Using DatabaseConnector through DBI and dbplyr

Martijn J. Schuemie

2023-01-30

Contents

1	Introduction	1
2	Connecting	1
3	Querying	2
4	Using dbplyr 4.1 Date functions	
5	Temp tables5.1 Temp tables in dbplyr5.2 Cleaning up emulated temp tables	3 3
6	Closing the connection	4

1 Introduction

This vignette describes how to use the DatabaseConnector package through the DBI and dbplyr interfaces. It assumes you already know how to create a connection as described in the 'Connecting to a database' vignette.

All functions of the DatabaseConnector DBI interface apply SQL translation, thus making it an interface to a virtual database platform speaking OHDSISql as defined in SqlRender.

2 Connecting

We can use the dbConnect() function, which is equivalent to the connect() function:

```
connection <- dbConnect(
  DatabaseConnectorDriver(),
  dbms = "postgresql",
  server = "localhost/postgres",
  user = "joe",
  password = "secret"
)</pre>
```

Connecting using PostgreSQL driver

```
dbIsValid(conn)
```

[1] TRUE

3 Querying

Querying and executing SQL can be done through the usual DBI functions. SQL statements are assumed to be written in 'OhdsiSql', a subset of SQL Server SQL (see the SqlRender package for details), and are automatically translated to the appropriate SQL dialect. For example:

```
dbGetQuery(connection, "SELECT TOP 3 * FROM cdmv5.person")
     person_id gender_concept_id year_of_birth
## 1
             1
                             8507
## 2
             2
                             8507
                                            1976
## 3
             3
                             8507
                                            1977
Or:
res <- dbSendQuery(connection, "SELECT TOP 3 * FROM cdmv5.person")
dbFetch(res)
     person_id gender_concept_id year_of_birth
## 1
             1
                             8507
                                            1975
## 2
             2
                             8507
                                            1976
## 3
             3
                             8507
                                            1977
dbHasCompleted(res)
## [1] TRUE
dbClearResult(res)
dbDisconnect(res)
```

4 Using dbplyr

We can create a table based on a DatabaseConnector connection. The inDatabaseSchema() function allows us to use standard databaseSchema notation promoted by SqlRender:

```
library(dpylr)
person <- tbl(connection, inDatabaseSchema("cdmv5", "person"))</pre>
person
     person_id gender_concept_id year_of_birth
##
## 1
              1
                               8507
              2
## 2
                                              1976
                               8507
## 3
              3
                               8507
                                              1977
we can apply the usual dplyr syntax:
```

```
person %>%
  filter(gender_concept_id == 8507) %>%
  count() %>%
  pull()
```

[1] 1234

4.1 Date functions

The dbplyr package does not support date functions, but DatabaseConnector provides the dateDiff(), dateAdd(), eoMonth(), dateFromParts(), year(), month(), and day() functions that will correctly translate to various data platforms:

```
observationPeriod <- tbl(connection, inDatabaseSchema("cdmv5", "observation_period"))
observationPeriod %>%
  filter(
   dateDiff("day", observation_period_start_date, observation_period_end_date) > 365
) %>%
  count() %>%
  pull()
```

[1] 987

4.2 Allowed table and field names in dbplyr

Because of the many idiosyncrasies in how different dataplatforms store and transform table and field names, it is currently not possible to use any names that would require quotes. So for example the names person, person_id, and observation_period are fine, but Person ID and Obs. Period are not. In general, it is highly recommend to use lower case snake_case for database table and field names.

5 Temp tables

The DBI interface uses temp table emulation on those platforms that do not support real temp tables. This does require that for those platforms the user specify a tempEmulationSchema, preferably using

```
option(sqlRenderTempEmulationSchema = "a_schema")
```

Where "a_schema" refers to a schema where the user has write access. If we know we will need temp tables, we can use the assertTempEmulationSchemaSet() to verify this option has been set. This function will throw an error if it is not set, but only if the provided dbms is a platform that requires temp table emulation.

In OHDSISql, temp tables are referred to using a '#' prefix. For example:

```
dbWriteTable(connection, "#temp", cars)
```

Inserting data took 0.053 secs

5.1 Temp tables in dbplyr

The copy_to function creates a temp table:

```
carsTable <- copy_to(connection, cars)</pre>
```

Created a temporary table named #cars

The compute() function also creates a temp table, for example:

```
tempTable <- person %>%
  filter(gender_concept_id == 8507) %>%
  compute()
```

Created a temporary table named #dbplyr 001

5.2 Cleaning up emulated temp tables

Emulated temp tables are not really temporary, and therefore have to be removed when no longer needed. A convenient way to drop all emulated temp tables created so far in an R session is using the dropEmulatedTempTables() function:

```
dropEmulatedTempTables(connection)
```

In our example, this does not do anything because were using a PostgreSQL server, which does natively support temp tables.

6 Closing the connection

We can use the dbDisconnect() function, which is equivalent to the disconnect() function: dbDisconnect(connection)