[MAC0313]
Introdução aos Sistemas de Bancos de Dados
Aula 25
Acessando um Banco de Dados a partir do R

Kelly Rosa Braghetto

DCC–IME–USP

14 de novembro de 2017

Usando a SQL para cálculos numéricos e estatísticos

Somente as funções básicas estão disponíveis na versão padrão da linguagem

- Operadores matemáticos: $+, -, *, /$
- Funções de agregação: MAX, MIN, COUNT, SUM, AVG

Exemplos de consultas difíceis de se fazer em SQL

- Medianas, quartis
- Mínimos quadrados (regressão linear)

Usando a SQL para cálculos numéricos e estatísticos

Outros problemas

- Nem sempre as funções de agregação são implementadas no SGBD com garantias de acurácia numérica
- Cálculos envolvendo diferentes tipos de valores numéricos podem ter resultados diferentes dos esperados
  - A SQL possui uma grande variedade de tipos numéricos (smallint, int, float, double, numeric)
  - Frequentemente, o cálculo de expressões matemáticas envolvendo diferentes tipos de valores requer a realização de conversões de tipos
  - As conversões entre tipos numéricos podem produzir truncamentos ou arredondamentos inesperados

Solução: usar um pacote estatístico para analisar os dados armazenados em um BD

Analisando os dados de um BD por meio de um pacote estatístico

Diferentes estratégias possíveis:

1. Extrair os dados do BD, armazená-los em arquivos (geralmente no formato CSV) e depois importá-los no software estatístico

   ou

2. A partir do software estatístico, abrir uma conexão com o BD e obter os dados desejados por meio da submissão de consultas ao SGBD

   ou

3. Implementar dentro do SGBD novas funções de agregação, que podem ser usadas em consultas SQL mas que são executadas por um software estatístico

Um "parênteses" sobre arquivos CSV

- CSV = *Comma-separated values*
- Arquivos CSV armazena dados tabulares (números ou texto) em formato puramente textual
- Cada linha do arquivo corresponde a um registro
- Cada registro tem um ou mais campos, separados por vírgulas
- Esse formato não é padronizado, mas softwares de análise de dados costumam reconhecê-lo facilmente
  - algum outro caractere pode ser usar usado para separar campos no lugar da vírgula (ex.: ';', 'tab', '|', etc.)
  - strings podem ter caracteres delimitadores (como aspas ou apóstrofes)
  - Valores nulos geralmente são denotados por vazios

Analisando os dados de um BD por meio de um pacote estatístico

Diferentes estratégias possíveis:

1. **Extrair os dados do BD, armazená-los em arquivos (texto: CSV) e depois importá-los no software estatístico**

   Vantagens

   - Simples de se implementar (não requer novos conhecimentos específicos)

   Desvantagens

   - Viável quando deseja-se fazer a análise dos dados do BD uma única (ou poucas) vez(es)
   - Dificulta a análise de grandes volumes de dados

Analisando os dados de um BD por meio de um pacote estatístico

Diferentes estratégias possíveis:

2. **A partir do software estatístico, abrir uma conexão com o BD e obter os dados desejados por meio da submissão de consultas ao SGBD**

   Vantagens

   - Garante que os dados analisados são sempre os mais novos presentes no BD
   - Fornece facilidades para lidar com grandes volumes de dados (recuperação em "lotes")

   Desvantagens

   - Requer o conhecimento de funções específicas para a comunicação com o BD a partir do software estatístico
   - A transferência de muitos dados entre o BD e o software estatístico pode ser lenta

Analisando os dados de um BD por meio de um pacote estatístico

Diferentes estratégias possíveis:

3. **Implementar dentro do SGBD novas funções de agregação, que podem ser usadas em consultas SQL mas que são executadas por um software estatístico**

   Vantagens

   - Garante que os dados analisados são sempre os mais novos presentes no BD
   - Isenta o usuário de se preocupar com o tratamento dos dados quando esses não cabem na memória principal

