

Database Tutorial

Jake Eisaguirre

2/3/22

Table of contents

RIBBITR Database Background	1
Non-RDBMS Example	2
RDMS Example	3
How to Connect and Interact with the RIBBITR Data Base	6
Necessary Library's	6
Connecting	6
Sample .Renviron file and Database Connection	7
Interacting	8
dbQetQuery	8
Entire SQL Code Chunk	9

RIBBITR Database Background

Our RIBBITR data is stored in a PostgreSQL database, more commonly called “postgres”. Postgres is a free and open source relational database management system (RDBMS). Postgres RDBMS began development in 1982 at UC Berkeley initially under the name Ingres. After many years of development the project ended and Postgres was released under an MIT License that allowed other developers to utilize the code (open source) and continue developing the code. Following this, many evolution's of the query language “SQL” occurred. By 1997 PostgreSQL was released and SQL was deemed the unifying language for RDBMS style databases. To this day, developers around the world work to build upon the open source software of PostgreSQL and SQL to improve the software. Side note: RDBMS are one “flavor” of databases with new methods for data management systems being developed constantly, like “graph databases”.

Postgres is considered a relational database management system (RDBMS) because each observation/row in a table contains a relationship to another observation/row in a different table. By maintaining relationships between data points you ensure the observations in that row will be joined correctly to each table. The rigid structure of the relationship between observations/rows from table to table is maintained through **primary keys** and **foreign keys**. A **primary key** from one table will be joined to the **foreign key** of the other table in which each row/observation will then align correctly between the two tables.

Non-RDBMS Example

For example, if you did not have a RDBMS and you had a `wind_speed` table that contains columns for “site”, “date”, and “wind_speed” and a `water_temp` table containing columns for “site”, “date”, and “water_temp” .

site	date	wind_speed
A	2022-01-03	22
B	2022-08-10	24
C	2022-05-07	36
A	2022-08-12	5
C	2022-09-14	12

site	date	water_temp
A	2022-01-03	3
B	2022-08-10	6
C	2022-05-07	23
A	2022-08-12	18
C	2022-09-14	3

Now say you wanted to join those two tables to have one table containing “site”, “date”, “wind_speed”, and “water_temp” you would `water_wind <- inner_join(water_temp, wind_speed, by = c("site", "date"))` on the unique relationship of “site” and “date”. Or in human speak, when site and date between the two tables equal each other place the “water_temp” row in the table. With a RDBMS the creation of primary and foreign keys maintains the unique relationship of “site” and “date” between the two tables.

site	date	water_temp	wind_speed
A	2022-01-03	3	22
B	2022-08-10	6	24
C	2022-05-07	23	36
A	2022-08-12	18	5
C	2022-09-14	3	12

RDMS Example

When looking at the `location` table and the `region` table within the `survey_data` schema in the RIBBITR database we can see how the RDMS preserves the data and reduces space. We have three unique locations where data can be collected “panama”, “brazil”, and “usa” in which each of those locations can have many regions. For example, “usa” has regions of “california”, “new_mexico”, “vermont”, “tennessee”, “louisiana”, and “pennsylvania”. Through the use of primary and foreign keys we now know how to join those tables. And due to the primary and foreign keys we only need to maintain the unique sites and regions. This saves space by removing repeated values and preserves the data by only having one spelling of “usa”. You could imagine with the above example, if sites had upper case and lower case spellings we could improperly join the data.

Location table

	asc location	location_id
1	panama	350ec32a-15f0-4a9a-b7e9-dc38017ad313
2	brazil	a050f271-0b01-4f08-9a6d-5e4bb23209f5
3	usa	677afa1b-5986-49fa-a694-d514590d11e1

