



# Computação em Estatística 2

## SQL

- Felipe Melo - 222036014
- Heitor Roncato-211061000
- André Costa - 180116223
- João Vitor Rodrigues-  
200064835



# Tópicos

1. Introdução ao SQL
2. SQLite
3. Consultas avançadas
4. Visualização de dados

---

# Introdução ao SQL



- Structured Query Language
- Linguagem de programação utilizada para gerenciar e manipular bancos de dados relacionais
- Linguagem baseada em consultas
- Armazena grande volume de dados

# Funções do SQL

Criando um banco de dados

```
1 •      create database veiculos;
2 •      use veiculos;
3
4 •      create table carros(
5          id_carro int auto_increment primary key,
6          nome varchar(30),
7          ano varchar(4),
8          valor decimal(10,2)
9      );
```



## Inserindo dados

```
10
11 •   INSERT INTO carros (nome, ano, valor)
12     VALUES ('Onix', 2015, 25000.00),
13         ('Civic', 2018, 32000.00),
14         ('Corolla', 2019, 40000.00),
15         ('Ka', 2016, 28000.00);
```

## Visualizar o banco de dados

```
24 • select * from carros;
```

	id_carro	nome	ano	valor
▶	1	Onix	2015	25000.00
	2	Civic	2018	32000.00
	3	Corolla	2019	40000.00
	4	Ka	2016	28000.00
	5	Palio	2017	35000.00
	6	Versa	2014	22000.00
	7	Cruze	2020	45000.00
	8	Sentra	2013	18000.00
	9	Jetta	2019	38000.00
	10	Polo	2016	27000.00
*	HULL	HULL	HULL	HULL

## Filtrando o banco de dados

```
26 •   SELECT nome, valor FROM carros  
27     WHERE valor > 30000;
```

	nome	valor
▶	Civic	32000.00
	Corolla	40000.00
	Palio	35000.00
	Cruze	45000.00
	Jetta	38000.00

## Atualizando o banco de dados

```
29 • UPDATE carros  
30     SET nome = 'Kwid', ano = 2022, valor = 39000.00  
31     WHERE id_carro = 1;
```

	id_carro	nome	ano	valor
▶	1	Kwid	2022	39000.00
	2	Civic	2018	32000.00
	3	Corolla	2019	40000.00
	4	Ka	2016	28000.00
	5	Palio	2017	35000.00
	6	Versa	2014	22000.00
	7	Cruze	2020	45000.00
	8	Sentra	2013	18000.00
	9	Jetta	2019	38000.00
	10	Polo	2016	27000.00
*	NULL	NULL	NULL	NULL



# SQLite

Aplicação do SQL no RStudio

# Introdução

## O que é SQLite?

---

- Um mecanismo de banco de dados, para auxiliar no gerenciamento de um banco de dados, que pode ser adaptado para diferentes fins.
- Possui pacotes aplicáveis para diferentes linguagens.
- Implementação completa do SQL .
- Aplicado no R, retorna as consultas em dataframes.

# Uso no R

## Criando uma conexão com o banco de dados desejado:

```
install.packages('RSQLite')
library('RSQLite')
library('DBI')
```

```
con <- dbConnect(RSQLite::SQLite(),'chinook.db')
```

```
dbListTables(con)
```

```
> dbListTables(con)
[1] "albums"           "artists"          "best_acdc"        "customers"
[5] "dimcustomer"     "employees"        "genres"          "invoice_items"
[9] "invoices"         "media_types"      "mtcars"          "playlist_track"
[13] "playlists"        "salesbycountrybyartist" "sqlite_sequence" "sqlite_stat1"
[17] "top15"            "tracks"
```

# Uso no R

Executando uma consulta usando **SELECT e FROM**:

```
comando <- 'SELECT * FROM mtcars'
```

```
resultado <- dbGetQuery(con, comando)
```

ou

```
resultado <- dbGetQuery(con, 'SELECT * FROM mtcars')
```

```
resultado
```