   Desvantagens

   - Requer conhecimentos específicos para a criação de novas funções de agregação no SGBD
   - A transferência de muitos dados entre o BD e o software estatístico pode ser lenta

Sobre o R

- Para exemplificar as estratégias discutidas nos slides anteriores, usaremos o SGBD PostgreSQL e o software estatístico **R**
- O **R** é um ambiente de software para computação estatística e gráficos
- Ele é um software livre e roda em uma grande variedade de plataformas Linux, Windows e MacOS
- Site: `http://www.r-project.org/`

Instalação do R

- No Linux: pacotes `r-base` (versão completa) ou `littler` (versão "leve")

  ```
  $ sudo apt-get install r-base
  ```
  ou
  ```
  $ sudo apt-get install littler
  ```

- No Windows ou (MAC) OS X:

  Baixar o instalador em:
  `http://nbcgib.uesc.br/mirrors/cran/`

- R com interface gráfica: RStudio
  (cuidado, algumas versões são pagas!)

  `https://www.rstudio.com/`

Estratégia 1: Exportando dados do BD

Exemplo 1 de exportação de dados no PostgreSQL

- No `psql` (linha de comando no Linux), usar o comando `\copy`:

```
\copy Produto to 'produto.csv'
              (FORMAT csv, DELIMITER ';', HEADER true,
               FORCE_QUOTE (fabricante,tipo));
```

O comando copia o conteúdo da tabela `Produto` em um arquivo texto (CSV), chamado "produto.csv", onde cada linha do arquivo corresponde a uma tupla da tabela e os valores dos atributos em cada linha aparecem separados pelo caracter delimitador ponto-e-vírgula (';') e o valor dos atributos `fabricante` e `tipo` (que é são *strings*) ficam entre aspas

Estratégia 1: Exportando dados do BD

Exemplo 2 de exportação de dados no PostgreSQL

- No `psql` (linha de comando no Linux), usar o comando `\copy`:

```
\copy (SElECT * FROM PC WHERE ram = 128)
        TO 'pc.csv'
        (FORMAT csv, DELIMITER ';', HEADER true);
```

  No exemplo acima, o comando é usado para gravar o resultado de uma consulta SQL em um arquivo CSV chamado "pc.csv"

Para ver outros parâmetros de configuração do comando `\copy`, acesse:

https://www.postgresql.org/docs/current/static/sql-copy.html

Estratégia 1: Exportando dados do BD

Exemplo 2 de exportação de dados no PostgreSQL

- No `psql` (linha de comando no Linux), usar o comando `\copy`:

```
\copy (SElECT * FROM PC WHERE ram = 128)
          TO 'pc.csv'
          (FORMAT csv, DELIMITER ';', HEADER true);
```

