Bases de datos y cómo trabajar con ellas en R

Natalia da Silva y Oscar Montañés

FCEA-UdelaR
Jornadas SUE 2022

natalia.dasilva@fcea.edu.uy - natydasilva.com - @pacocuak
omontanes@gmail.com-

Octubre 2022

Quienes somos

- Oscar Montañés: Ingeniero en Computación, Trabaja como gerente de producto e ingeniero de datos en HG.SA. Integrante del Proyecto ANII-IESTA-CEIBAL
- Natalia da Silva: Estadística, docente/investigadora(IESTA-UDELAR)

Veremos

- Intro a Bases de datos, entender BD y su estructura
- Realizar consultas bśicas SQL
- Cómo conectarnos a una base de datos desde R
- Cómo hacer consultas SQL desde R con `DBI` y `dplyr`

- Cuando los datos son de **grandes volumenes**, es imposible guardarlos en formatos de archivo tradicionales
- Los archivos tradicionales no caben en dispositivos con almacenamiento limitado y no se pueden compartir fácilmente entre colaboradores.
- En estos casos es recomendable almacenar datos en bases de datos para una gestión de datos más escalable y confiable.
- Permite la concurrencia y en muchos casos la referencia de los datos.

Bases de datos

- Una base de datos almacena datos
- Los datos son almacenados y organizados en objetos que llamamos **tablas**
- Bases de datos relacionales, definen relaciones entre tablas de datos en una base de datos.

Ventajas de las bases de datos

- Se pueden almacenar grandes volúmenes de información
- El almacenamiento es más seguro que guardarlos en otros formatos
- Muchos usuarios pueden hacer consultas al mismo tiempo a la base de datos
- Los datos en esa base de datos no se modifican cuando se hacen las consultas.

Conceptos

- SQL : Es el lenguaje de codificación, basado en el álgebra relacional de conjuntos y tiene un alto porcentaje en común en todos los motores
- Motor: Es el software que instalamos en un sistema operativo. Una máquina/servidor puede tener varios motores (distintas versiones o distintos fabricantes) aunque no es buena práctica.
- Base de datos: Conjunto de tablas, procedimientos y funciones que se alojan en un motor. Un motor puede tener varias bases de datos

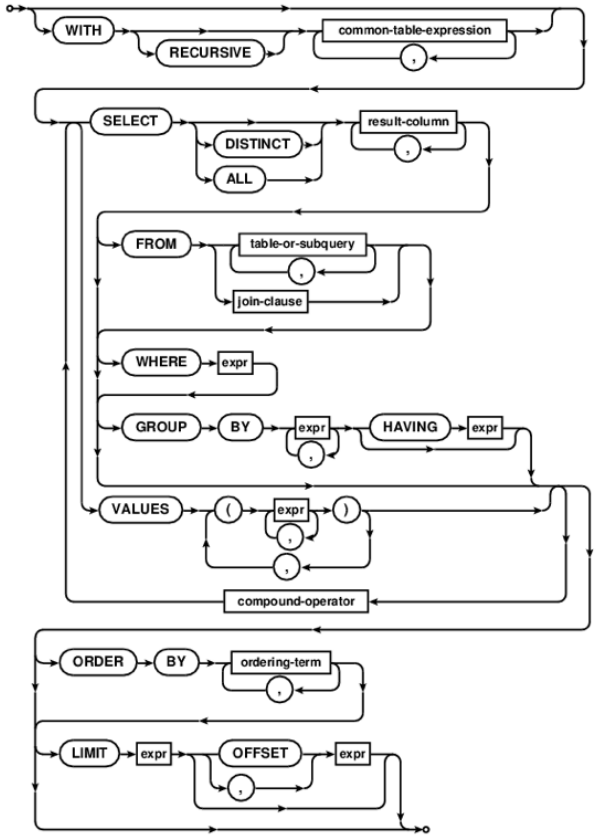
SQL, Structured Query Language

SQL es es el lenguaje más ampliamente usado para bases de datos 2 sublenguajes:

- Data Definition Languaje - lenguaje de definición de datos): las sentencias DDL son aquellas utilizadas para la creación de una base de datos y todos sus componentes: tablas, índices, vistas, procedimientos almacenados, etc. Tipicamente (CREATE, DROP y ALTER TABLE)
- DML:las sentencias DML son aquellas utilizadas para insertar, borrar, modificar y consultar los datos de una base de datos.

SELECT

- SELECCIONAR Columnas de una tabla o subconsulta
- JOINEAR con otras tablas o subconsultas bajo una condicion
- DONDE (WHERE) se cumplan condiciones
- GROUP BY valores



Orden

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

Consultas básicas I

- Obtener todas las columnas de una tabla
  `SELECT * FROM airbnb_listings;`
- Devuelve la columna city de una tabla
  `SELECT city FROM airbnb_listings;`
- Obtiene las columnas city a year_listed de una tabla `SELECT city, year_listed FROM airbnb_listings;`
- Obtine la lista de id, city, ordenado por el number_of_rooms en forma ascendente `SELECT city, year_listed FROM airbnb_listings ORDER BY number_of_rooms ASC;`

Consutas básicas II

- Obtine la lista de id, city, ordenado por number_of_rooms en forma descendente
  ```
  SELECT city, year_listed
  FROM airbnb_listings ORDER BY number_of_rooms DESC;
  ```
- Obtiene las primeras 5 filas de airbnb_listings
  ```
  SELECT * FROM airbnb_lisitings LIMIT 5;
  ```
- Obtiene una única lista de ciudades donde hay opciones de alojamiento
  ```
  SELECT DISTINCT city FROM airbnb_lisitings;
  ```

Filtrando

- Otiene todas las opciones de alojamiento donde el number_of_rooms mayor i igual a 3

```
SELECT*
FROM airbnb_listings
WHERE number_of_rooms >= 3;
```

- Filtra columnas en un rango, todas los que tienen entre 3 y 6 cuartos

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms BETWEEN 3 AND 6;
```

- Filtra columnas en un rango, obtiene todas las que tienen entre 3 y 6 u 8 cuartos SELECT *

```
FROM airbnb_listings
WHERE number_of_rooms BETWEEN 3 AND 6 OR 8 (otra manera
number_of_rooms IN (3,4,5,6,8);)
```

Agregaciones I

Agregaciones simples

- Obtiene el total de cuartos disponibles para todos los alojamientos disponibles

