. Duplicated row ids result in duplicated rows, duplicated column names lead to an exception.

Usage:

```
DataBackendDplyr$data(rows, cols, data_format = "data.table")
```

Arguments:

`rows integer()`
: Row indices.

`cols character()`
: Column names.

`data_format (character(1))`
: Desired data format, e.g. `"data.table"` or `"Matrix"`.

Method `head()`**:** Retrieve the first `n` rows.

Usage:

```
DataBackendDplyr$head(n = 6L)
```

Arguments:

`n (integer(1))`
: Number of rows.

Returns: `data.table::data.table()` of the first `n` rows.

Method `distinct()`**:** Returns a named list of vectors of distinct values for each column specified. If `na_rm` is `TRUE`, missing values are removed from the returned vectors of distinct values. Non-existing rows and columns are silently ignored.

Usage:

```
DataBackendDplyr$distinct(rows, cols, na_rm = TRUE)
```

Arguments:

`rows integer()`
: Row indices.

`cols character()`
: Column names.

`na_rm logical(1)`
: Whether to remove NAs or not.

Returns: Named `list()` of distinct values.

Method `missings()`**:** Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.

Usage:

```
DataBackendDplyr$missings(rows, cols)
```

Arguments:

`rows integer()`
: Row indices.

`cols character()`
: Column names.

Returns: Total of missing values per column (named `numeric()`).

Examples

```
# Backend using a in-memory tibble
data = tibble::as_tibble(iris)
data$Sepal.Length[1:30] = NA
data$row_id = 1:150
b = DataBackendDplyr$new(data, primary_key = "row_id")

# Object supports all accessors of DataBackend
print(b)
b$nrow
b$ncol
b$colnames
b$data(rows = 100:101, cols = "Species")
b$distinct(b$rownames, "Species")

# Classification task using this backend
task = mlr3::TaskClassif$new(id = "iris_tibble", backend = b, target = "Species")
print(task)
task$head()

# Create a temporary SQLite data base
con = DBI::dbConnect(RSQLite::SQLite(), ":memory:")
dplyr::copy_to(con, data)
tbl = dplyr::tbl(con, "data")

# Define a backend on a subset of the data base
tbl = dplyr::select_at(tbl, setdiff(colnames(tbl), "Sepal.Width")) # do not use column "Sepal.Width"
tbl = dplyr::filter(tbl, row_id %in% 1:120) # Use only first 120 rows
b = DataBackendDplyr$new(tbl, primary_key = "row_id")
print(b)

# Query disinct values
b$distinct(b$rownames, "Species")

# Query number of missing values
b$missings(b$rownames, b$colnames)

# Note that SQLite does not support factors, column Species has been converted to character
lapply(b$head(), class)

# Cleanup
rm(tbl)
DBI::dbDisconnect(con)
```

Index