Region table

	asc region	region_id	location_id
1	santa_virginia	2a748131-f5f2-4d1d-99c8-d7b4e241a68c	a050f271-0b01-4f08-9a6d-5e4bb23209f5
2	boraceia	efd65fcb-df57-433e-b848-ec2e2efbb957	a050f271-0b01-4f08-9a6d-5e4bb23209f5
3	fortuna	5ee6026b-4924-44a0-84d7-63770b7356f3	350ec32a-15f0-4a9a-b7e9-dc38017ad313
4	santa_fe	751296b0-b307-4737-a383-e4f889b85770	350ec32a-15f0-4a9a-b7e9-dc38017ad313
5	altos_de_campana	0686e569-a95b-4b97-ace4-7618aa36a934	350ec32a-15f0-4a9a-b7e9-dc38017ad313
6	chiriqui	0f46e2a3-a21c-4bfc-81e3-8af5f2cfa171	350ec32a-15f0-4a9a-b7e9-dc38017ad313
7	caribbean	fe837323-3df7-449c-8775-10286ae544fb	350ec32a-15f0-4a9a-b7e9-dc38017ad313
8	el_valle	83742e97-2811-4487-b9a4-032c7a2c856c	350ec32a-15f0-4a9a-b7e9-dc38017ad313
9	el_cope	6bf52a62-51fa-4cea-b0e9-441b426c8323	350ec32a-15f0-4a9a-b7e9-dc38017ad313
10	gamboa	5591ef77-4df8-42ef-b192-c58a597bebf4	350ec32a-15f0-4a9a-b7e9-dc38017ad313
11	pennsylvania	79362018-9bbb-4e29-91c9-81a0bf9b061d	677afa1b-5986-49fa-a694-d514590d11e1
12	vermont	ad11b0d0-64b6-4305-841a-40361259b9cb	677afa1b-5986-49fa-a694-d514590d11e1
13	new_mexico	4ce50809-088c-43c5-b26e-37fc038fd06b	677afa1b-5986-49fa-a694-d514590d11e1
14	tennessee	87c74c31-d629-4c67-97e0-07b7d2fdcd5c	677afa1b-5986-49fa-a694-d514590d11e1
15	louisiana	165e29be-fe32-4c47-b5d5-1dec2d826ec6	677afa1b-5986-49fa-a694-d514590d11e1
16	california	695634e6-2cd8-4786-bab8-368a07821dbb	677afa1b-5986-49fa-a694-d514590d11e1

To join these two table we would join the `location` table primary key (`location_id`) to the `region` table foreign key (`location_id`). In RStudio, `location_region <- inner_join(location, region, by = c("location_id"))`

	ABC location ▼	ABC region ▼
1	brazil	santa_virginia
2	brazil	boraceia
3	panama	fortuna
4	panama	santa_fe
5	panama	altos_de_campa
6	panama	chiriqui
7	panama	caribbean
8	panama	el_valle
9	panama	el_cope
10	panama	gamboa
11	usa	pennsylvania
12	usa	vermont
13	usa	new_mexico
14	usa	tennessee
15	usa	louisiana
16	usa	california