```
SELECT SUM(number_of_rooms)
FROM airbnb_listings;
```

- Obtiene el número promedio de cuartos de la lista entreo todos los alojamientos

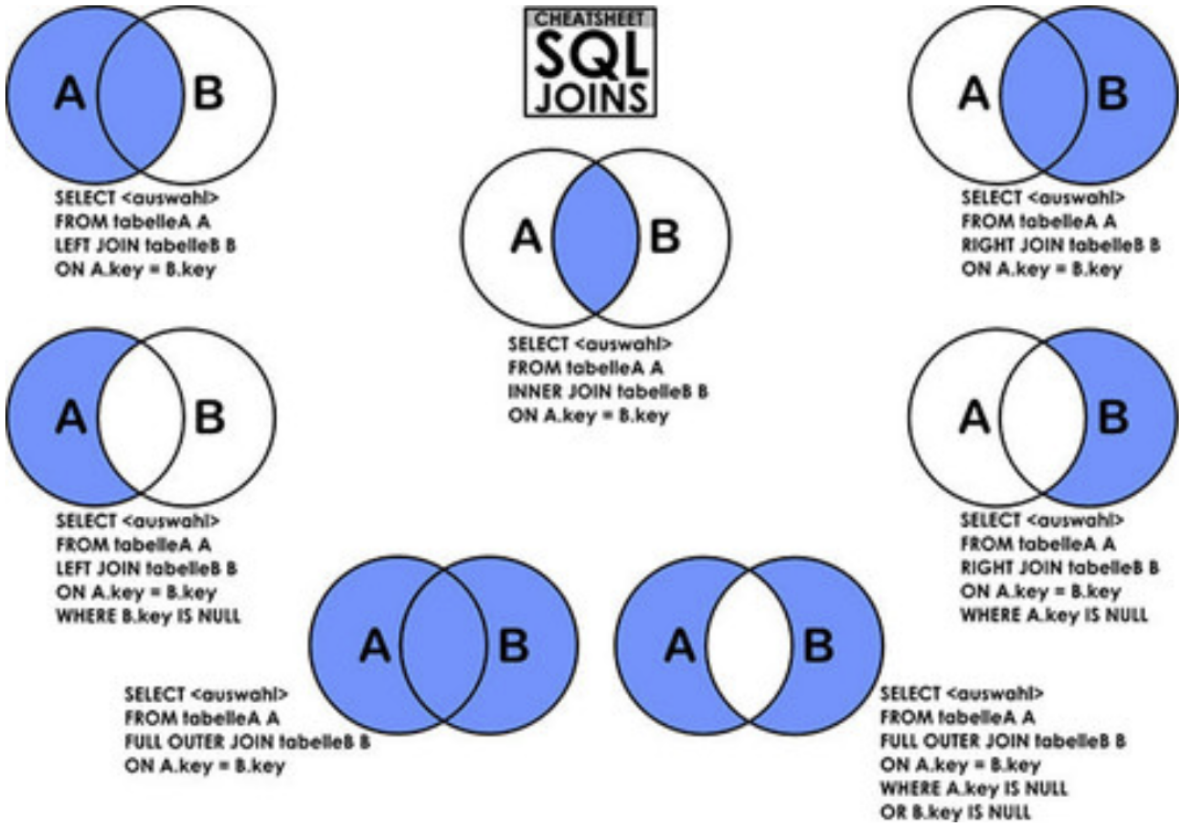
```
SELECT AVG(number_of_rooms)
FROM airbnb_listings;
```

Agregaciones II

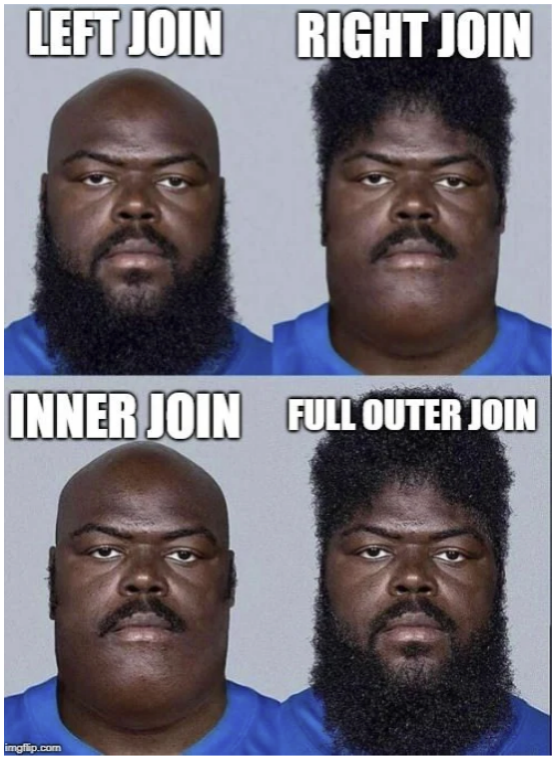
- Obtiene el nḿero de cuartos para cada país
```
SELECT country, SUM(number_of_rooms)
FROM airbnb_listings
GROUP BY country;
```
- Obtiene el número promedio de cuartos para cada país
```
SELECT country, AVG(number_of_rooms)
FROM airbnb_listings GROUP BY country;
```
- Obtiene todos los años donde hay más de 100 alojamientos disponibles por año
```
SELECT year_listed
FROM airbnb_listings
GROUP BY year_listed HAVING COUNT(id) > 100;
```

Joint

Joins


CHEATSHEET
SQL JOINS
A
B
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
SELECT <auswahl>
FROM tabelleA A
INNER JOIN tabelleB B
ON A.key = B.key
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
SELECT <auswahl>
FROM tabelleA A
LEFT JOIN tabelleB B
ON A.key = B.key
WHERE B.key IS NULL
SELECT <auswahl>
FROM tabelleA A
RIGHT JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
SELECT <auswahl>
FROM tabelleA A
FULL OUTER JOIN tabelleB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL

Joint

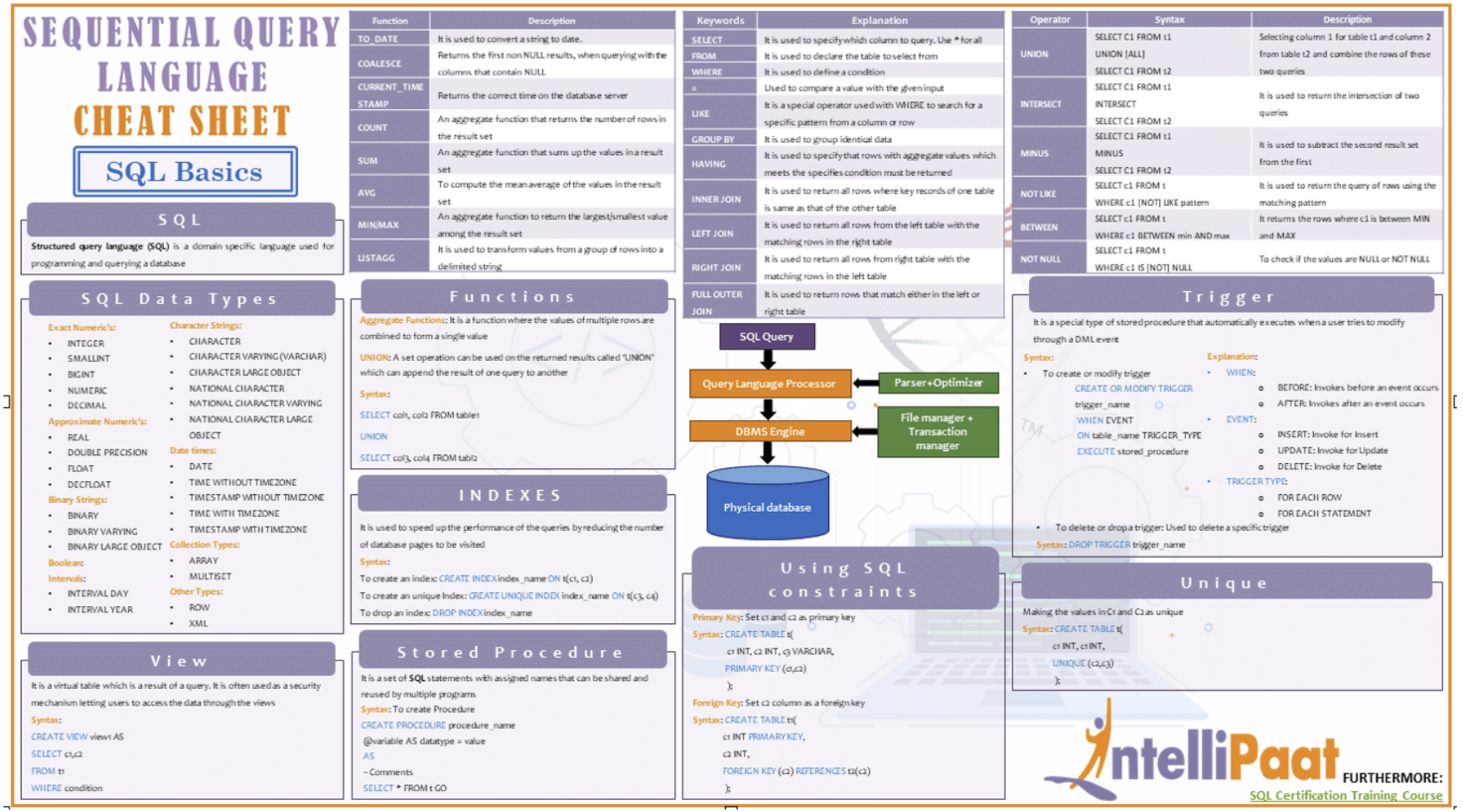


