**Breaking change**

> install.packages('MonetDBLite')

Warning in install.packages :

package ‘MonetDBLite’ is not available (for R version 3.6.1)

People who based their works on MonetDBLite may ask what happened, what to do. Not to play a risky game with database and tools choices for future works… (“It’s really fast but we may waste some time if we have to replace it by another solution”).

It’s the game with open source. Remember big changes in dplyr 0.7.   
Sometimes we want better tools, and most of the time they become better. It’s really great.  
And sometimes we don’t have time and energy to adapt our work to tools that became better in a too iterative way. Or in a too subjective way.   
We want it to work, not break.  
Keeping code as simple as possible (and avoid nebulous dependencies, so, tidy?) is one of the key point.   
Stocking data in a database is another one.

**All that we can say is that “we’re walking on works in progress”. Like number of eggshells, more works in progress here probably means more breaking changes.**

Works in progress for packages, also for (embedded) databases!

**From Monet to Duck**

MonetDBLite philosophy is to be like a “very very fast SQLite”. But it’s time for change (or it seems to be).  
Then we can thanks MonetDBLite developers as it was a nice adventure to play/work with MonetDB speed!  
As a question, is there another person, some volunteers, possibilities to maintain MonetDBLite (somewhere a nice tool)?  
There are not so many informations for the moment about what happened and that’s why I write this post.

[Here](https://github.com/MonetDB/MonetDBLite-R/issues/38#issuecomment-490589276), I read that they are now working on a new solution, under MIT License, named [DuckDB](https://www.duckdb.org/), see [here](https://hannes.muehleisen.org/SIGMOD2019-demo-duckdb.pdf) for more details.

As I’m just a R user and haven’t collaborate to the project, I would just say for short: DuckDB takes good parts from SQLite and PostGreSQL (Parser), see [here](https://github.com/cwida/duckdb#standing-on-the-shoulders-of-giants) for complete list, it looks promising. As in MonetDB, philosophy is focused on columns and speed. And dates for instance are handled correctly, not having to convert them in “ISO-8601 – like” character strings.

It can be called from C/C++, Python and R.

[Here](https://uwekorn.com/2019/10/19/taking-duckdb-for-a-spin.html) is a post about python binding.

I also put a link at the bottom of this page which give some explanations about the name of this new tool and DuckDB developers point’s of view about data manipulation and storage[1](https://guillaumepressiat.github.io/blog/2019/10/duckdb#fn:2).

**Beginning with duckDB in R**

**Create / connect to the db**

# remotes::install\_github("cwida/duckdb/tools/rpkg", build = FALSE)

library(duckdb)

library(dplyr)

library(DBI)

# Create or connect to the db

con\_duck <- dbConnect(duckdb::duckdb(), "~/Documents/data/duckdb/my\_first.duckdb")

#con <- dbConnect(duckdb::duckdb(), ":memory:")

con\_duck

**iris**

dbWriteTable(con\_duck, "iris", iris)

tbl(con, 'iris')

**Put some rows and columns in db**

> dim(nycflights13::flights)

[1] 336776 19

> object.size(nycflights13::flights) %>% format(units = "Mb")

[1] "38.8 Mb"

Sampling it to get more rows, then duplicating columns, two time.

# Sample to get bigger data.frame

df\_test <- nycflights13::flights %>%

sample\_n(2e6, replace = TRUE) %>%

bind\_cols(., rename\_all(., function(x){paste0(x, '\_bind\_cols')})) %>%

bind\_cols(., rename\_all(., function(x){paste0(x, '\_bind\_cols\_bis')}))

> dim(df\_test)

[1] 2000000 76

> object.size(df\_test) %>% format(units = "Mb")

[1] "916.4 Mb"

Write in db

tictoc::tic()

dbWriteTable(con\_duck, "df\_test", df\_test)

tictoc::toc()

It take some times compared to MonetDBLite (no benchmark here, I just run this several times and it was consistent).

# DuckDB : 23.251 sec elapsed

# SQLite : 20.23 sec elapsed

# MonetDBLite : 8.4 sec elapsed

The three are pretty fast.  
Most importantly if queries are fast, and they are, most of the time we’re allwright.

**I want to say here that’s for now it’s a work in progress, we have to wait more communication from DuckDB developers. I just write this to share the news.**

**Glimpse**

> tbl(con\_duck, 'df\_test') %>% glimpse()

Observations: ??