▲	row_names	▼	mpg	▼	cyl	▼	disp	▼	hp	▼	drat	▼	wt	▼	qsec	▼	vs	▼	am	▼	gear	▼	carb	▼
1	Mazda RX4		21.0		6		160.0		110		3.90		2.620		16.46		0		1		4		4	
2	Mazda RX4 Wag		21.0		6		160.0		110		3.90		2.875		17.02		0		1		4		4	
3	Datsun 710		22.8		4		108.0		93		3.85		2.320		18.61		1		1		4		1	
4	Hornet 4 Drive		21.4		6		258.0		110		3.08		3.215		19.44		1		0		3		1	
5	Hornet Sportabout		18.7		8		360.0		175		3.15		3.440		17.02		0		0		3		2	
6	Valiant		18.1		6		225.0		105		2.76		3.460		20.22		1		0		3		1	
7	Duster 360		14.3		8		360.0		245		3.21		3.570		15.84		0		0		3		4	
8	Merc 240D		24.4		4		146.7		62		3.69		3.190		20.00		1		0		4		2	
9	Merc 230		22.8		4		140.8		95		3.92		3.150		22.90		1		0		4		2	
10	Merc 280		19.2		6		167.6		123		3.92		3.440		18.30		1		0		4		4	
11	Merc 280C		17.8		6		167.6		123		3.92		3.440		18.90		1		0		4		4	
12	Merc 450SE		16.4		8		275.8		180		3.07		4.070		17.40		0		0		3		3	
13	Merc 450SL		17.3		8		275.8		180		3.07		3.730		17.60		0		0		3		3	
14	Merc 450SLC		15.2		8		275.8		180		3.07		3.780		18.00		0		0		3		3	
15	Cadillac Fleetwood		10.4		8		472.0		205		2.93		5.250		17.98		0		0		3		4	
16	Lincoln Continental		10.4		8		460.0		215		3.00		5.424		17.82		0		0		3		4	
17	Chrysler Imperial		14.7		8		440.0		230		3.23		5.345		17.42		0		0		3		4	
18	Fiat 128		32.4		4		78.7		66		4.08		2.200		19.47		1		1		4		1	
19	Honda Civic		30.4		4		75.7		52		4.93		1.615		18.52		1		1		4		2	
20	Toyota Corolla		33.9		4		71.1		65		4.22		1.835		19.90		1		1		4		1	
21	Toyota Corona		21.5		4		120.1		97		3.70		2.465		20.01		1		0		3		1	

# Uso no R

## Uso da função WHERE:

```
comando <- 'SELECT row_names, cyl FROM mtcars WHERE cyl = 8'
```

```
resultado <- dbGetQuery(con, comando)
```

```
resultado
```

	row.names	cyl
1	Hornet Sportabout	8
2	Duster 360	8
3	Merc 450SE	8
4	Merc 450SL	8
5	Merc 450SLC	8
6	Cadillac Fleetwood	8
7	Lincoln Continental	8
8	Chrysler Imperial	8
9	Dodge Challenger	8
10	AMC Javelin	8
11	Camaro Z28	8
12	Pontiac Firebird	8
13	Ford Pantera L	8
14	Maserati Bora	8

# Uso no R

## Uso da função ORDER BY:

```
comando <- 'SELECT * FROM mtcars ORDER BY hp DESC'
```

```
resultado <- dbGetQuery(con, comando)
```

```
resultado
```

#	row_names	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
2	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
3	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
4	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
5	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
6	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
7	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
8	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
9	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
10	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
11	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
12	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
13	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
14	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
15	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
16	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
17	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
18	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
19	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
20	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
21	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
22	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
23	Valliant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
24	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
25	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
26	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
27	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
28	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
29	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
30	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
31	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
32	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2

# Uso no R

Uso da função GROUP BY:

	cyl	n_por_cyl
1	4	11
2	6	7
3	8	14

```
comando <- 'SELECT cyl, COUNT(*) AS n_por_cyl  
FROM mtcars  
GROUP BY cyl'
```

```
resultado <- dbGetQuery(con, comando)
```

```
resultado
```