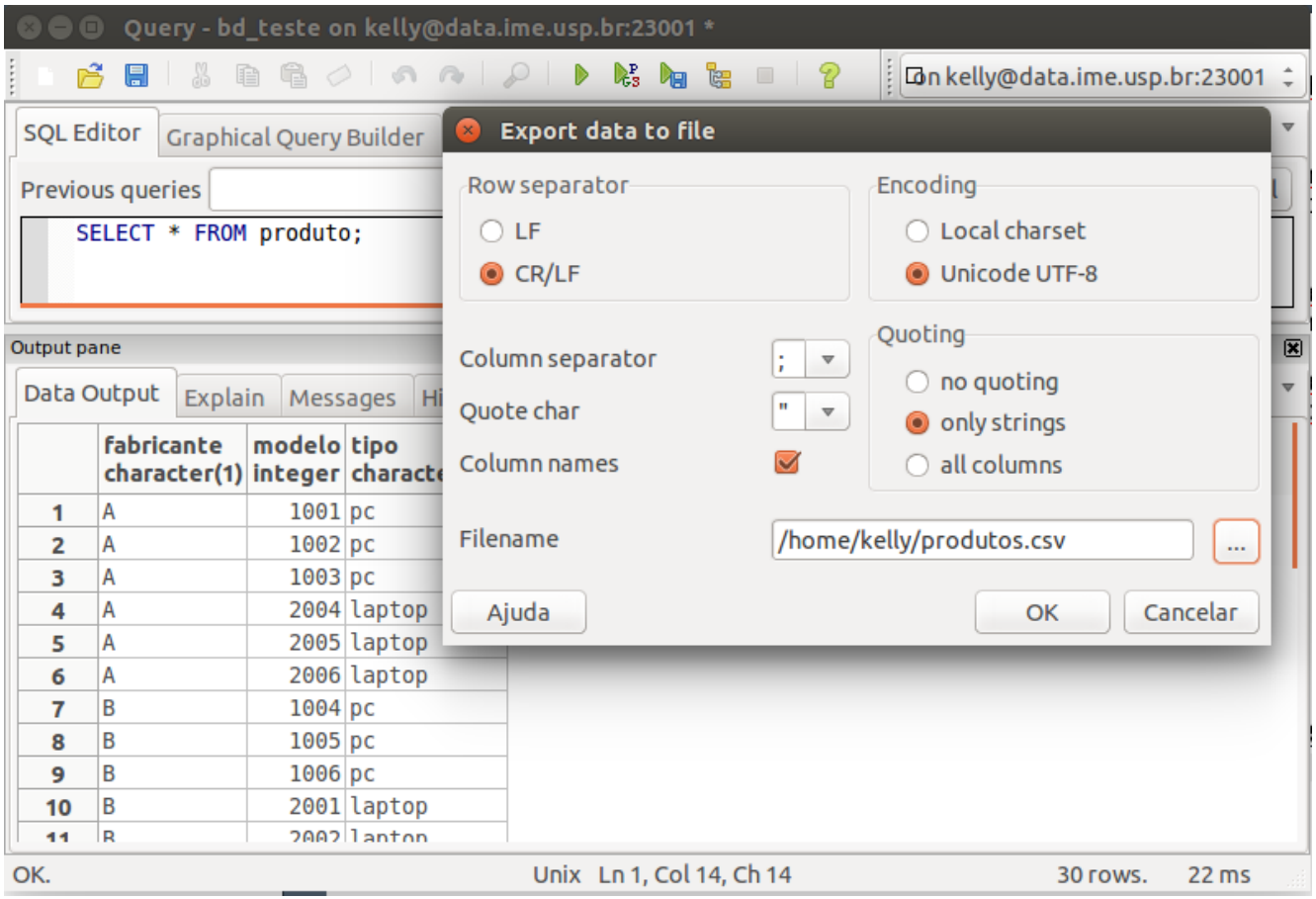
O conteúdo do arquivo `'pc.csv'` gerado é:

```
modelo;velocidade;ram;hd;cd;preco
1002;1500;128;60;2x;2499.00
1003;866;128;20;8x;1999.00
1005;1000;128;20;2x;1499.00
1007;1400;128;80;2x;2299.00
1009;1200;128;80;6x;1699.00
1011;1100;128;60;6x;1299.00
```