```
SELECT att1,att2
FROM tabla 1
LEFT JOIN tabla2 ON a1= b2 AND
c1=d2
WHERE cosa1
```

```
SELECT att1,att2
FROM tabla 1
LEFT JOIN tabla2 ON a1= b2
WHERE cosa1 AND c1=d2
```

OJO NO SON LO MISMO

SQL Cheat Sheet



SEQUENTIAL QUERY LANGUAGE CHEAT SHEET

SQL Basics

SQL

Structured query language (SQL) is a domain specific language used for programming and querying a database

SQL Data Types

Exact Numeric's:
- INTEGER
- SMALLINT
- BIGINT
- NUMERIC
- DECIMAL

Approximate Numeric's:
- REAL
- DOUBLE PRECISION
- FLOAT
- DECFLOAT

Binary Strings:
- BINARY
- BINARY VARYING
- BINARY LARGE OBJECT

Boolean:

Intervals:
- INTERVAL DAY
- INTERVAL YEAR

Character Strings:
- CHARACTER
- CHARACTER VARYING (VARCHAR)
- CHARACTER LARGE OBJECT
- NATIONAL CHARACTER
- NATIONAL CHARACTER VARYING
- NATIONAL CHARACTER LARGE OBJECT

Date times:
- DATE
- TIME WITHOUT TIMEZONE
- TIMESTAMP WITHOUT TIMEZONE
- TIME WITH TIMEZONE
- TIMESTAMP WITH TIMEZONE

Collection Types:
- ARRAY
- MULTISET

Other Types:
- ROW
- XML

View

It is a virtual table which is a result of a query. It is often used as a security mechanism letting users to access the data through the views

Syntax:

```
CREATE VIEW view1 AS
SELECT c1,c2
FROM t1
WHERE condition
```

Function	Description
TO_DATE	It is used to convert a string to date.
COALESCE	Returns the first non NULL results, when querying with the columns that contain NULL
CURRENT_TIMESTAMP	Returns the correct time on the database server
COUNT	An aggregate function that returns the number of rows in the result set
SUM	An aggregate function that sums up the values in a result set
AVG	To compute the mean average of the values in the result set
MIN/MAX	An aggregate function to return the largest/smallest value among the result set
LISTAGG	It is used to transform values from a group of rows into a delimited string

Functions

Aggregate Functions: It is a function where the values of multiple rows are combined to form a single value

UNION: A set operation can be used on the returned results called "UNION" which can append the result of one query to another

Syntax:

```
SELECT col1, col2 FROM table1
UNION
SELECT col3, col4 FROM table2
```

INDEXES

It is used to speed up the performance of the queries by reducing the number of database pages to be visited

Syntax:

To create an index: `CREATE INDEX index_name ON t(c1, c2)`

To create an unique Index: `CREATE UNIQUE INDEX index_name ON t(c3, c4)`

To drop an index: `DROP INDEX index_name`

Stored Procedure

It is a set of **SQL** statements with assigned names that can be shared and reused by multiple programs

Syntax: To create Procedure