We can now see the location column populates the correct value where ever the primary key and foreign key equal each other or in other words, the location_id from both tables equal each other.



```

graph TD
    survey[survey] --- site[site]
    survey --- visit[visit]
    survey --- capture[capture]
    survey --- aural[aural]
    survey --- ves[ves]
    survey --- serdp_nearest_microbiome_mucosoma_antifungal[serdp_nearest_microbiome_mucosoma_antifungal]
    survey --- serdp_amp[serdp_amp]
    survey --- serdp_bd[serdp_bd]
    survey --- serdp_bd_genomic[serdp_bd_genomic]
    survey --- panama_bd_temp[panama_bd_temp]

    site --> location_id
    site --> region_id
    site --> site_code
    site --> elevation_m
    site --> area_sq_m
    site --> depth_m
    site --> topo
    site --> wilderness
    site --> site_comments
    site --> region_id

    visit --> date
    visit --> survey_time
    visit --> campaign
    visit --> visit_status
    visit --> visit_comments
    visit --> site_id

    capture --> capture_id
    capture --> species_capture
    capture --> time_of_capture
    capture --> capture_tx_loc
    capture --> microhabitat_type
    capture --> body_temp_c
    capture --> substrate_temp_c
    capture --> wtl_mm
    capture --> body_mass_g
    capture --> life_stage
    capture --> sex
    capture --> capture_animal_state
    capture --> photo
    capture --> photo_id
    capture --> microhabitat_moredetail
    capture --> body_and_bag_mass_g
    capture --> bag_mass_g
    capture --> marked
    capture --> capture_comments
    capture --> capture_time
    capture --> capture_utm
    capture --> capture_type
    capture --> antibody_id
    capture --> duration_min
    capture --> bag_id
    capture --> processor
    capture --> capture_mark_recapture
    capture --> microhabitat_notes
    capture --> d1_n
    capture --> d1_s
    capture --> d1_a
    capture --> d1_w
    capture --> d1_percent_cover
    capture --> densimeter_d2_num_covered
    capture --> d2_n
    capture --> d2_s
    capture --> d2_a
    capture --> d2_w
    capture --> d2_percent_cover
    capture --> depth_of_water_firm_d2_cm
    capture --> percent_vegetation_cover
    capture --> vegetation_notes
    capture --> secchi_depth_cm
    capture --> visit_id

    aural --> aural_id
    aural --> species_aural
    aural --> call_index
    aural --> aural_comments
    aural --> count
    aural --> detection_location
    aural --> microhab
    aural --> life_stage
    aural --> sex
    aural --> microhab_moredetail
    aural --> survey_id

    ves --> ves_id
    ves --> species_ves
    ves --> count
    ves --> detection_location
    ves --> microhab
    ves --> life_stage
    ves --> sex
    ves --> ves_comments
    ves --> microhab_moredetail
    ves --> visual_animal_state
    ves --> survey_id

    serdp_nearest_microbiome_mucosoma_antifungal --> mucosoma_id
    serdp_nearest_microbiome_mucosoma_antifungal --> microbiome_swab_id
    serdp_nearest_microbiome_mucosoma_antifungal --> swab_id
    serdp_nearest_microbiome_mucosoma_antifungal --> observed_features
    serdp_nearest_microbiome_mucosoma_antifungal --> faith_pd
    serdp_nearest_microbiome_mucosoma_antifungal --> shannon_entropy
    serdp_nearest_microbiome_mucosoma_antifungal --> esp_evenness
    serdp_nearest_microbiome_mucosoma_antifungal --> esp_shannon
    serdp_nearest_microbiome_mucosoma_antifungal --> all_seqs_richness
    serdp_nearest_microbiome_mucosoma_antifungal --> total_all_seqs
    serdp_nearest_microbiome_mucosoma_antifungal --> proper_total_all_seqs
    serdp_nearest_microbiome_mucosoma_antifungal --> facilitating_richness
    serdp_nearest_microbiome_mucosoma_antifungal --> total_facilitating
    serdp_nearest_microbiome_mucosoma_antifungal --> proper_total_facilitating_richness
    serdp_nearest_microbiome_mucosoma_antifungal --> anti_fungal_richness
    serdp_nearest_microbiome_mucosoma_antifungal --> total_anti_fungal
    serdp_nearest_microbiome_mucosoma_antifungal --> proper_total_anti_fungal_richness
    serdp_nearest_microbiome_mucosoma_antifungal --> tbr_richness
    serdp_nearest_microbiome_mucosoma_antifungal --> total_ttr
    serdp_nearest_microbiome_mucosoma_antifungal --> proper_total_ttr
    serdp_nearest_microbiome_mucosoma_antifungal --> corrected_viability
    serdp_nearest_microbiome_mucosoma_antifungal --> extraction_hl
    serdp_nearest_microbiome_mucosoma_antifungal --> shedding_c1
    serdp_nearest_microbiome_mucosoma_antifungal --> shedding_bd_mean
    serdp_nearest_microbiome_mucosoma_antifungal --> shedding_bd_swab
