Package 'mlr3db'

February 19, 2020

Title Data Base Backend for 'mlr3'

Version 0.1.5

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 (>= 0.1.4), R6

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

Encoding UTF-8

RoxygenNote 7.0.2

NeedsCompilation no

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

Maintainer Michel Lang <michellang@gmail.com>

Repository CRAN

Date/Publication 2020-02-19 12:40:12 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.

Format

R6::R6Class object inheriting from mlr3::DataBackend.

Construction

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

- `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). The backend is queried for distinct values of the respective columns 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.

Alternatively, use `mlr3::as_data_backend()` on a `dplyr::tbl()` to construct a DataBackend for you. 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.

Fields

All fields from mlr3::DataBackend, and additionally:

- `levels` :: named `list()`
  List of factor levels, named with column names. Referenced columns get automatically converted to factors in $data() and $head().
- `connector` :: function()
  Function which is called to re-connect in case the connection became invalid.
- `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

All methods from mlr3::DataBackend, and additionally:

- `finalize()`
  () -> `logical(1)`
  Finalizer which disconnects from the data base. Is called during garbage collection, but may also be called manually.

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