```
CREATE PROCEDURE procedure_name
@variable AS datatype = value
AS
-- Comments
SELECT * FROM t GO
```

Keywords	Explanation
SELECT	It is used to specify which column to query. Use * for all
FROM	It is used to declare the table to select from
WHERE	It is used to define a condition
=	Used to compare a value with the given input
LIKE	It is a special operator used with WHERE to search for a specific pattern from a column or row
GROUP BY	It is used to group identical data
HAVING	It is used to specify that rows with aggregate values which meets the specifies condition must be returned
INNER JOIN	It is used to return all rows where key records of one table is same as that of the other table
LEFT JOIN	It is used to return all rows from the left table with the matching rows in the right table
RIGHT JOIN	It is used to return all rows from right table with the matching rows in the left table
FULL OUTER JOIN	It is used to return rows that match either in the left or right table

Using SQL constraints

Primary Key: Set c1 and c2 as primary key

```
Syntax: CREATE TABLE t(
        c1 INT, c2 INT, c3 VARCHAR,
        PRIMARY KEY (c1,c2)
        );
```

Foreign Key: Set c2 column as a foreign key

```
Syntax: CREATE TABLE t1(
        c1 INT PRIMARY KEY,
        c2 INT,
        FOREIGN KEY (c2) REFERENCES t2(c2)
        );
```

Operator	Syntax	Description
UNION	SELECT C1 FROM t1 UNION [ALL] SELECT C1 FROM t2	Selecting column 1 for table t1 and column 2 from table t2 and combine the rows of these two queries
INTERSECT	SELECT C1 FROM t1 INTERSECT SELECT C1 FROM t2	It is used to return the intersection of two queries
MINUS	SELECT C1 FROM t1 MINUS SELECT C1 FROM t2	It is used to subtract the second result set from the first
NOT LIKE	SELECT c1 FROM t WHERE c1 [NOT] LIKE pattern	It is used to return the query of rows using the matching pattern
BETWEEN	SELECT c1 FROM t WHERE c1 BETWEEN min AND max	It returns the rows where c1 is between MIN and MAX
NOT NULL	SELECT c1 FROM t WHERE c1 IS [NOT] NULL	To check if the values are NULL or NOT NULL

Trigger

It is a special type of stored procedure that automatically executes when a user tries to modify through a DML event

Syntax:
- To create or modify trigger

```
CREATE OR MODIFY TRIGGER
trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure
```

Explanation:
- WHEN:
  - BEFORE: Invokes before an event occurs
  - AFTER: Invokes after an event occurs
- EVENT:
  - INSERT: Invoke for Insert
  - UPDATE: Invoke for Update
  - DELETE: Invoke for Delete
- TRIGGER TYPE:
  - FOR EACH ROW
  - FOR EACH STATEMENT

- To delete or drop a trigger: Used to delete a specific trigger

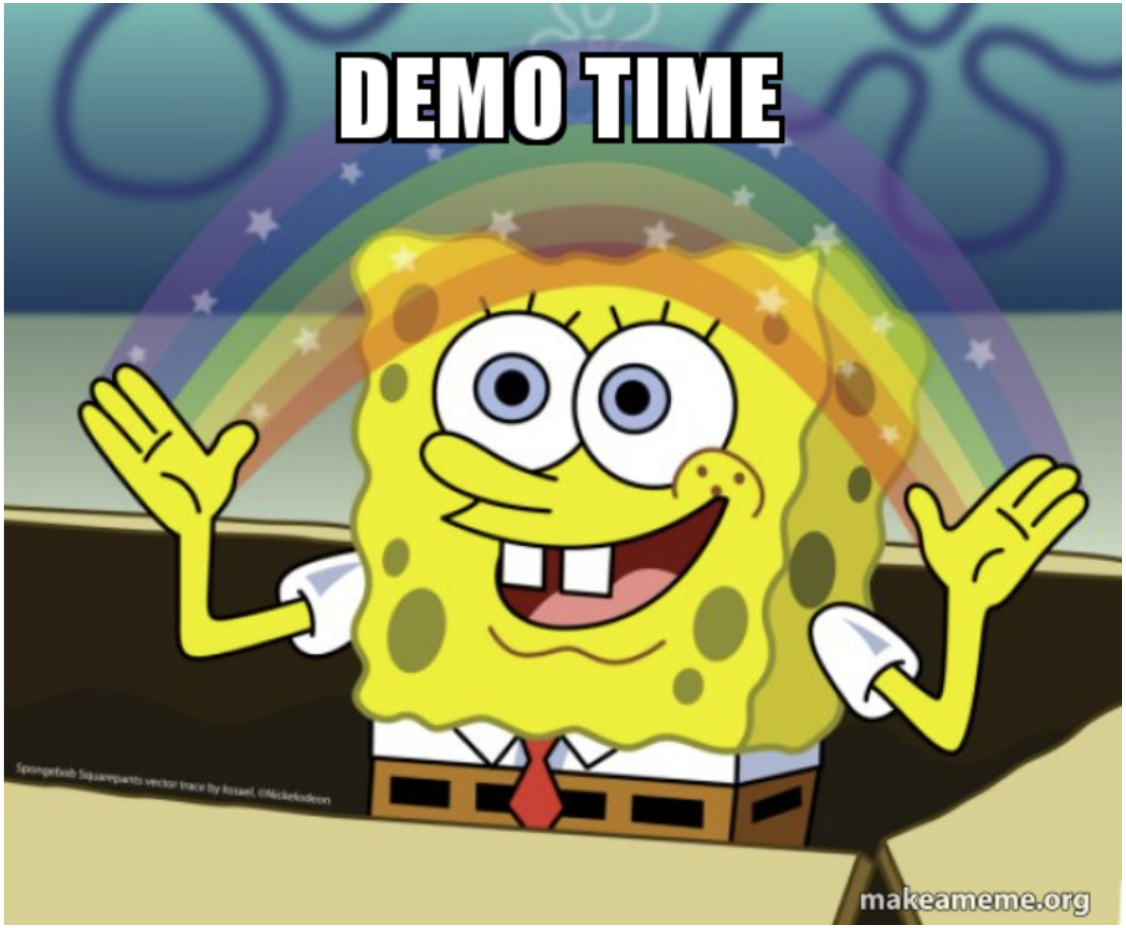
Syntax: `DROP TRIGGER trigger_name`

Unique

Making the values in C1 and C2 as unique