    serdp_nearest_microbiome_mucosoma_antifungal --> shedding_infected

    serdp_amp --> amp_id
    serdp_amp --> temp_tp_1368_9
    serdp_amp --> brev_tp1_1834
    serdp_amp --> brev_tp1_intensity
    serdp_amp --> brev_tpa_2582_4
    serdp_amp --> brev_tpa_intensity
    serdp_amp --> brev_tpb_2576_4
    serdp_amp --> brev_tpb_intensity
    serdp_amp --> brev_tpc_2595_5
    serdp_amp --> brev_tpc_intensity
    serdp_amp --> rana_tp_2999_5
    serdp_amp --> rana_tp_intensity
    serdp_amp --> rana_tpb_3521
    serdp_amp --> rana_tpb_intensity
    serdp_amp --> field_notes
    serdp_amp --> censor
    serdp_amp --> total_peptides_up
    serdp_amp --> water
    serdp_amp --> season
    serdp_amp --> julian_date
    serdp_amp --> peptides_up_gbw
    serdp_amp --> peptides_up_cm2
    serdp_amp --> peptides_up_mt_mucus
    serdp_amp --> anti
    serdp_amp --> gla_plate_gss_growth
    serdp_amp --> date_gla
    serdp_amp --> gla_final_reading
    serdp_amp --> percent_inhibition_18 Ug_mL
    serdp_amp --> percent_inhibition_31 Ug_mL
    serdp_amp --> percent_inhibition_63 Ug_mL
    serdp_amp --> percent_inhibition_125 Ug_mL
    serdp_amp --> percent_inhibition_250 Ug_mL
    serdp_amp --> percent_inhibition_500 Ug_mL
    serdp_amp --> slope
    serdp_amp --> tcs50
    serdp_amp --> mic
    serdp_amp --> gla_notes
    serdp_amp --> daily_arg_1
    serdp_amp --> daily_arg_2
    serdp_amp --> daily_arg_3
    serdp_amp --> daily_arg_4
    serdp_amp --> daily_arg_5
    serdp_amp --> daily_arg_6
    serdp_amp --> daily_arg_7
    serdp_amp --> daily_arg_8
    serdp_amp --> daily_arg_9
    serdp_amp --> daily_arg_10
    serdp_amp --> daily_arg_11
    serdp_amp --> daily_arg_12
    serdp_amp --> daily_arg_13
    serdp_amp --> daily_arg_14
    serdp_amp --> daily_arg_15
    serdp_amp --> daily_arg_16
    serdp_amp --> daily_arg_17
    serdp_amp --> daily_arg_18
    serdp_amp --> daily_arg_19
    serdp_amp --> daily_arg_20
    serdp_amp --> daily_arg_21
    serdp_amp --> daily_arg_22
    serdp_amp --> daily_arg_23
    serdp_amp --> daily_arg_24
    serdp_amp --> daily_arg_25
    serdp_amp --> daily_arg_26
    serdp_amp --> daily_arg_27
    serdp_amp --> daily_arg_28
    serdp_amp --> daily_arg_29
    serdp_amp --> daily_arg_30
    serdp_amp --> daily_arg_31
    serdp_amp --> daily_arg_32
    serdp_amp --> daily_arg_33
    serdp_amp --> daily_arg_34
    serdp_amp --> daily_arg_35
    serdp_amp --> daily_arg_36
    serdp_amp --> daily_arg_37
    serdp_amp --> daily_arg_38
    serdp_amp --> daily_arg_39
    serdp_amp --> daily_arg_40
    serdp_amp --> daily_arg_41
    serdp_amp --> daily_arg_42
    serdp_amp --> daily_arg_43
    serdp_amp --> daily_arg_44
    serdp_amp --> daily_arg_45
    serdp_amp --> daily_arg_46
    serdp_amp --> daily_arg_47
    serdp_amp --> daily_arg_48
    serdp_amp --> daily_arg_49
    serdp_amp --> daily_arg_50
    serdp_amp --> daily_arg_51
    serdp_amp --> daily_arg_52
    serdp_amp --> daily_arg_53
    serdp_amp --> daily_arg_54
    serdp_amp --> daily_arg_55
    serdp_amp --> daily_arg_56
    serdp_amp --> daily_arg_57
    serdp_amp --> daily_arg_58
    serdp_amp --> daily_arg_59
    serdp_amp --> daily_arg_60
    serdp_amp --> daily_arg_61
    serdp_amp --> daily_arg_62
    serdp_amp --> daily_arg_63
    serdp_amp --> daily_arg_64
    serdp_amp --> daily_arg_65
    serdp_amp --> daily_arg_66
    serdp_amp --> daily_arg_67
    serdp_amp --> daily_arg_68
    serdp_amp --> daily_arg_69
    serdp_amp --> daily_arg_70
    serdp_amp --> daily_arg_71
    serdp_amp --> daily_arg_72
    serdp_amp --> daily_arg_73
    serdp_amp --> daily_arg_74
    serdp_amp --> daily_arg_75
    serdp_amp --> daily_arg_76
    serdp_amp --> daily_arg_77
    serdp_amp --> daily_arg_78
    serdp_amp --> daily_arg_79
    serdp_amp --> daily_arg_80
    serdp_amp --> daily_arg_81
    serdp_amp --> daily_arg_82
    serdp_amp --> daily_arg_83
    serdp_amp --> daily_arg_84
    serdp_amp --> daily_arg_85
    serdp_amp --> daily_arg_86
    serdp_amp --> daily_arg_87
    serdp_amp --> daily_arg_88
    serdp_amp --> daily_arg_89
    serdp_amp --> daily_arg_90
    serdp_amp --> daily_arg_91
    serdp_amp --> daily_arg_92
    serdp_amp --> daily_arg_93
    serdp_amp --> daily_arg_94
    serdp_amp --> daily_arg_95
    serdp_amp --> daily_arg_96
    serdp_amp --> daily_arg_97
    serdp_amp --> daily_arg_98
    serdp_amp --> daily_arg_99
    serdp_amp --> daily_arg_100
    serdp_amp --> daily_arg_101
    serdp_amp --> daily_arg_102
    serdp_amp --> daily_arg_103
    serdp_amp --> daily_arg_104
    serdp_amp --> daily_arg_105
    serdp_amp --> daily_arg_106
    serdp_amp --&gt
```

