



# R y SQL

## Gestión de Datos

---

Maximiliano Arancibia

Educación Profesional - Escuela de Ingeniería

# Tabla de Contenidos

---

SQL en R

Manipulación de Datos en R



SQL en R

Manipulación de Datos en R



Para leer bases de datos SQL usaremos DBI (Database Interface), además necesitamos un package especial dependiendo de la base de datos que deseamos leer:

- RSQLite
- Rpostgres
- RMariaDB
- bigrquery
- Otros

Estos packages se encargan de la conexión con las bases de datos pero desde el punto de vista de R son (casi) idénticas.



Las siguiente funcionalidades se admiten en DBI

- Conectar a DB
- Funciones básicas
- Cargar datos a SQL
- Consultas a DB
- Consultas por lotes
- Consultas parametrizadas



Necesitamos entender las siguientes funciones:

- dbConnect - dbDisconnect
- dbListTables - dbListFields - dbReadTable
- dbWriteTable
- dbGetQuery - dbExecute
- dbSendQuery - dbHasCompleted - dbFetch - dbClearResult
- dbBind



En nuestro caso usaremos postgres asi que debemos instalar e importar DBI y Rpostgres.

De ser necesario en nuestro sistema operativo demos instalar postgres, inicializarlo, crear un superusuario y crear una base de datos (veremos un ejemplo en colab).



## Conectar a DB

Para conectar a una base de datos usamos el comando `dbConnect`:

```
con <- dbConnect(RPostgres::Postgres(),  
                 dbname = 'DATABASE_NAME',  
                 user = 'post',  
                 password = 'PASSWORD')
```

Con esta misma función podemos conectarnos a una base de datos remota. Para desconectarnos simplemente usamos el comando:

```
dbDisconnect()
```





Algunas funciones básicas incluidas en DBI:

- *dbListTables(con)*: entrega la lista de tablas en la base de datos a la cual estamos conectados.
- *dbListFields(con, table\_name)*: entrega la lista de atributos en la tabla 'table\_name'
- *dbReadTable(con, table\_name)*: entrega la tabla de nombre 'table\_name'



Una de las ventajas de usar R y SQL es que se pueden intercambiar bases de datos entre ambos lenguajes. En particular para cargar una tabla usamos:

```
dbWriteTable(con, "table_name", R_dataframe)
```

En este caso tenemos que `table_name` es el nombre de la tabla y `R_dataframe` es un dataframe en R.



Podemos también agregar más datos a una tabla mediante el comando:

```
dbAppendTable(con, "table_name", R_dataframe)
```



Para realizar consultas a una base de datos usamos los comandos:

- *dbGetQuery(con, <sql query>)*: cuando esperamos que nuestro output sea un dataframe de R.
- *dbExecute(con, <sql query>)*: cuando no esperamos un output, nos entrega la cantidad de filas modificadas con el comando.

Acá <sql query> representa cualquier consulta a la base de datos, ejemplo: 'SELECT \* FROM Peliculas'



En un proyecto usual de machine learning podríamos necesitar almacenar, limpiar y extraer los datos de nuestro RBDMS a R para entrenar un modelo.

Si realizamos una consulta al motor de bases de datos muchas veces el tamaño de los datos sigue siendo poco manejable por R. El problema es que al cargar los datos de SQL a R pasamos de tenerlos en disco duro a memoria RAM.



Soluciones a este problema:

- Usar una fracción de la base de datos
- Comprar más memoria RAM
- Utilizar un consultas por lotes (batches).



Las consultas por lotes nos permiten realizar una consulta (posiblemente muy compleja) para luego extraer solo una parte de la consulta. De este modo podemos evitar quedarnos sin memoria RAM.

Para esto necesitamos principalmente:

- `query <- dbSendQuery(con, <sql query>)`: crea un 'puntero' al resultado de la consulta a la base de datos, acá lo guardamos en la variable `query`
- `dbFetch(query, n)`: extrae  $n$  tuplas de la `query`. En este caso `query` es un puntero a un resultado de una consulta.



Otras funciones que son de utilidad para las consultas por lotes:

- `dbHasCompleted(query)`: entrega Verdadero si ya se han extraído todas las tuplas de la consulta `query` y Falso si no.
- `dbClearResult(query)`: termina y libera todos los recursos asociados a la consulta `query`.





Al estar trabajando en un entorno de R surge la posibilidad de realizar múltiples consultas a través de un loop. De gran utilidad puede resultar parametrizar las consultas, es decir dejar incógnitas dentro de la consulta e ir completandolas con valores existentes en el entorno de R. La función que necesitamos es:

- `dbBind(query, lista)`: reemplaza las incógnitas al interior de la consulta query por los valores de la lista.



El procedimiento viene de esta forma:

- Crear consulta con incógnitas \$1, \$2. ...

```
query <- dbSendQuery(con, "select * from  
Peliculas where Calificacion<$1")
```

- Transformar la query usando dbBind.

```
dbBind(query, list(8))
```

- Usar dbFetch para extraer las tuplas

```
dbFetch(query)
```



SQL en R

Manipulación de Datos en R



## Leer datos directo en R

- Rápido.
- Ejecución de comandos lento.
- Datos locales (o en archivo online con los permisos adecuados)
- Posibles problemas con memoria.



## Leer datos con DBMS

- Proceso inicial más lento.
- Datos y ejecución local o en server.
- Operaciones a realizar tienen más restricciones.



Funciones nuevas importantes:

- `arrange()`: reordena según atributos de columnas
- `mutate()`: crea nuevas variables a partir de transformaciones de las columnas
- `summarise()`: contrae muchos valores en un solo resumen

**ojo:** todas estas funciones tienen de input un DF y devuelven DF por lo que son compatibles con el operador pipe (`%>%`).



En R también encontramos el comando 'group\_by' como en SQL su sintaxis es similar y permiten la aplicación de una gran cantidad de funciones complejas.

Todas las anteriormente vistas son compatibles con group\_by() y ungroup(), agrupar o desagrupar según alguna variable (o categoría).



Algunas funciones importantes para los grupos y summarise:

- Conteo y suma (`n()`, `sum()`): cuenta el tamaño del grupo, incluidos datos faltantes
- Posición (`first()`, `last()`, `nth(2)`): primer, ultimo y segundo valor del grupo
- Centralidad y dispersión (`mean()`, `sd()`, `mad()`): momentos y medidas importantes para el grupo.
- Rango (`min()`, `quantile(0.75)` `max()`): rango de valores y descripciones para el grupo

También dentro de los grupos podemos generar una columna nueva en base a un orden asignando una columna a la función `row_number()`.