```
Syntax: CREATE TABLE t(
        c1 INT, c1 INT,
        UNIQUE (c2,c3)
        );
```

IntelliPaat FURTHERMORE:
SQL Certification Training Course


DEMO TIME
makeameme.org

CRAN Task View: Databases with R

- Para tener una idea general de algunos de los paquetes para bases de datos explorar Task View de R para base de datos: `https://cran.r-project.org/web/views/Databases.html`
- Se han desarrollado muchos paquetes en R que permiten de forma sensilla usar bases de datos en R .

Paquetes para conectarnos a la base de datos

Dependiendo del tipo de base de datos a la que nos queremos conectar el paquete a usar

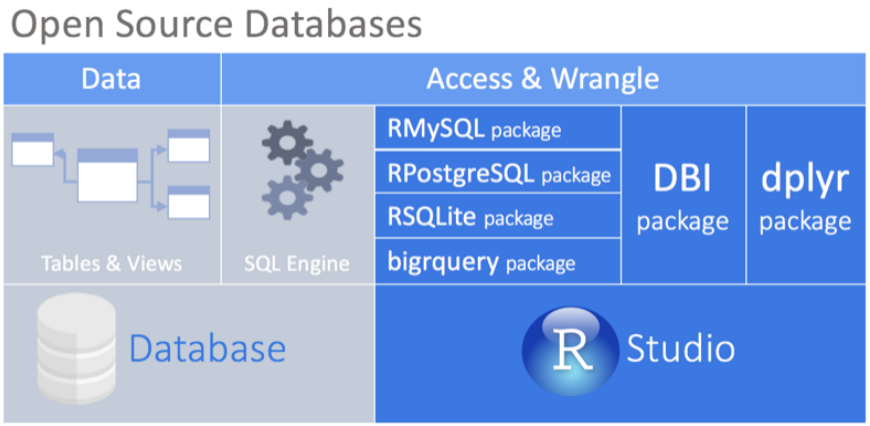
- `RMySQL` para MySQL
- `RPostgresSQL` para PostgresSQL
- `ROracle` para Oracle
- ....

Para acceder y manipular los datos en R

Hay varias formas de hacer consultas de datos con R, las tres más comunes son:

- Usando el paquete `DBI`
- Usando el paquete `dplyr`
- Usando Notebooks en R

No hay un opción mucho mejor que otra, cada una tiene sus ventajas. Nos concentraremos en `DBI` y `dplyr`



- `DBI` intermediario entre paquetes que permite la conectividad a la base de datos. (DBI como interfaz y PostgresSQL, MySQL, etc la implementación)
- `dplyr` depende del paquete `DBI` para comunicarse con la base
- Hay paquetes que premiten la conexión directa entre una base de datos abierta y R. Disponibles para las siguientes bases de datos: MySQL, SQLite, PostgreSQL, and BigQuery

DBI, DataBase Interface

- el paquete `DBI` brinda una interfase consitente entre R y Sistemas de Gestión de Bases de Datos (DBMS)
- `DBI` soporta aprox 30 DBMS inclyendo:
  - MySQL, con el paquete `RMySQL`
  - MariaDB, con el paquete `RMariaDB`
  - Postgres, con el paquete `RPostgres`
  - SQLite, con el paquete `RSQLite`

La lista completa en: `https://github.com/r-dbi/backends`

DBI

Las funcionalidades para cada DBMS son:

- Manejar una conexión a una base de datos
- Listar las tablas en una base de datos
- Listar el nombre de las columnas en una tabla
- Leer una tabla en un data frame
- ....

Características más avanzadas como consultas parametrizadas, transacciones ver `vignette("DBI-advanced", package = "DBI")`.

Algunas funciones en DBI

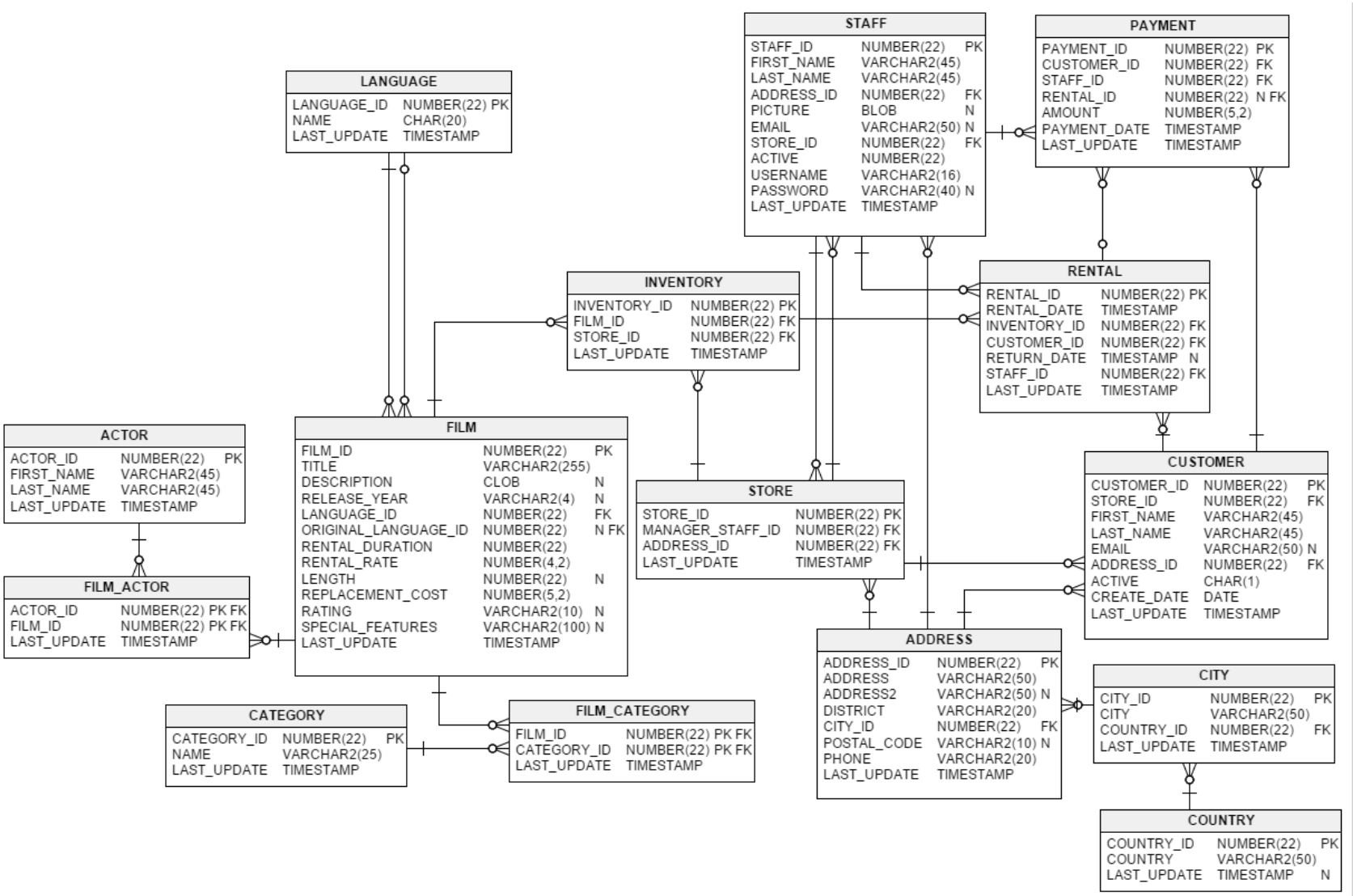
- `dbConnect`: Crea una conexión a DBMS
- `dbListTables`: Lista tablas remotas
- `dbCreateTable`: Crea una tabla en la base de datos
- `dbDisconnect`: Desconecta la base
- `dbGetInfo` : Metadata de DBMS
- `dbGetQuery` : Envia la consulta muestra resultados y limpia
- `dbReadTable`: Copia data frames de tablas de la base de datos
- `sqlData`: Convierte un data frame en un formato adecuado para subirlo a la base de datos SQL

Veremos los ejemplos de la vignette del paquete `DBI`

¿Cómo conectarnos a una base de datos usando DBI?

Veremos los ejemplos de la vignette del paquete DBI

- Estableceremos una conexión a la base de datos Pagila (Sakila para postgres) hospedada en el repositorio de bases de datos relacionales en: https://github.com/devrimgunduz/pagila
