Package ‘arkdb’

October 31, 2018

Version 0.0.5

Title Archive and Unarchive Databases Using Flat Files

Description Flat text files provide a robust, compressible, and portable way to store tables from databases. This package provides convenient functions for exporting tables from relational database connections into compressed text files and streaming those text files back into a database without requiring the whole table to fit in working memory.

URL https://github.com/ropensci/arkdb

BugReports https://github.com/ropensci/arkdb/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

ByteCompile true

VignetteBuilder knitr

RoxygenNote 6.1.0

Imports DBI, progress, tools, utils

Suggests readr, spelling, dplyr, dbplyr, nycflights13, testthat, knitr, covr, fs, rmarkdown, RSQLite, MonetDBLite (>= 0.5.0)

Language en-US

NeedsCompilation no

Author Carl Boettiger [aut, cre, cph]

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN

Date/Publication 2018-10-31 21:10:03 UTC
**Description**

Flat text files provide a more robust, compressible, and portable way to store tables. This package provides convenient functions for exporting tables from relational database connections into compressed text files and streaming those text files back into a database without requiring the whole table to fit in working memory.

**Details**

It has two functions:

- `ark()`: archive a database into flat files, chunk by chunk.
- `unark()`: Unarchive flat files back into a database connection.

`arkdb` will work with any DBI supported connection. This makes it a convenient and robust way to migrate between different databases as well.

**Author(s)**

**Maintainer**: Carl Boettiger <cboettig@gmail.com> (0000-0002-1642-628X) [copyright holder]

Other contributors:

- Richard FitzJohn [contributor]

**See Also**

Useful links:

- [https://github.com/ropensci/arkdb](https://github.com/ropensci/arkdb)
- Report bugs at [https://github.com/ropensci/arkdb/issues](https://github.com/ropensci/arkdb/issues)
Ark

Archive tables from a database as flat files

Description

Archive tables from a database as flat files

Usage

ark(db_con, dir, streamable_table = streamable_base_tsv(),
    lines = 50000L, compress = c("bzip2", "gzip", "xz", "none"),
    tables = list_tables(db_con), method = c("keep-open", "window",
    "sql-window"), overwrite = "ask")

Arguments

db_con a database connection

dir a directory where we will write the compressed text files output

streamable_table interface for serializing/deserializing in chunks

lines the number of lines to use in each single chunk

compress file compression algorithm. Should be one of "bzip2" (default), "gzip" (faster
close times, a bit less compression), "xz", or "none", for no compression.

tables a list of tables from the database that should be archived. By default, will archive
all tables. Table list should specify schema if appropriate, see examples.

method method to use to query the database, see details.

overwrite should any existing text files of the same name be overwritten? default is "ask",

Details

ark will archive tables from a database as (compressed) tsv files. ark does this by reading only
chunks at a time into memory, allowing it to process tables that would be too large to read into
memory all at once (which is probably why you are using a database in the first place!) Compressed
text files will likely take up much less space, making them easier to store and transfer over networks.
Compressed plain-text files are also more archival friendly, as they rely on widely available and
long-established open source compression algorithms and plain text, making them less vulnerable
to loss by changes in database technology and formats.

In almost all cases, the default method should be the best choice. If the DBI::dbSendQuery()
implementation for your database platform returns the full results to the client immediately rather
than supporting chunking with n parameter, you may want to use "window" method, which is the
most generic. The "sql-window" method provides a faster alternative for databases like PostgreSQL
that support windowing natively (i.e. BETWEEN queries).
streamable_base_csv

streamable csv using base R functions

Description
streamable csv using base R functions

Usage
streamable_base_csv()

Details
Follows the comma-separate-values standard using utils::read.table()

Value
a streamable_table object (S3)

See Also
utils::read.table(), utils::write.table()
streamable_base_tsv

streamable_base_tsv   streamable tsv using base R functions

Description

streamable tsv using base R functions

Usage

streamable_base_tsv()

Details

Follows the tab-separate-values standard using `utils::read.table()`, see IANA specification at: https://www.iana.org/assignments/media-types/text/tab-separated-values

Value

a `streamable_table` object (S3)