How to Connect and Interact with the RIBBITR Data Base

Necessary Library's

```
library(RPostgres) # The database driver
library(DBI)        # Functions needed to interact with the database
library(rstudioapi) # Package that asked for credentials

# The method below for calling in packages does the same as using `library`
# but will download any packages that are not initially downloaded. This is a
# cleaner method and more user
# friendly for sharing scripts between colleagues

if (!require(librarian)){
  install.packages("librarian")
  library(librarian)
}
# librarian downloads, if not already downloaded, and reads in needed packages
librarian::shelf(RPostgres, DBI, rstudioapi)
```

Connecting

To connect to the database you will use `dbConnect` function from the `DBI` package. You will need the

- database driver: `drv = dbDriver("Postgres")`,
- the database name: `dbname = "ribbitr"`,
- the host: `host = "ribbitr.c6p56tuocn5n.us-west-1.rds.amazonaws.com"`,
- port: `port = "5432"`,
- your username: `user = "<insert_username>"`
- and password: `password = "<insert_password>"`

I strongly recommend not storing `user` and `password`, or any database info with in your script. A secure method for storing this info is in your `.Renviron` file that is stored locally on your device and will never be pushed to github. To edit your `.Renviron` file run this in the console: `usethis::edit_r_environ()` and the file will open. Populate the file based on the sample below, hit save, and restart your R session. You can then use `Sys.getenv()` to call on those variables stored locally on your device. Another method is to use the `rstudioapi` package



with the `askForPassword()` function. Replace `Sys.getenv()` with `askForPassword()`. A log in box will appear after running the `dbConnect`.