- La base de datos representa un negocio ficticio de alquiler de películas e incluye tablas que describen películas, actores, clientes, tiendas, etc.
- Pueden explorar info adicional en https://dataedo.com/samples/html/Pagila/doc/Pagila_10/home.html
- Veremos todas las tablas en la base de datos
- Finalmente cerraremos la conexión


STAFF
STAFF_ID NUMBER(22) PK
FIRST_NAME VARCHAR2(45)
LAST_NAME VARCHAR2(45)
ADDRESS_ID NUMBER(22) FK
PICTURE BLOB N
EMAIL VARCHAR2(50) N
STORE_ID NUMBER(22) FK
ACTIVE NUMBER(22)
USERNAME VARCHAR2(16)
PASSWORD VARCHAR2(40) N
LAST_UPDATE TIMESTAMP
PAYMENT
PAYMENT_ID NUMBER(22) PK
CUSTOMER_ID NUMBER(22) FK
STAFF_ID NUMBER(22) FK
RENTAL_ID NUMBER(22) N FK
AMOUNT NUMBER(5,2)
PAYMENT_DATE TIMESTAMP
LAST_UPDATE TIMESTAMP
LANGUAGE
LANGUAGE_ID NUMBER(22) PK
NAME CHAR(20)
LAST_UPDATE TIMESTAMP
INVENTORY
INVENTORY_ID NUMBER(22) PK
FILM_ID NUMBER(22) FK
STORE_ID NUMBER(22) FK
LAST_UPDATE TIMESTAMP
RENTAL
RENTAL_ID NUMBER(22) PK
RENTAL_DATE TIMESTAMP
INVENTORY_ID NUMBER(22) FK
CUSTOMER_ID NUMBER(22) FK
RETURN_DATE TIMESTAMP N
STAFF_ID NUMBER(22) FK
LAST_UPDATE TIMESTAMP
ACTOR
ACTOR_ID NUMBER(22) PK
FIRST_NAME VARCHAR2(45)
LAST_NAME VARCHAR2(45)
LAST_UPDATE TIMESTAMP
FILM
FILM_ID NUMBER(22) PK
TITLE VARCHAR2(255)
DESCRIPTION CLOB N
RELEASE_YEAR VARCHAR2(4) N
LANGUAGE_ID NUMBER(22) FK
ORIGINAL_LANGUAGE_ID NUMBER(22) N FK
RENTAL_DURATION NUMBER(22)
RENTAL_RATE NUMBER(4,2)
LENGTH NUMBER(22) N
REPLACEMENT_COST NUMBER(5,2)
RATING VARCHAR2(10) N
SPECIAL_FEATURES VARCHAR2(100) N
LAST_UPDATE TIMESTAMP
STORE
STORE_ID NUMBER(22) PK
MANAGER_STAFF_ID NUMBER(22) FK
ADDRESS_ID NUMBER(22) FK
LAST_UPDATE TIMESTAMP
CUSTOMER
CUSTOMER_ID NUMBER(22) PK
STORE_ID NUMBER(22) FK
FIRST_NAME VARCHAR2(45)
LAST_NAME VARCHAR2(45)
EMAIL VARCHAR2(50) N
ADDRESS_ID NUMBER(22) FK
ACTIVE CHAR(1)
CREATE_DATE DATE
LAST_UPDATE TIMESTAMP
FILM_ACTOR
ACTOR_ID NUMBER(22) PK FK
FILM_ID NUMBER(22) PK FK
LAST_UPDATE TIMESTAMP
ADDRESS
ADDRESS_ID NUMBER(22) PK
ADDRESS VARCHAR2(50)
ADDRESS2 VARCHAR2(50) N
DISTRICT VARCHAR2(20)
CITY_ID NUMBER(22) FK
POSTAL_CODE VARCHAR2(10) N
PHONE VARCHAR2(20)
LAST_UPDATE TIMESTAMP
CITY
CITY_ID NUMBER(22) PK
CITY VARCHAR2(50)
COUNTRY_ID NUMBER(22) FK
LAST_UPDATE TIMESTAMP
CATEGORY
CATEGORY_ID NUMBER(22) PK
NAME VARCHAR2(25)
LAST_UPDATE TIMESTAMP
FILM_CATEGORY
FILM_ID NUMBER(22) PK FK
CATEGORY_ID NUMBER(22) PK FK
LAST_UPDATE TIMESTAMP
COUNTRY
COUNTRY_ID NUMBER(22) PK
COUNTRY VARCHAR2(50)
LAST_UPDATE TIMESTAMP N

Usaremos la función `dbConnect()` para conectarnos a la base de datos, para cada DBMS hay distintos sitemas de autenticación que deberían ser verificados `DBI` recomienda los siguientes argumentos como parámetros de autentificación

- `host` para el nombre del anfitrió
- `port` para el número de puerto
- `user` para el nombre de usuario
- `password` para la clave
- `dbname` para el nombre de la base de datos en el host

Nos conectamos a la base

En el lab hay que instalar los paquetes `DBI` y `RPostgres` y cargarlos.
Nos conectamos con el siguiente código:

```
library(DBI)
library(RPostgres)
con <- dbConnect(
  RPostgres::Postgres(),
  host = "179.27.99.121",
  port = 5432,
  user = "sue",
  password = "qatar2022",
  dbname = "sue"
)
```

- Primer argumento, driver para conectarnos, en este caso Postgres, entonces usamos `RPostgres::Postgres()` .
- Los otros argumentos dependen de los requerimientos específicos para DBMS, en este caso se requieren `host, port, user, password, dbname`.

dbListTables(con)

- dbListTables() toma cómo único argumento la conexión a la base de datos y nos devuelve un vector de tipo character con todas las tablas y nombres de vistas en la base de datos.
- Después de completar una sesión con un DBMS, libere siempre la conexión con una llamada a dbDisconnect().

```
dbListTables(conn = con)
```

```
 [1] "payment"           "payment_p2022_07" "actor"            "address"
 [5] "category"          "city"             "country"          "customer
 [9] "film"              "film_actor"       "film_category"    "inventor
[13] "language"          "payment_p2022_01" "payment_p2022_02" "payment_
[17] "payment_p2022_04" "payment_p2022_05" "payment_p2022_06" "rental"
[21] "staff"             "store"
```

¿Cómo recuperar los nombres de las columnas de una tabla?

`dbListFields()` permite enumerar los nombres de las columnas de una tabla. Toma como argumentos la conexión a la base de datos y una tabla y devuelve un vector de caracteres de los nombres de las columnas en orden.
En este caso seleccionamos la tabla `film`

```
dbListFields(conn = con, name = "film")
 [1] "film_id"              "title"                "description"
 [4] "release_year"         "language_id"          "original_language_id
 [7] "rental_duration"      "rental_rate"          "length"
[10] "replacement_cost"      "rating"               "last_update"
[13] "special_features"      "fulltext"
```

Leer una tabla en un data.frame

- `dbReadTable()` lee una tabla y retorna un data.frame.
- Es equivalente a hacer una consulta SQL con `SELECT * FROM <name>`.
- Las columnas de el data.frame que obtenemos comparten el mismo nombre que las columnas en la tabla.

```
df <- dbReadTable(conn = con, name = "film")
head(df, 3)
film_id              title
1       1 ACADEMY DINOSAUR
2       2   ACE GOLDFINGER
3       3 ADAPTATION HOLES

1     A Epic Drama of a Feminist And a Mad Scientist who must Battle a
2 A Astounding Epistle of a Database Administrator And a Explorer who m
3     A Astounding Reflection of a Lumberjack And a Car who must Sink a
  release_year language_id original_language_id rental_duration rental_
1         2006           1                   NA               6
2         2006           1                   NA               3
3         2006           1                   NA               7
  replacement_cost rating          last_update                          spe
```

Leer filas y columnas seleccionadas en un data.frame

- Para leer un subconjunto de datos de una tabla en un data.frame, `DBI` brinda funciones para correr consultas `SQL` y manejar los resultados.
- Para conjuntos de datos pequeños se puede usar la función `dbGetQuery()`
- Usa la consulta con `SQL SELECT` para ejecutar y retornar un data.frame

Con la función `dbGetQuery()` simplemente ha que pegar el código SQL en la función como una sequencia de caracteres.

Leer filas y columnas seleccionadas en un data.frame

La consuta especifica las columnas que queremos (`film_id, title` y `description`) y que filas (`records`) estamos interesados. Aquí seleccionamos las columnas `film_id`, `title` y `description` de la tabla `film` donde `release_year` es 2006 (películas estrenadas en 2006)

```
df <- dbGetQuery(con,
                 "SELECT film_id, title, description
                 FROM film
                 WHERE release_year =2006
                 ")
head(df, 3)
```

```
 film_id             title
1       1 ACADEMY DINOSAUR
2       2   ACE GOLDFINGER
3       3 ADAPTATION HOLES

1     A Epic Drama of a Feminist And a Mad Scientist who must Battle a
2 A Astounding Epistle of a Database Administrator And a Explorer who m
3     A Astounding Reflection of a Lumberjack And a Car who must Sink a
```

Ejemplo como armar una base de datos de prueba en R usando una tablas que ya están en `tidyverse`