Estratégia 1: Exportando dados do BD

Exemplo 3 de exportação de dados no PostgreSQL

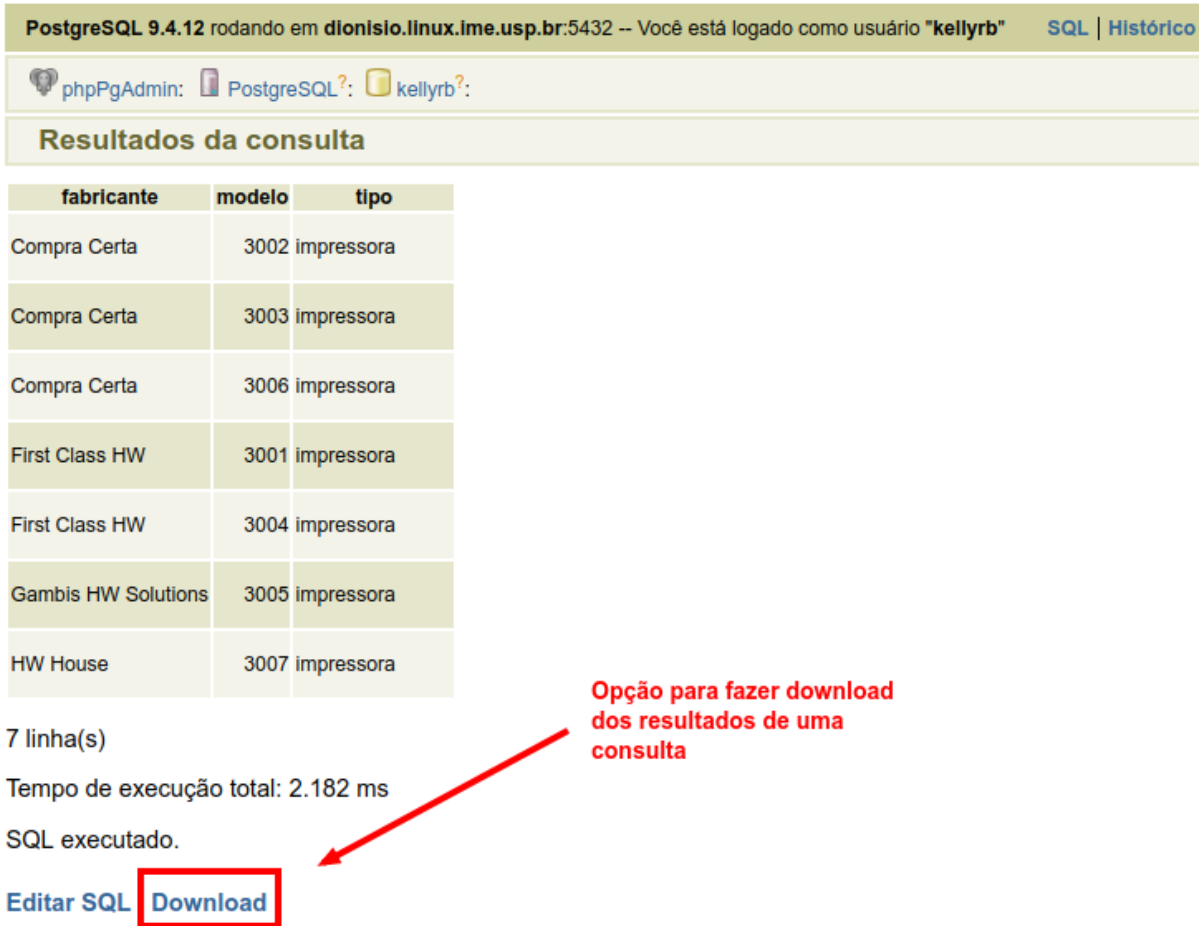
No `PgAdmin`, na janela `Query Tool` é possível gravar o resultado de uma consulta SQL num arquivo CSV, no menu "File » Export..."



Estratégia 1: Exportando dados do BD

Exemplo 4 de exportação de dados no PostgreSQL

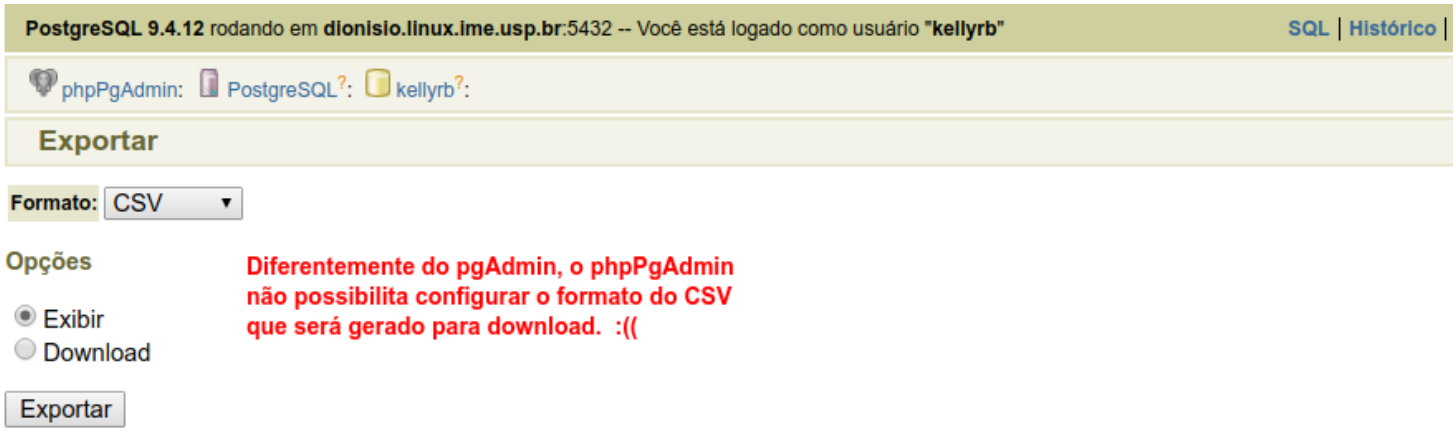
No `phpPgAdmin`, na página que mostra os resultados de uma consulta SQL, há uma opção para fazer *download* dos resultados