# Consultas Avançadas em SQL

# Consultas Avançadas em SQL

---

Apresentação dos tópicos a serem abordados:

- Recursos avançados do SQL: junções (JOIN), subconsultas (subqueries) e agregação de dados.
- Exemplo de junções (JOIN): combinação de dados de múltiplas tabelas.
- Exemplo de subconsultas (subqueries): realização de consultas aninhadas.
- Exemplo de agregação de dados: operações como SUM, AVG, COUNT, MAX, MIN.
- Consultas envolvendo múltiplas tabelas e operações de agregação.
- Funções: Controle de Fluxo, agrupamento, Agregação, Navegação de Hierarquia.

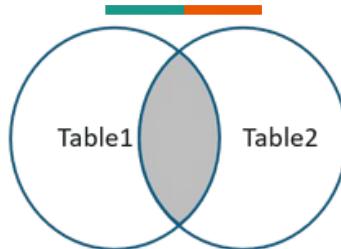
# Banco de Dados Chinook

- Banco de dados fictício utilizado para fins de demonstração.
- Simula um sistema de gerenciamento de uma loja de música.
- Contém tabelas de informações sobre álbuns, artistas, clientes, funcionários, gênero e outros dados relacionados.

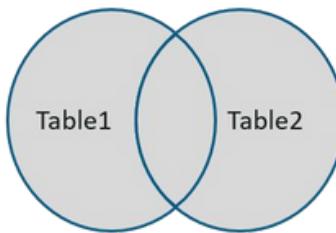
```
# Conectar ao banco de dados chinook.db
con <- dbConnect(RSQLite::SQLite(), "chinook.db")
# Listar as tabelas no banco de dados
dbListTables(con)

## [1] "albums"          "artists"         "customers"        "employees"
## [5] "genres"          "invoice_items"   "invoices"         "media_types"
## [9] "playlist_track"  "playlists"        "sqlite_sequence" "sqlite_stat1"
## [13] "tracks"
```

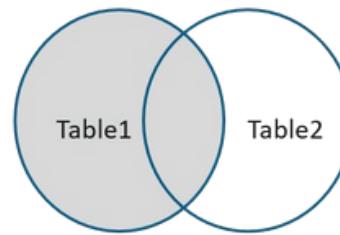
# Cláusula Join - combinação de colunas de tabelas



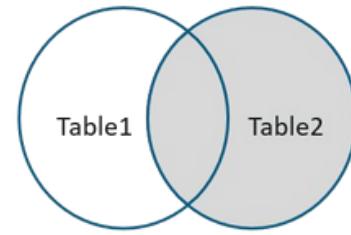
**INNER JOIN**



**FULL JOIN**



**LEFT JOIN**



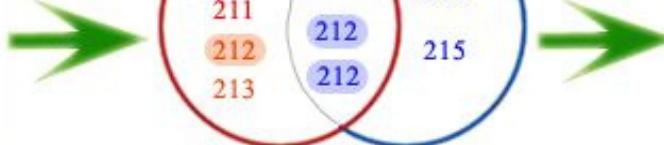
**RIGHT JOIN**

Left Join or Left outer Join

table-A    table-B

210
211
212
213

210
214
215
212
212



result

210	
211	
212	
213	
	214
	215

## Consulta com INNER JOIN

- Utilização do INNER JOIN (retorna apenas os registros que possuem correspondência nas duas tabelas) para combinar dados das tabelas tracks e albums.

```
query_inner <- "
SELECT t.TrackId, t.Name, a.ArtistId
FROM tracks AS t
INNER JOIN albums AS a ON t.AlbumId = a.AlbumId
"
resultado_inner <- dbGetQuery(con, query_inner)
```

##	TrackId	Name	ArtistId
## 1	1	For Those About To Rock (We Salute You)	1
## 2	6	Put The Finger On You	1
## 3	7	Let's Get It Up	1