Variables: 76

Database: duckdb\_connection

$ year <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,…

$ month <int> 11, 10, 3, 5, 12, 9, 7, 3, 9, 4, 7, 6, 1, 1, 9, 10, 9, 8, 4, 1, 4, 9, 6…

$ day <int> 29, 7, 1, 2, 18, 18, 20, 7, 15, 25, 22, 1, 29, 18, 30, 27, 27, 22, 19, …

$ dep\_time <int> 1608, 2218, 1920, NA, 1506, 1917, 1034, 655, 1039, 1752, 2018, 1732, 82…

$ sched\_dep\_time <int> 1612, 2127, 1920, 2159, 1500, 1900, 1030, 700, 1045, 1720, 1629, 1728, …

$ dep\_delay <dbl> -4, 51, 0, NA, 6, 17, 4, -5, -6, 32, 229, 4, -9, -3, -4, -3, 9, 38, 34,…

$ arr\_time <int> 1904, 2321, 2102, NA, 1806, 2142, 1337, 938, 1307, 2103, 2314, 1934, 11…

$ sched\_arr\_time <int> 1920, 2237, 2116, 2326, 1806, 2131, 1345, 958, 1313, 2025, 1927, 2011, …

$ arr\_delay <dbl> -16, 44, -14, NA, 0, 11, -8, -20, -6, 38, 227, -37, -16, -12, -10, -39,…

$ carrier <chr> "UA", "EV", "9E", "UA", "DL", "DL", "VX", "UA", "UA", "AA", "B6", "UA",…

$ flight <int> 1242, 4372, 3525, 424, 2181, 2454, 187, 1627, 1409, 695, 1161, 457, 717…

$ tailnum <chr> "N24211", "N13994", "N910XJ", NA, "N329NB", "N3749D", "N530VA", "N37281…

$ origin "EWR", "EWR", "JFK", "EWR", "LGA", "JFK", "EWR", "EWR", "EWR", "JFK", "…

$ dest <chr> "FLL", "DCA", "ORD", "BOS", "MCO", "DEN", "SFO", "PBI", "LAS", "AUS", "…

$ air\_time <dbl> 155, 42, 116, NA, 131, 217, 346, 134, 301, 230, 153, 276, 217, 83, 36, …

$ distance <dbl> 1065, 199, 740, 200, 950, 1626, 2565, 1023, 2227, 1521, 1035, 2133, 138…

$ hour <dbl> 16, 21, 19, 21, 15, 19, 10, 7, 10, 17, 16, 17, 8, 14, 8, 19, 15, 16, 20…

$ minute <dbl> 12, 27, 20, 59, 0, 0, 30, 0, 45, 20, 29, 28, 35, 50, 25, 0, 35, 55, 0, …

$ time\_hour <dttm> 2013-11-29 21:00:00, 2013-10-08 01:00:00, 2013-03-02 00:00:00, 2013-05…

..

..

..

$ minute\_bind\_cols <dbl> 12, 27, 20, 59, 0, 0, 30, 0, 45, 20, 29, 28, 35, 50, 25, 0, 35, 55, 0, …

$ time\_hour\_bind\_cols <dttm> 2013-11-29 21:00:00, 2013-10-08 01:00:00, 2013-03-02 00:00:00, 2013-05…

$ year\_bind\_cols\_bis <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,…

$ month\_bind\_cols\_bis <int> 11, 10, 3, 5, 12, 9, 7, 3, 9, 4, 7, 6, 1, 1, 9, 10, 9, 8, 4, 1, 4, 9, 6…

$ day\_bind\_cols\_bis <int> 29, 7, 1, 2, 18, 18, 20, 7, 15, 25, 22, 1, 29, 18, 30, 27, 27, 22, 19, …

..

..

..

$ distance\_bind\_cols\_bind\_cols\_bis <dbl> 1065, 199, 740, 200, 950, 1626, 2565, 1023, 2227, 1521, 1035, 2133, 138…

$ hour\_bind\_cols\_bind\_cols\_bis <dbl> 16, 21, 19, 21, 15, 19, 10, 7, 10, 17, 16, 17, 8, 14, 8, 19, 15, 16, 20…

$ minute\_bind\_cols\_bind\_cols\_bis <dbl> 12, 27, 20, 59, 0, 0, 30, 0, 45, 20, 29, 28, 35, 50, 25, 0, 35, 55, 0, …

$ time\_hour\_bind\_cols\_bind\_cols\_bis <dttm> 2013-11-29 21:00:00, 2013-10-08 01:00:00, 2013-03-02 00:00:00, 2013-05…

**Count**

> tbl(con\_duck, 'df\_test') %>% count()

# Source: lazy query [?? x 1]

# Database: duckdb\_connection

n

<dbl>

1 2000000

**Dates**

Compared to SQLite it handles dates/times correctly. No need to convert in character.