Estratégia 1: Exportando dados do BD

Exemplo 4 de exportação de dados no PostgreSQL

No `phpPgAdmin`, não é possível configurar o formato do CSV gerado (separador de campos, delimitador de *strings*, inclusão de cabeçalho, etc.)



Estratégia 1: Importando arquivos no R

Exemplo de comando de importação de arquivos CSV no R

- No prompt do `R` (linha de comando no Linux), usar a função `read.csv()`:

  ```
  dados <- read.csv("pc.csv", sep=";")
  ```

  O comando acima lerá o conteúdo do arquivo `'pc.csv'` e o armazenará em um objeto do tipo *data.frame* do R.

Estratégia 2: Acessando o BD a partir do software estatístico

- O software estatístico pode estar instalado em uma máquina diferente da do SGBD
- A conexão com um BD hospedado em um SGBD é feita por meio de funcionalidades especiais providas pelo software estatístico
- No R, o pacote **DBI – Database Interface** é quem provê a interface de acesso a BDs
  - A interface DBI permite que diferentes tipos de SGBDs sejam acessados a partir do R de **modo uniforme**

Estratégia 2: Acessando o BD a partir do software estatístico

A interface DBI do R é composta por três elementos principais:

- o *driver* – que provê a comunicação entre uma sessão do R e um tipo particular de SGBD (como o PostgreSQL, por exemplo)
- a *conexão* – que pode ser vista como o "canal" de comunicação com o SGBD; é quem transmite as consultas e outros comandos ao SGBD
- o *resultado* – que "rastreia" o status de uma consulta, como o número de linhas que foram recuperadas e se a consulta foi ou não completada

Estratégia 2: Acessando um BD a partir do R

Para criar um *data frame* no R com dados recuperados de um BD, há 4 passos básicos:

1. Instanciar um driver
2. A partir do *driver* instanciado, criar uma conexão com o BD
3. Submeter uma consulta ao BD por meio da conexão criada
4. Manipular os resultados obtidos da execução da consulta

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Instalação da DBI para PostgreSQL no R

- Para que possamos instanciar um *driver* para o PostgreSQL, precisamos primeiro instalar no R a implementação da DBI para PostgreSQL – a *RPostgreSQL*
- No prompt do R, digite:

```
> install.packages("RPostgreSQL");
```

- Após a instalação da RPostgreSQL, você estará apto a acessar um BD hospedado num SGBD PostgreSQL a partir do R
- Para informações sobre a RPostgreSQL, acesse: `https://cran.r-project.org/web/packages/RPostgreSQL/index.html`

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Instanciando ("carregando") um driver PostgreSQL

- No prompt do R, primeiro indique que o pacote RPostgreSQL será usado por meio da função `library()`:

```
> library(RPostgreSQL)
```

- Depois, crie uma instância do *driver* do PostgreSQL por meio da função `dbDriver()` (no exemplo, a instância foi chamada de `drv`):

```
> drv <- dbDriver("PostgreSQL")
```