```
library(RSQL)
library(RSQLite)
library(tidyverse)

data("population")
data("who")

con <- dbConnect(drv = RSQLite::SQLite(),
                 dbname = ":memory:")
dbListTables(con)


dbWriteTable(conn = con,
             name = "population",
             value = population)


dbWriteTable(conn = con,
             name = "who",
             value = who)
```

dplyr & dbplyr y bases de datos

dplyr & dbplyr

- El paquete dplyr tiene por detrás un SQL que permite la comunicación con bases de datos
- El paquete dbplyr traduce código de R en una variante específica de base de datos para que sean usadas por dplyr
- Las variantes de SQL que están disponibles son: Oracle, Microsoft SQL Server, PostgreSQL, Amazon Redshift, Apache Hive, and Apache Impala.

Uno puede escribir el código directamente usando la sintaxis de `dplyr` y se traducirá directamente a código SQL
Varios beneficios de usar `dplyr`

- Mantenemos un lenguaje consistente para los objetos de R y las tablas de la base de datos
- No nencesitamos conocer SQL o variantes de SQL
- Y aprovechar el hecho que `dplyr` usa lazy evaluation.
- La sintaxis de `dplyr` es sencilla pero siempre se puede inspeccionar la traducción a SQL usando la función `show_query()`

General `dplyr`

Paquete para manipulación y transfomación de datos, enfoca en herramientas para trabajar con `data.frames` o `tibbles`

Principales verbos `dplyr`

Las principales funcionalidades de `dplyr` son un conjuntos de verbos que representan distintos objetivos del análisis de datos:

- `filter()`
- `select()`
- `arrange()`
- `mutate()`
- `summarise()`
- `group_by()`

`filter()`

- `filter()` es una función que toma un conjunto de datos y los subdivide de acuerdo a una condición. Selecciona las filas que cumplen la condición
- `filter()` toma expresiones lógicas y devuelve todas las filas que cumplen la condición.

Ejemplo datos de mtcars

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

filter()

Selecciono las filas que tiene la variable mpg mayor a 22

```
library(dplyr)
filter(mtcars, mpg > 22)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

select()

- Se usa para seleccionar una columna particular del 'data.frame()'
- Hay funciones selectoras como:
  - `starts_with()` comienza con un prefijo
  - `ends_with()` termina con un prefijo
  - `matches()` machea una expresión regular
  - `contains()` contiene un string particular

```
select(mtcars, ends_with("t") )
```

```
##                    drat    wt
## Mazda RX4          3.90 2.620
## Mazda RX4 Wag      3.90 2.875
## Datsun 710         3.85 2.320
## Hornet 4 Drive     3.08 3.215
## Hornet Sportabout  3.15 3.440
## Valiant            2.76 3.460
## Duster 360         3.21 3.570
## Merc 240D          3.69 3.190
## Merc 230           3.92 3.150
## Merc 280           3.92 3.440
## Merc 280C          3.92 3.440
## Merc 450SE         3.07 4.070
```

mutate()

- mutate() se usa para crear o modificar variables (columna) como función de valores existentes.
- Siempre se agrega al final de data.frame()
- El número de columnas cambia pero no las filas

```
mutate(mtcars, wtkg = wt*0.45359*1000)
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear c
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3
```

arrange()

- Reordena las filas de un `data.frame`, por defecto ascendente.
- En vez de seleccionar filas (`filter()`) las ordena según alguna variable.
- Si usamos más de una columna, cada columna adicional se usará para resolver empates que surgen de ordenar con las columnas anteriores.
- `desc()` ordena las filas en orden descendente

```
arrange(mtcars, mpg )
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear c
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3
```

Operador pipe%>%

`magrittr` pkg ofrece un conjunto de operadores que hacen tu código más legible.

- estructura una secuencia de operadores de datos de izquierda a derecha
- evita funciones anidadas
- reduce la necesidad de variables locales y definir funciones y es más sencillo agregar pasos en la secuencia de operaciones

Operador pipe%>% lo que queda del lado izquierdo se aplica a expresiones que aparecen del lado derecho. Ejemplo puedo remplazar f(x) como `x%>% f()`

group_by(), summarise()

- group_by() toma los datos y introduce grupos para cada nivel de la variable de grupo
- summarise() Crea resúmenes de data.frame() por grupos, el resultado es una fila para cada grupo de datos.

group_by() y summarise() para un data.frame() agrupado el resumen estadístico será calculado para cada grupo.

group_by(), summarise()

```
mtcars %>%
  group_by(cyl) %>%
  summarise(cyl_n = n() )
```

```
## # A tibble: 3 x 2
##     cyl cyl_n
##   <dbl> <int>
## 1     4    11
## 2     6     7
## 3     8    14
```

dplyr y bases de datos

La operación equivalente usando `dplyr` reconstruye la consulta `SQL` usando tres funciones para especificar la tabla (`tbl()`), el subconjunto de filas (`filter()`) y las columnas que necesitamos (`select()`).

```
library(dplyr)
lazy_df <-
  tbl(con, "film") %>%
  filter(release_year == 2006 & rating == "G") %>%
  select(film_id, title, description)

show_query(lazy_df)
```

```
tbl(con, "payment") %>%
  colnames()
  filter(release_year == 2006 & rating == "G") %>%
  select(film_id, title, description)
```

Tu Turno, consultas básicas!

- Utilizando DBI y consultas SQL, selecioná de la tabla `customer`, selccionando las colunas `customer_id`, `store_id`.
- Con la consulta anterior filtrá solamente los que no están activos (donde `activo` vale 0)
- Ordená la consulta según `store_id` decendente.
- Ahora realizá la misma consulta usando funciones de `dplyr`

Solución

```
consulta_sql<- dbGetQuery(con,
              "SELECT  customer_id, store_id
              FROM customer
              WHERE active = 0 ORDER BY store_id DESC")

consuta_dplyr <- tbl(con,"customer") %>%
  select(customer_id, store_id, active) %>%
  filter(active==0) %>%
  arrange(desc(store_id) )
#podemos ver que traduce
show_query(consuta_dplyr )
```

Qué diferencia ven en la consulta con dplyr y su traducción a SQL con hacerlo derecho en SQL?

Tu turno!

- Utilizando DBI y consultas SQL, cuantos clientes únicos hay en la tabla `customer` también hacelo con `dplyr` usando la función `distinct` en ambos caso guardá el resultado como `cuento_cl`

Solución

```
dbGetQuery(con, "SELECT
           COUNT(DISTINCT customer_id) AS cuento_cl FROM customer")

tbl(con,"customer") %>%
  distinct(customer_id) %>%
  count(name="cuento_cl")
```

Tu Turno!

- Utilizando DBI y consultas SQL, obtener los 5 actores con más películas , indicando primer nombre, primer apellido y la cantidad.
- Repetir el ejecicio anterior ahora usando dplyr (usar `left_join`).

```
t1<- tbl(con, "actor")
t2<- tbl(con, "film_actor")

bb <-t2 %>%
  group_by(actor_id) %>%
  summarise(n = n()) %>%
  arrange(desc(n)) %>%
  head(n = 5) %>%
  left_join(t1, by = 'actor_id' )

show_query(bb)
```