tbl(con\_duck, 'df\_test') %>% select(time\_hour)

# Source: lazy query [?? x 1]

# Database: duckdb\_connection

time\_hour

<dttm>

1 2013-11-29 21:00:00.000000

2 2013-10-08 01:00:00.000000

3 2013-03-02 00:00:00.000000

4 2013-05-03 01:00:00.000000

5 2013-12-18 20:00:00.000000

6 2013-09-18 23:00:00.000000

7 2013-07-20 14:00:00.000000

8 2013-03-07 12:00:00.000000

9 2013-09-15 14:00:00.000000

10 2013-04-25 21:00:00.000000

# … with more rows

tbl(con\_sqlite, 'df\_test') %>% select(time\_hour)

# Source: lazy query [?? x 1]

# Database: sqlite 3.22.0 [/Users/guillaumepressiat/Documents/data/sqlite.sqlite]

time\_hour

<dbl>

1 1385758800

2 1381194000

3 1362182400

4 1367542800

5 1387396800

6 1379545200

7 1374328800

8 1362657600

9 1379253600

10 1366923600

# … with more rows

**Some querying**

Running some queries

**dplyr**

It already works nicely with dplyr.

> tbl(con\_duck, 'iris') %>%

+ group\_by(Species) %>%

+ summarise(min(Sepal.Width)) %>%

+ collect()

# A tibble: 3 x 2

Species `min(Sepal.Width)`

<chr> <dbl>

1 virginica 2.2

2 setosa 2.3

3 versicolor 2

> tbl(con\_duck, 'iris') %>%

+ group\_by(Species) %>%

+ summarise(min(Sepal.Width)) %>% show\_query()

<SQL>

SELECT "Species", MIN("Sepal.Width") AS "min(Sepal.Width)"

FROM "iris"

GROUP BY "Species"

**sql**

Run query as a string

dbGetQuery(con\_duck, 'SELECT "Species", MIN("Sepal.Width") FROM iris GROUP BY "Species"')

Species min(Sepal.Width)

1 virginica 2.2

2 setosa 2.3

3 versicolor 2.0

Like for all data sources with DBI, if the query is more complex, we can write it comfortably in an external file and launch it like this for example:

dbGetQuery(con\_duck, readr::read\_file('~/Documents/scripts/script.sql'))

**“Little” benchmarks**

**Collecting this big data frame**

This has no sense but give some idea of read speed. We collect df\_test in memory, from duckdb, monetdb and sqlite.

> microbenchmark::microbenchmark(

+ a = collect(tbl(con\_duck, 'df\_test')),

+ times = 5)

Unit: seconds

expr min lq mean median uq max neval

a 3.58703 3.632507 3.763129 3.676669 3.725148 4.19429 5

> microbenchmark::microbenchmark(

+ b = collect(tbl(con\_monet, 'df\_test')),

+ times = 5)

Unit: milliseconds

expr min lq mean median uq max neval

b 973.1111 990.3699 1003.417 1010.651 1013.858 1029.097 5

> microbenchmark::microbenchmark(

+ d = collect(tbl(con\_sqlite, 'df\_test')),

+ times = 1)

Unit: seconds

expr min lq mean median uq max neval

d 52.08785 52.08785 52.08785 52.08785 52.08785 52.08785 1

Really good !

**Simple count**

Count then collect aggregate rows.

> microbenchmark::microbenchmark(

+ a = collect(tbl(con\_duck, 'df\_test') %>% count(year, month)),

+ times = 20)

Unit: milliseconds

expr min lq mean median uq max neval

a 50.18014 53.24197 54.87532 54.68203 57.09206 58.94873 20

> microbenchmark::microbenchmark(

+ b = collect(tbl(con\_monet, 'df\_test') %>% count(year, month)),

+ times = 20)

Unit: milliseconds

expr min lq mean median uq max neval

b 151.729 157.9267 160.5727 160.8815 163.8343 167.477 20

> microbenchmark::microbenchmark(

+ d = collect(tbl(con\_sqlite, 'df\_test') %>% count(year, month)),

+ times = 20)

Unit: seconds

expr min lq mean median uq max neval

d 2.167202 2.196288 2.205281 2.20486 2.216594 2.253606 20

Faster !

It remains to test joins, filters, sorts, etc.

**Informations**

I find that there are not so many communications for the moment about this work and binding for R, so I made this post to highlight it.

MonetDBLite speed is amazing, do you will give DuckDB a try ?

In any case thanks to DuckDB developers and welcome to the new duck.