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Criando uma conexão com um BD PostgreSQL

- No prompt do R, use a função `dbConnect()` para criar um objeto de conexão (no exemplo, a seguir, o nome atribuído a esse objeto é `con`):

```
> con <- dbConnect(drv, host="postgresql.linux.ime.usp.br",
                   port="5432", dbname="login_rede",
                   user="login_rede", password="senha")
```

- São parâmetros para a função `dbConnect` o *driver* para o SGBD que hospeda o BD ao qual se deseja conectar, mais as informações para a conexão com o banco

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Criando uma conexão com um BD PostgreSQL

- Podemos criar dentro de uma mesma sessão do R várias conexões (para um mesmo BD ou para BDs diferentes no mesmo servidor – quando o SGBD oferecer suporte para isso – ou BDs em servidores diferentes)
- Uma conexão fica aberta até que o usuário a encerre explicitamente ou até que a sessão R na qual ela foi criada seja encerrada
- Para listar todas as conexões gerenciadas por um objeto *driver* chamado `drv`, execute no prompt do R o seguinte comando:

```
> dbListConnections(drv)
```

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Submetendo consultas a um BD PostgreSQL

- Depois que uma conexão é aberta, podemos submeter consultas ao BD
- Algumas consultas são submetidas de forma explícita pelo usuário da sessão R
- Outras consultas são submetidas implicitamente por funções do R. Exemplos:
  - `dbListTables(con)` – devolve a lista dos nomes de tabelas encontrados na conexão `con`
  - `dbExistsTable(con,"NomeTabela")` – verifica se a tabela de nome "NomeTabela" existe na conexão `con` (o resultado é um valor booleano)
  - `dbListFields(con,"NomeTabela")` – devolve a lista dos nomes dos atributos da tabela "NomeTabela" na conexão `con`

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Submetendo consultas a um BD PostgreSQL

- Para recuperar todas as tuplas de uma tabela do BD e armazená-las em um objeto do tipo *data frame*, podemos usar a função `dbReadTable`, como mostrado a seguir:

```
> todosProdutos <- dbReadTable(con, "produto",
                               row.names = "modelo")
```

- O comando acima importa a tabela `Produto` do BD no R como um *data frame* chamado `todosProdutos`, usando o atributo `modelo` como `row.names` para o *data frame*

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Submetendo consultas a um BD PostgreSQL

- Podemos usar a função data `dbGetQuery` para submeter consultas SQL diretamente ao BD, como mostrado a seguir:

```
> todosPCs <- dbGetQuery(con,
                "SELECT modelo, preco, velocidade FROM pc;")
```

- O resultado da consulta do exemplo acima é importado no R como um *data frame* chamado `todosPCs`

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Submetendo consultas a um BD PostgreSQL

- Em algumas consultas, o conjunto de tuplas devolvido como resposta pode ser bem grande
- Para evitar que todas as tuplas sejam transferidas do BD para o R de uma só vez, é possível solicitar que as tuplas sejam trazidas em "lotes" (que podem ser analisados separadamente)
- A função `dbSendQuery` nos permite submeter uma consulta SQL ao BD, mas sem trazer a resposta da consulta em seguida. A resposta é recuperada por meio da função `fetch`. Exemplo:

```
> rs <- dbSendQuery(con, "SELECT * FROM produto;")
> lote1Produtos <- fetch(rs, n = 5)
> lote2Produtos <- fetch(rs, n = 10)
```

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Submetendo consultas a um BD PostgreSQL

- A função `dbSendQuery` devolve um objeto do tipo *result set*, que no exemplo do slide anterior foi atribuído à variável `rs`
- A partir de um objeto *result set*, podemos obter todas as informações sobre o resultado da consulta da qual ele resultou:
  - consulta SQL de origem
  - nomes dos atributos na resposta
  - número de tuplas na resposta
  - quantas tuplas já foram recuperadas do resultado ("fetched")
- A função `dbGetInfo(rs)` mostra as informações do *result set* `rs`

