Database interoperability for spatial objects in R

Etienne Racine

2020-04-02

Project team

Etienne Racine and Edzer Pebesma. Etienne Racine has written all the database connection code in `sf`, Edzer Pebesma is main author and maintainer of `sf`.

The Problem

Large spatial data can be hard to analyze with R because of RAM limitations. Users often turn to spatial databases for this process and go back-and-forth with R, since R is still needed for e.g. modeling. Also corporations and large research groups, among others, often store data in a central database. The ability to interact directly with a spatial database, from R, can accelerate the work of R users.

The `sf` package is built on the simple feature OGC standard, which facilitates the interoperability with spatial databases. We already provide read and write operations to PostGIS from `sf`, and we can also integrate with the `dbplyr` package to push execution into the database. The `gdal` driver within `sf` also provides interoperability, but does not allow integration with the `dplyr` workflow.

This proposal is to :

- Complete and clarify the interface for the `RPostgres` and `odbc` drivers by creating a separate package, `sfdbi`, to provide spatial database interface
- Create a tutorial and good documentation to use the spatial database interface

The proposal

Overview

This proposal builds on the successfully R-Consortium funded *Simple Features for R* (`sf`) project. The `sf` package already supports direct read and write for PostGIS, one of the most powerful and commonly used spatial databases, and can use the `dbplyr` package to push execution into the database. This feature however is not complete, not well documented, and little used.

The proposal is to complete the database support by making more `sf` operations compatible with spatial databases, facilitate integration tests by moving database support from `sf` to another package (`sfdbi`) and make it easier to add support for other databases, thereby unlocking many other spatial databases compatible with the OGC standard. The following table provides an overview of the potential of the project by presenting some of the most important spatial databases in use.

Table 1 Spatial databases and drivers. This proposal focuses on the first two rows (`RPostgres` and `odbc`), and the new architecture will make it easier to add more drivers.

Database	Spatial extension	R Driver	Current support (1)
PostgreSQL	Postgis	RPostgres	Partial
PostgreSQL	Postgis	odbc	Partial - Not tested
PostgreSQL	Postgis	RPostgreSQL	Partial
Couchdb	Geocouch	sofa	Unknown
PostgreSQL	ArcSDE	odbc	Unknown - Reported usage
Spark	Geospark	sparklyr	Partial with geospark package
Oracle	Oracle Spatial	odbc	Unknown
Microsoft SQL Server	Native	odbc	Unknown - Reported usage
MySQL	Native (primitive)	odbc	Unknown
SQlite	SpatialLite	RSQLite	Unknown
MariaDB	Native	RMariaDB	Unknown
BigQuery	BigQuery GIS	bigrquery	Unknown
H2	H2GIS	RH2	Unknown
Redshift	Native (primitive)	odbc	Unknown
MongoDB	Native (primitive)	mongolite	Unknown
Parquet	None	arrow	Unknown - Speculative
Feather	None	arrow	Unknown - Speculative

Notes (1):

- *Partial support*: is currently supported by `sf`, but the set of features is limited
- *Not tested*: not included in the continuous integration tests (although there are some ad-hoc tests that can be run manually)
- *Unknown*: not tested, might work partially
- *Reported usage*: not tested, but users have notified they had used it
- *Speculative*: we've had discussions with the Ursa Labs team, and it seems possible and will require development on our side and on the arrow side – this would greatly facilitate interoperability with the spatial python ecosystem and geopandas.

Detail

Minimally, we propose to support `dplyr` workflows with `collect`, `tbl`, `copy_to` *et al.* methods for `postgis_connection` objects. The current implementation makes it hard to distinguish specific implementations of the driver, postgis and postgres. The proposed architecture will make it easier to build spatial database connectors (see table 1 for potential).

While `sf` already has many compatible functions, we want to handle exceptions and make `sf` functions compatible with postgis to the extent possible.

We will teach, through a tutorial, how to work with the database interface, and document how to implement database connectors.