## Consulta com LEFT JOIN

- Utilização do LEFT JOIN: Retorna todos os registros da tabela à esquerda, primeira tabela mencionada na consulta, e os registros correspondentes da tabela à direita, segunda tabela mencionada na consulta.

```
query_left <- "
SELECT t.TrackId, t.Name, a.ArtistId
FROM tracks AS t
LEFT JOIN albums AS a ON t.AlbumId = a.AlbumId
"
resultado_left <- dbGetQuery(con, query_left)
```

```
##   TrackId                      Name ArtistId
## 1      1 For Those About To Rock (We Salute You)     1
## 2      2          Balls to the Wall                 2
## 3      3           Fast As a Shark                2
```

## Consulta com RIGHT JOIN

- Utilização do RIGHT JOIN: Retorna todos os registros da tabela à direita e os registros correspondentes da tabela à esquerda. Se não houver correspondência na tabela à esquerda, os valores NULL são retornados para as colunas dessa tabela.

```
query_right <- "
SELECT t.TrackId, t.Name, a.ArtistId
FROM albums AS a
RIGHT JOIN tracks AS t ON a.AlbumId = t.AlbumId
"
resultado_right <- dbGetQuery(con, query_right)
```

##	TrackId	Name	ArtistId
## 1	1	For Those About To Rock (We Salute You)	1
## 2	6	Put The Finger On You	1
## 3	7	Let's Get It Up	1

## Consulta com FULL JOIN

- Utilização do FULL JOIN: Retorna todos os registros de ambas as tabelas, combinando-os com base nas colunas especificadas. Se houver correspondência, os valores correspondentes são retornados. Se não houver correspondência, os valores NULL são retornados para as colunas ausentes.

```
query_full <- "
SELECT t.TrackId, t.Name, a.ArtistId
FROM tracks AS t
FULL JOIN albums AS a ON t.AlbumId = a.AlbumId
"
resultado_full <- dbGetQuery(con, query_full)
```

##	TrackId	Name	ArtistId
## 1	1	For Those About To Rock (We Salute You)	1
## 2	2	Balls to the Wall	2
## 3	3	Fast As a Shark	2

## Exemplo de subconsultas em cláusulas WHERE:

- Retornando o nome das faixas de um artista específico com ArtistId igual a 1, utilizando uma subconsulta para filtrar os AlbumId correspondentes.

```
query_where <- "
SELECT Name
FROM tracks
WHERE AlbumId IN (
    SELECT AlbumId
    FROM albums
    WHERE ArtistId = 1
)
"
resultado_where <- dbGetQuery(con, query_where)
resultado_where
```

```
##                                     Name
## 1 For Those About To Rock (We Salute You)
```

## Exemplo de subconsultas em cláusulas SELECT:

- Subconsulta usando SELECT para calcular a média dos preços de cada faixa na tabela tracks. Retorna o nome da faixa (Name) e o preço médio (AveragePrice) calculado pela subconsulta.

```
query_select <- "
SELECT Name, (
    SELECT AVG(UnitPrice)
    FROM invoice_items
    WHERE tracks.TrackId = invoice_items.TrackId
) AS AveragePrice
FROM tracks
LIMIT 10
"
resultado_select <- dbGetQuery(con, query_select)
```

```
##                                     Name AveragePrice
## 1 For Those About To Rock (We Salute You)      0.99
```

## Exemplos de Operações de Agregação

---

- SUM: Calcula a soma de valores em uma coluna numérica.
- AVG: Calcula a média de valores em uma coluna numérica.
- COUNT: Conta o número de registros em uma tabela ou coluna.
- MAX: Encontra o valor máximo em uma coluna.
- MIN: Encontra o valor mínimo em uma coluna.

