When using shiny in production, often you will want to have some sort of database interactions for storing and accessing data. The `DBI` package provides an easy way to do the database interactions with a variety of SQL database flavors. In this example, I'm going to use a SQLite in memory database for reproducibility. In practice, you will just switch to the code to use a persistent database. Let's start by creating a table to right to.

```
library(RSQLite)
library(DBI)
con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbExecute(con, 'CREATE TABLE mytable (col1 int, col2 varchar(10));')
```

```
## [1] 0
```

Check that the table exists now.

```
dbListTables(con)
```

```
## [1] "mytable"
```

Add a single row to the table to check that we can insert.

```
dbExecute(con, 'INSERT INTO mytable ([col1], [col2]) VALUES (1, "b");"')
```

```
## [1] 1
```

```
dbGetQuery(con, 'SELECT col1, col2 FROM mytable')
```

```
##   col1 col2
## 1    1    b
```

So now we have a table in the database and we can write to it. When the shiny application loads, we will want to show the user what is in the table.

```
mytableinshiny <- reactive({
  dbGetQuery(con, 'SELECT col1, col2 from mytable')
})
mytableinshiny()
```

```
##   col1 col2
## 1    1    b
```

This works great and now we can use the data from the database to do whatever we need it to do in the shiny application. If we create the following action, the function does the necessary side effect of writing to the database but unless the user reloads the application `mytableinshiny` does not update in the application.

```
observeEvent(input$writetodb, {
  sql <- sqlInterpolate(con, 'INSERT INTO mytable ([col1], [col2]) VALUES (col1,
?col2);"',
                        col1 = input$col1, col2 = input$col2)
  dbExecute(con, sql)
})
```

```
mytableinshiny()
```

```
##   col1 col2
## 1    1    b
```

After the database insertion, we need a way to inform shiny to update the

reactive data.frame. We can use a reactive trigger to "push" a button that lets shiny know that the database has changed and it should re-execute the `SELECT` query.

```
makereactivetrigger <- function() {
  rv <- reactiveValues(a = 0)
  list(
    depend = function() {
      rv$a
      invisible()
    },
    trigger = function() {
      rv$a <- isolate(rv$a + 1)
    }
  )
}
dbtrigger <- makereactivetrigger()
```

The trigger is now an object that has a couple of functions to pass messages around. We need to set the reactive value to re-execute so add the `depend` function and after the database interaction executes we add the `trigger` function. Also, it's good practice to use `sqlInterpolate` to help prevent sql injection attacks.

```
mytableinshiny <- reactive({
  dbtrigger$depend()
  dbGetQuery(con, 'SELECT col1, col2 from mytable')
})
observeEvent(input$writetodb, {
  sql <- sqlInterpolate(con, 'INSERT INTO mytable ([col1], [col2]) VALUES
(?col1, ?col2)',
                        col1 = input$col1, col2 = input$col2)
  dbExecute(con, sql)
  dbtrigger$trigger()
})
```

Within the app, we put the data in the shiny input forms and hit the write to database action button when ready.

```
mytableinshiny()
```

```
##   col1 col2
## 1    1    b
## 2    2    a
```

Now when the database is updated the shiny application syncs to what is in the database after the insertion.

Here's a toy example to play with:

```
library(shiny)
library(RSQLite)
library(DBI)
makereactivetrigger <- function() {
    rv <- reactiveValues(a = 0)
    list(
        depend = function() {
            rv$a
            invisible()
        },
```

```
        trigger = function() {
            rv$a <- isolate(rv$a + 1)
        }
    )
}
dbtrigger <- makereactivetrigger()
con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbExecute(con, 'CREATE TABLE mytable (col1 int, col2 varchar(10));')
ui <- fluidPage(
    numericInput('col1', 'col1', value = 1L, step = 1L),
    textInput('col2', 'col2', value = 'a'),
    actionButton('writetodb', 'Save'),
    tableOutput('dbtable')
)
server <- function(input, output) {
    mytableinshiny <- reactive({
        dbtrigger$depend()
        dbGetQuery(con, 'SELECT col1, col2 from mytable')
    })
    observeEvent(input$writetodb, {
        sql <- sqlInterpolate(con, 'INSERT INTO mytable ([col1], [col2]) VALUES
(?col1, ?col2)',
                              col1 = input$col1, col2 = input$col2)
        dbExecute(con, sql)
        dbtrigger$trigger()
    })
    output$dbtable <- renderTable({
        mytableinshiny()
    })
}
shinyApp(ui = ui, server = server)
```