See Also

`utils::read.table`, `utils::write.table`

streamable_readr_csv

streamable_readr_csv   streamable csv using readr

Description

streamable csv using readr

Usage

streamable_readr_csv()

Value

a `streamable_table` object (S3)

See Also

`readr::read_csv`, `readr::write_csv`
streamable_readr_tsv   streamable tsv using readr

Description

streamable tsv using readr

Usage

streamable_readr_tsv()

Value

a streamable_table object (S3)

See Also

readr::read_tsv(), readr::write_tsv()

streamable_table   streamable table

Description

streamable table

Usage

streamable_table(read, write, extension)

Arguments

read    read function. Arguments should be "file" (must be able to take a connection() object) and "..." (for) additional arguments.
write   write function. Arguments should be "data" (a data.frame), file (must be able to take a connection() object), and "omit_header" logical, include header (initial write) or not (for appending subsequent chunks)
extension    file extension to use (e.g. "tsv", "csv")

Details

Note several constraints on this design. The write method must be able to take a generic R connection object (which will allow it to handle the compression methods used, if any), and the read method must be able to take a textConnection object. readr functions handle these cases out of the box, so the above method is easy to write. Also note that the write method must be able to omit_header. See the built-in methods for more examples.
Value

a `streamable_table` object (S3)

Examples

```r
streamable_readr_tsv <- function() {
  streamable_table(
    function(file, ...) readr::read_tsv(file, ...),
    function(x, path, omit_header) 
      readr::write_tsv(x = x, path = path, omit_header = omit_header),
      "tsv")
}
```

Description

Unarchive a list of compressed tsv files into a database

Usage

```r
unark(files, db_con, streamable_table = NULL, lines = 50000L,
  overwrite = "ask", encoding = Sys.getenv("encoding", "UTF-8"),
  tablenames = NULL, ...)
```

Arguments

- **files**: vector of filenames to be read in. Must be tsv format, optionally compressed using bzip2, gzip, zip, or xz format at present.
- **db_con**: a database src (`src_dbi` object from `dplyr`)
- **streamable_table**: interface for serializing/deserializing in chunks
- **lines**: number of lines to read in a chunk.
- **overwrite**: should any existing text files of the same name be overwritten? default is "ask", which will ask for confirmation in an interactive session, and overwrite in a non-interactive script. TRUE will always overwrite, FALSE will always skip such tables.
- **encoding**: encoding to be assumed for input files.
- **tablenames**: vector of tablenames to be used for corresponding files. By default, tables will be named using lowercase names from file basename with special characters replaced with underscores (for SQL compatibility).
- ... additional arguments to `streamable_table$read` method.
Details

`unark` will read in a files in chunks and write them into a database. This is essential for processing large compressed tables which may be too large to read into memory before writing into a database. In general, increasing the `lines` parameter will result in a faster total transfer but require more free memory for working with these larger chunks.

If using `readr`-based streamable-table, you can suppress the progress bar by using `options(readr.show_progress = FALSE)` when reading in large files.

Value

the database connection (invisibly)

Examples

```r
## Setup: create an archive.
library(dplyr)
dir <- tempdir()
db <- dbplyr::nycflights13_sqlite(tempdir())

## database -> .tsv.bz2
unark(db, dir)

## list all files in archive (full paths)
files <- list.files(dir, "[.]tsv\b\b\b\b2$", full.names = TRUE)

## Read archived files into a new database (another sqlite in this case)
new_db <- src_sqlite(file.path(dir, "local.sqlite"), create=TRUE)
unark(files, new_db)

## Prove table is returned successfully.
tbl(new_db, "flights")
```
Index

ark, 3
ark(), 2
arkdb (arkdb-package), 2
arkdb-package, 2

connection(), 6

DBI::dbSendQuery(), 3

readr::read_csv(), 5
readr::read_tsv(), 6
readr::write_csv(), 5
readr::write_tsv(), 6

streamable_base_csv, 4
streamable_base_tsv, 5
streamable_readr_csv, 5
streamable_readr_tsv, 6
streamable_table, 6

unark, 7
unark(), 2

utils::read.table(), 4, 5
utils::write.table(), 4, 5