Once you have connected to the database it is important to set `search_path` to the appropriate database schema of interest. This is identical to setting your working directory but for databases. To set search path we will use `dbExecute` in the DBI package. You will tell the `dbExecute` the connection and the SQL statement that points to the schema. Note, you must use single quotes around the schema of choice `dbExecute(conn = ribbitr_connection, "set search_path = 'survey_data'")`

Sample .Renviron file and Database Connection

```
# 1) run this in the console: "usethis::edit_r_environ()"

# 2) copy the sample below and input your own username and password.

# 3) Save and restart your r sessio. Click "session" at the top and "Restart R".
# sample .Renviron file #
dbname = "ribbitr"
host = "ribbitr.c6p56tuocn5n.us-west-1.rds.amazonaws.com"
port = "5432"
user = "username" # you will input your own username
password = "password" # you will input your own password

# connecting to RIBBITR Database
ribbitr_connection <- dbConnect(drv = dbDriver("Postgres"),
                                dbname = "ribbitr",
                                host = "ribbitr.c6p56tuocn5n.us
                                      -west-1.rds.amazonaws.com",
                                port = "5432",
                                user = Sys.getenv("user"),
                                password = Sys.getenv("password"))

# Same method for connecting to RIBBITR databse but will give feedback,
# I recommend copying this

tryCatch({
```

```
print("Connecting to Database...")

ribbitr_connection <- dbConnect(drv = dbDriver("Postgres"),
                                dbname = "ribbitr",
                                host = "ribbitr.c6p56tuocn5n.us-
                                        west-1.rds.amazonaws.com",
                                port = "5432",
                                user = askForPassword("user"),
                                password = askForPassword("password"))

print("Database Connected!")
},

error=function(cond) {
    print("Unable to connect to Database.")
})

# setting your search path
dbExecute(conn = ribbitr_connection,
          statement = "set search_path = 'survey_data'")
```

Interacting

Now you should be connected to the `ribbitr` database with in RStudio and working with in the `survey_data` schema. Interacting with the database through R will involve learning some SQL. Don't stress, SQL is a very intuitive and straight forward language. A more thorough SQL tutorial will follow and please do not hesitate to reach out for help with developing queries. I will now show two methods in how one can query the database.

dbGetQuery

Using `dbGetQuery` in the DBI package we can feed the function our database connection `conn = ribbitr_connection` and then the SQL `statement =`. Note, the `*` symbol in SQL is equivalent as saying "everything" and in R that is `.`

```
# location table only

# human speak: "select everthing from the location table"
```



```
sql_statement <- "select * from location"

location <- dbGetQuery(conn = ribbitr_connection,
                      statement = sql_statement)

# location and region table joined

# note, if you want to select specific columns from a table you must start
# with table alias, that being either "l" for location or "r" for region.
sql_statement <- "select l.location, r.region
                  from location l
                  join region r on l.location_id = r.location_id"

location_region <- dbGetQuery(conn = ribbitr_connection,
                             statement = sql_statement)

# picking columns of interest from each table and join all tables together.
# Try to always select specific columns of interest from each table.

sql_statement <- "select l.location, r.region, s.site, s.utme, s.utmn, s.utm_zone,
                        v.date, v.survey_time, su.start_time, su.end_time, su.duration_minutes,
                        c.species_capture, c.time_of_capture, c.body_temp_c, c.svl_mm, c.sex
                  from location l
                  join region r on l.location_id = r.location_id
                  join site s on r.region_id = s.region_id
                  join visit v on s.site_id = v.site_id
                  join survey su on v.visit_id = su.visit_id
                  join capture c on su.survey_id = c.survey_id;"

capture_data <- dbGetQuery(conn = ribbitr_connection,
                          statement = sql_statement)
```

Entire SQL Code Chunk

If you are using a Quarto or RMarkdown document with code chunks you can specify in the code chunk that you will be using SQL, the database connection, and what you want the output variable to be.

```
select *
from location

select l.location, r.region, s.site, s.utme, s.utmn, s.utm_zone,
v.date, v.survey_time, su.start_time, su.end_time, su.duration_minutes,
c.species_capture, c.time_of_capture, c.body_temp_c, c.svl_mm, c.sex

from location l

join region r on l.location_id = r.location_id
join site s on r.region_id = s.region_id
join visit v on s.site_id = v.site_id
join survey su on v.visit_id = su.visit_id
join capture c on su.survey_id = c.survey_id;
```