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Submetendo consultas a um BD PostgreSQL

- A função `fetch` recupera a partir de um objeto *result set* um novo lote de tuplas ainda não recuperadas do resultado e o devolve como um objeto do tipo *data frame*
- No exemplo abaixo, o primeiro `fetch` devolve as primeiras 5 tuplas da resposta ao comando SQL submetido ao BD, enquanto o segundo `fetch` devolve as 10 tuplas seguintes:

```
> rs <- dbSendQuery(con, "SELECT * FROM Produto;")
> lote1Produtos <- fetch(rs, n = 10)
> lote2Produtos <- fetch(rs, n = 15)
```

- A execução de um `fetch` com $n = -1$ faz com que todas as tuplas ainda não recuperadas no *result set* sejam recuperadas e carregadas no *data frame*

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Outras funções relacionadas a um objeto result set

- A função `dbGetStatement(rs)` devolve a consulta SQL associada ao *result set* `rs`
- A função `dbColumnInfo(rs)` devolve informações sobre os atributos que aparecem no conjunto resposta da consulta associada ao *result set* `rs`
- A função `dbGetRowCount(rs)` devolve o número total de tuplas no conjunto resposta da consulta associada ao *result set* `rs`

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Outras funções relacionadas a um objeto result set

- A função `dbGetRowsAffected(rs)` devolve o número de tuplas afetadas pela consulta associada ao *result set* `rs`; -1 indica que nenhuma tupla foi afetada
  (Só pode ser diferente de -1 quando a consulta é na verdade um comando de modificação do BD)
- A função `dbHasCompleted(rs)` devolve um valor booleano que indica se todas as tuplas do conjunto resposta da consulta associada ao *result set* `rs` já foram recuperadas ("fetched")
- A função `dbClearResult(rs)` "encerra" o *result set* `rs` sem carregar nenhuma de suas tuplas ainda não recuperadas
  (Obs: depois da execução dessa função, nenhuma operação pode ser realizada sobre o *result set*)

Outras funções do RPostgreSQL

- A função `dbGetException(con)` mostra informações sobre o status da execução do último comando submetido pela conexão `con`. Caso a execução do último comando tenha gerado um erro, a função mostrará os detalhes sobre o erro (o seu código, a mensagem explicativa, etc.)
- A função `dbRemoveTable(con,"NomeTabela")` remove a tabela de nome "NomeTabela" se ela existir na conexão `con`. A função devolve um valor booleano indicando se a remoção foi feita com sucesso
- A função `dbWriteTable(con,"NovaTabela",dados)` grava as tuplas contidas no *data frame* `dados` em uma nova tabela chamada "NovaTabela" na conexão `con`. A função devolve um valor booleano indicando se a gravação foi feita com sucesso

Outras funções do RPostgreSQL

- As funções `dbGetInfo(objeto)` e `summary(objeto)` mostram informações sobre um objeto de banco de dados (um *driver*, uma conexão ou um *result set*). Exemplos:

```
> dbGetInfo(drv)
> summary(con)
```

Outras funções do RPostgreSQL

Ao final de um acesso a um BD, é sempre bom liberar os objetos usados no acesso:

- Para encerrar uma conexão `con`:

  ```
  > dbDisconnect(con)
  ```

- Para liberar os recursos usados por um *driver* `dvr`:

  ```
  > dbUnloadDriver(drv)
  ```

Estratégia 2: Acessando um BD PostgreSQL a partir do R

Exemplo

No Paca, junto com estes slides, há um *script* R com um exemplo completo de acesso a um BD PostgreSQL.

Referências Bibliográficas

- Site do RPostgreSQL: `https://cran.r-project.org/web/packages/RPostgreSQL/index.html`
- Documentação completa do pacote DBI: `http://cran.r-project.org/web/packages/DBI/DBI.pdf`
- "Accessing a Database from R" – material da disciplina "Stat 133: Concepts in Computing with Data" da Universidade de Berkeley `http://www.stat.berkeley.edu/~nolan/stat133/Fall05/lectures/SQL-R.pdf`