# Consultas com Agrupamento de Dados e Cálculos Agregados

- Retornando o nome do gênero, o número de faixas em cada gênero e o preço médio das faixas em cada gênero. Isso é feito através de um agrupamento dos dados pela coluna Genrelid e do uso das funções de agregação COUNT e AVG.

```
# Consulta com agrupamento e cálculos agregados
query_aggregation <- "
SELECT g.Name AS Genre, COUNT(t.TrackId) AS TrackCount, AVG(t.UnitPrice) AS AveragePrice
FROM tracks AS t
JOIN genres AS g ON t.GenreId = g.GenreId
GROUP BY g.Name
"
resultado_aggregation <- dbGetQuery(con, query_aggregation)
resultado_aggregation
```

# OUTPUT:

```
##             Genre TrackCount AveragePrice
## 1      Alternative        40       0.99
## 2 Alternative & Punk     332       0.99
## 3          Blues         81       0.99
## 4      Bossa Nova        15       0.99
## 5      Classical        74       0.99
## 6          Comedy        17       1.99
## 7          Drama         64       1.99
## 8  Easy Listening        24       0.99
## 9 Electronica/Dance     30       0.99
## 10     Heavy Metal        28       0.99
## 11    Hip Hop/Rap        35       0.99
## 12          Jazz        130       0.99
## 13          Latin        579       0.99
## 14          Metal        374       0.99
## 15          Opera         1       0.99
```

# Exemplo de Consulta Complexa

- Número de faturas e vendas em ordem decrescente por estado e país (eua, califorina).

```
query_complex <- "
SELECT c.Country, COUNT(DISTINCT i.InvoiceId) AS TotalInvoices, SUM(i.Total) AS TotalSales
FROM customers AS c
JOIN invoices AS i ON c.CustomerId = i.CustomerId
WHERE c.Country IN (
    SELECT Country FROM customers WHERE State = 'CA'
)
GROUP BY c.Country
HAVING COUNT(DISTINCT i.InvoiceId) > 10
ORDER BY TotalSales DESC
"
resultado_complex <- dbGetQuery(con, query_complex)
#   Country TotalInvoices TotalSales
1     USA          91      523.06
```

## Funções CASE e IF-ELSE

- Consulta utilizando a função CASE para classificar álbuns de acordo com o número de faixas

```
query <- "
SELECT AlbumId, Title,
CASE
    WHEN (
        SELECT COUNT(*)
        FROM tracks
        WHERE tracks.AlbumId = albums.AlbumId
    ) > 10 THEN 'Mais de 10 faixas'
    ELSE 'Menos de 10 faixas'
END AS Track_Count_Category
FROM albums
"
result <- dbGetQuery(con, query)
result
```

# Resultado:

---

##	AlbumId	Title	Track_Count_Category
## 1	1	For Those About To Rock We Salute You	Menos de 10 faixas
## 2	2	Balls to the Wall	Menos de 10 faixas
## 3	3	Restless and Wild	Menos de 10 faixas
## 4	4	Let There Be Rock	Menos de 10 faixas
## 5	5	Big Ones	Mais de 10 faixas
## 6	6	Jagged Little Pill	Mais de 10 faixas
## 7	7	Facelift	Mais de 10 faixas
## 8	8	Warner 25 Anos	Mais de 10 faixas
## 9	9	Plays Metallica By Four Cellos	Menos de 10 faixas
## 10	10	Audioslave	Mais de 10 faixas
## 11	11	Out Of Exile	Mais de 10 faixas
## 12	12	BackBeat Soundtrack	Mais de 10 faixas
## 13	13	The Best Of Billy Cobham	Menos de 10 faixas
## 14	14	Alcohol Fueled Brewtality Live! [Disc 1]	Mais de 10 faixas
## 15	15	Alcohol Fueled Brewtality Live! [Disc 2]	Menos de 10 faixas
## 16	16	Black Sabbath	Menos de 10 faixas

## OUTRO EXEMPLO:

- Consulta utilizando a função IF-ELSE para classificar faixas de acordo com seu gênero

```
query <- "
SELECT TrackId, Name, GenreId,
       IF(GenreId = 1, 'Rock',
          IF(GenreId = 2, 'Jazz',
             IF(GenreId = 3, 'Metal', 'Outro'))
       )
     ) AS Genre_Category
FROM tracks
"
resultado <- dbGetQuery(con, query)
resultado
```