Project plan

The plan is to:

- Extract the database interface in a separate package by refactoring code in `sf` (it will make maintenance easier and accelerate continuous integration checks)
- remove `DBI` dependency of `sf`
- adapt, in `sf`, to the extent possible, functionalities for the database interaction (transfer learnings from `etiennebr/geotidy`)

- build a second package, `sfdbi`, to contain the spatial interface
- create a Tutorial to interact with a spatial database
- create a Reference guide to implement spatial database connectors

Start-up phase

The collaboration will happen through the r-spatial GitHub organisation, using issues on public repositories. The `sf` repository will receive the updates, the `sfdbi` package will be hosted on r-spatial and all public communication and status updates will happen on r-spatial/discuss, the r-sgi-geo mailing list and using the #rspatial tag on twitter.

The `sfdbi` will be added to the r-spatial github org. The license, code of conduct will be the same than `sf` and we'll use the tidyverse template for other cases.

Technical delivery

The intended design is to create a spatial database class that could be used to extend the `tbl` functionality to read and write spatial data using `copy_to()` and `collect()`. This is what a minimal usage would look like:

```
# A DBI connection modifier (to allow support for more
# spatial databases (see table 1)
postgis <- function(...){
  structure(..., class = c("postgis_connection", class(...)))
}

copy_to.postgis_connection <- function(con,...) {
  x <- dbWriteTable(con, ...)  # we already have a dbWriteTable
  structure(x, class = c("tbl_sf", class(...)))
}

tbl.sf <- function(...){
  structure(..., class = c("tbl_sf", class(...)))
}

copy_to.tbl_sf <- function(con,...) {
  dbWriteTable(con, ...)  # we already have a sf::dbWriteTable for DBIObject
}

collect.tbl_sf <- function(con,...) {
  # execute the remote query via dbplyr
  # [...]
  st_read(con, x)  # we already have a sf::st_read.DBIObject
}

# Usage
con <- DBI::dbConnect(RPostgres::Postgres()) %>%
  postgis()

# Copy spatial data to database
x <- copy_to(con, tibble(
  lon = 1:3,
  lat = 3:1,
```

```
  geom = st_makepoint(longitude, latitude)
))

# execute in the database
y <- x %>%
  mutate(
    geom = st_buffer(geom, 10)
  )

collect(y)  # will return a tbl with an sfc column
```

Other aspects

- Announcement post early July through #rspatial, r-spatial/discuss, r-sgi-geo
- Tutorial and reference guide at the end of September announced on #rspatial, r-spatial/discuss, r-sgi-geo

Requirements

The path to the solution is clear, and the implementation seems to be relatively straightforward. In the worst case, we will minimize the changes to `sf` to avoid breaking existing code. We don't expect to require changes to `dbplyr`, but this could be a source of delays or modification to the plan. Package `odbc` could also need some modifications, but we have already existing workarounds. We have not identified other external factors that could impact this project.

People

Etienne Racine will lead the project: refactor, develop and document; Edzer Pebesma will be involved in global design and change review for `sf`.

Processes

Etienne Racine and Edzer Pebesma will communicate through github, emails and Skype when needed. We'll engage the R spatial community as much as possible. The new package `sfdbi` will be added to the the `r-spatial` GitHub organisation and will adhere to the code of conduct of the `sf` package. The tutorials will be shared via github pages and ideally could be run on docker, rstudio.cloud, or netlify to make it easy for the community to use. The accomplishments will be documented and advertised on blog posts on `r-spatial.org`.

Tools & Tech

We will use GitHub actions to systematically test on linux (currently the only OS to support services on github actions). We might perform ad-hoc tests for other OS when needed.

Funding

We request US$ 6,000 for this project.

Summary

- Refactor `sf` (2000)
- Develop `sfdbi` (2000)
- Write a tutorial and polish documentation (2000)

Success

Definition of done

We can interact with a `PostgreSQL` database from R by using `dplyr`, while using `sf` expressions for spatial operations. Workflows and case studies are documented in vignettes and blog posts. Docker files show complete, reproducible examples that include setting up of a spatial database and loading it with data. Users start using it because the documentation is clear and there are good tutorials.

Measuring success

Success is achieved when:

- working and tested code is published,
- reproducible examples (e.g. in GitHub actions and docker files) are published,
- documentation, vignettes and blog posts advertise the new functionality.

Future work

This work will make it easier to add support for further spatial databases (table 1).

Key risks

The database connector code (odbc, DBI, dbplyr) is not in our hands; in the past however we have had several productive interactions, including at rstudio::conf meetings, with Kirill Mueller, Jim Hester, and Hadley Wickham.