## Utilizando a função de agrupamento (GROUP BY):

- Consulta utilizando a função GROUP BY para obter o total de vendas por país.

```
query <- "
SELECT BillingCountry, SUM(Total) AS TotalSales
FROM invoices
GROUP BY BillingCountry
"
"
```

- Consulta utilizando a função HAVING para filtrar os resultados de acordo com uma condição

```
query <- "
SELECT GenreId, COUNT(*) AS TrackCount
FROM tracks
GROUP BY GenreId
HAVING TrackCount > 100
"
"
```

## Funções de agregação em SQL (GROUP\_CONCAT, STDDEV, VARIANCE)

- Utilizando a função GROUP\_CONCAT
- Consulta utilizando a função GROUP\_CONCAT para obter todos os gêneros musicais por artista

```
query <- "
SELECT artists.Name AS Artist, GROUP_CONCAT(DISTINCT genres.Name) AS Genres
FROM artists
JOIN albums ON artists.ArtistId = albums.ArtistId
JOIN tracks ON albums.AlbumId = tracks.AlbumId
JOIN genres ON tracks.GenreId = genres.GenreId
GROUP BY artists.ArtistId
"
resultado <- dbGetQuery(con, query)
```

## Outro exemplo:

- Consulta utilizando as funções STDDEV e VARIANCE para calcular a variação e desvio padrão do preço de cada faixa.

```
query <- "
SELECT TrackId, Name,
       ROUND(AVG(UnitPrice), 2) AS AvgPrice,
       ROUND(STDDEV(UnitPrice), 2) AS PriceStdDev,
       ROUND(VARIANCE(UnitPrice), 2) AS PriceVariance
FROM tracks
GROUP BY TrackId
"
resultado <- dbGetQuery(con, query)
resultado
```

- Funções de navegação de hierarquia em SQL (CONNECT BY, START WITH)
- Consulta utilizando a função CONNECT BY para obter a hierarquia das playlists.

## Função CONNECT BY

- Utilizando a função CONNECT BY para obter a hierarquia das playlists

```
query <- "
SELECT PlaylistId, Name, PlaylistId AS ParentId, 0 AS Level
FROM playlists
WHERE PlaylistId = 1
UNION ALL
SELECT p.PlaylistId, p.Name, t.PlaylistId AS ParentId, t.Level + 1 AS Level
FROM playlists p
JOIN playlist_track t ON p.PlaylistId = t.TrackId
"
resultado <- dbGetQuery(con, query)
resultado
```

## Outro Exemplo:

- Consulta utilizando a função START WITH para obter a hierarquia de funcionários.

```
query <- "
SELECT EmployeeId, FirstName, LastName, ReportsTo, 0 AS Level
FROM employees
WHERE EmployeeId = 1
UNION ALL
SELECT e.EmployeeId, e.FirstName, e.LastName, e.ReportsTo, t.Level + 1 AS Level
FROM employees e
JOIN employees t ON e.EmployeeId = t.ReportsTo
"
resultado <- dbGetQuery(con, query)
resultado
```

---

# Visualização de Dados

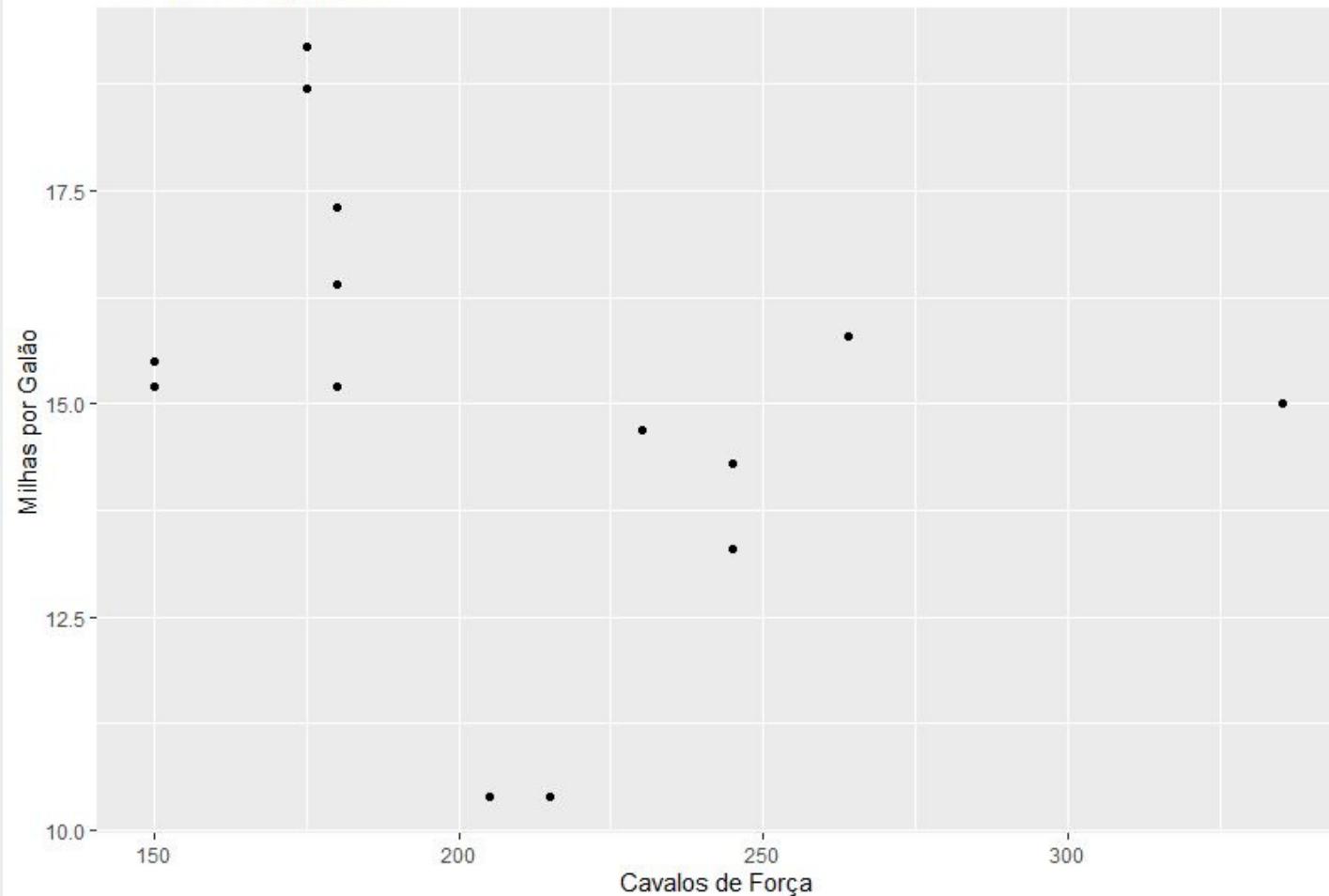
Como o retorno de consultas do SQLite é dado em forma de dataframes, podemos utilizar o pacote ggplot2 para gerar gráficos a partir deles.

```
library(ggplot2)

dados <- dbGetQuery(con, 'SELECT * FROM mtcars WHERE cyl = 8')

ggplot(dados, aes(x = hp, y = mpg)) +
  geom_point() +
  labs(x = "Cavalos de Força", y = "Milhas por Galão", title = "Gráfico de Dispersão")
```

## Gráfico de Dispersão



---

## Ex: Gráfico de Barras

```
dados2 <- dbGetQuery(con, 'SELECT * FROM mtcars')

ggplot(dados2, aes(x = cyl)) +
  geom_bar() +
  labs(x = "Cilindrada", y = "Contagem", title = "Gráfico de Barras")
```

## Gráfico de Barras

