

Bancos de dados relacionais: MySQL

Aula 13

Curso: BIG863 - Basic Python Programming for Ecologists

Professora: Dra. Cecilia F. Fiorini

Supervisor: Prof. Dr. Fernando A. O. Silveira

<https://meet.google.com/zdi-ueoz-nsr>, 7 de junho de 2023



Roteiro*

- 1 RDBMS
- 2 MySQL
- 3 Interagindo com o MySQL a partir do Python
- 4 Explorando dados com o SQL

*Conteúdo adaptado a partir do livro "Practical Computing for Biologists" de Steven H. D. Haddock e Casey W. Dunn

Gerenciamento de dados

- O uso pervasivo de planilhas pode dar a falsa impressão de que é melhor armazenar todos os tipos de dados em grades de duas dimensões.
- Contudo, se você tiver um conjunto de dados complexo para diversos elementos distintos porém relacionados de estudo, talvez não seja possível armazená-los em uma única tabela.
- Dados bancos de dados relacionais (RDBMS) podem ser úteis para acessar dados que estão organizados desta forma.

RDBMS

- RDBMS são coleções de informações estruturadas.
- Nestas, os dados estão organizados em tabelas com as quais o usuário não interage diretamente, pois o sistema de gerenciamento age como intermediário.
- O conceito que conduz os RDBMS é que cada pedaço de informação é armazenado apenas uma vez e, em seguida, vinculado por meio de relações a outros dados, em vez de copiados.
- Isso facilita as atualizações, reduz a chance de inconsistências e torna o banco de dados mais eficiente em termos de memória e computação.
- A maioria dessas vantagens se torna mais aparente para projetos de análise maiores e mais complexos. Para projetos menores, os sistemas de banco de dados podem ser um exagero.

RDBMS

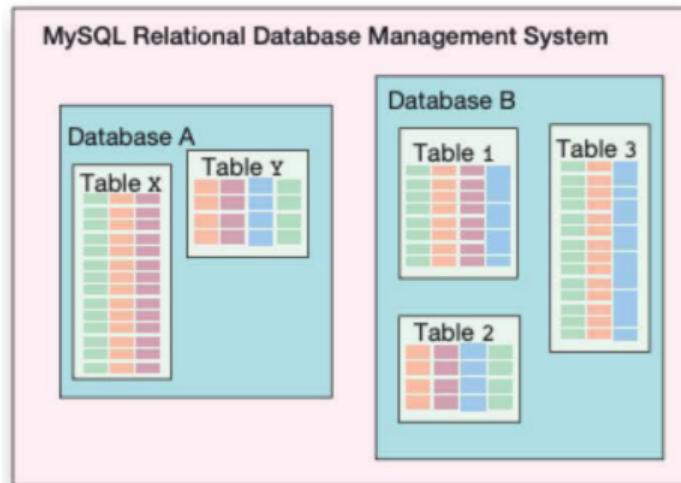
- Sistemas de gerenciamento comerciais são:
 - FileMaker
 - Microsoft Office Access
 - Microsoft SQL Server
- Opções opensource incluem:
 - MySQL
 - PostgreSQL
 - SQLite

SQL

- A interação com os sistemas de gerenciamento de bancos de dados modernos é feita através de uma linguagem chamada SQL.
- É possível interagir com os sistemas de gerenciamento de bancos de dados a partir de outros softwares, como Python, R e MATLAB.
 - As consultas (queries) SQL são construídas como strings e qualquer dado que é retornado pode ser manejado com estes softwares.

Bancos de dados

- Um RDBMS pode hospedar qualquer número de bancos de dados.
 - Cada banco de dados, por sua vez, contém tabelas bidimensionais que armazenam os dados reais.
 - Cada coluna em uma tabela tem um tipo de dado diferente, e cada linha contém um registro.



Bancos de dados

- O tipo de dados em cada coluna de uma tabela do banco de dados deve ser especificado.
- Um erro ocorrerá se você tentar adicionar dados que não estejam em conformidade com o tipo especificado.

TABLE 15.1 Common RDBMS data types

Data type	Description
INTEGER	An integer ranging in value from -2147483648 to 2147483647; INT can be used as an abbreviation for INTEGER
FLOAT	A floating point number, including scientific notation: 3.14159 or 6.022e+23
DATE	A date in 'YYYY-MM-DD' format
DATETIME	A date and time in 'YYYY-MM-DD HH:MM:SS' format
TEXT	A string containing up to 65535 characters
TINYTEXT	A string containing up to 255 characters
BLOB	A piece of information encoded in binary, including images or other non-text data; there are four sizes of blob data types, with different storage capacities

Bancos de dados

- Ao criar uma nova tabela, uma das colunas deve ser especificada como a chave primária, e cada linha da tabela deve ter um valor de chave primária exclusivo que distingue aquele registro ou linha.
- A chave primária geralmente é um valor inteiro que é automaticamente calculado pelo sistema de gerenciamento de banco de dados e armazenado quando uma nova linha é adicionada.

MySQL

- O MySQL é um dos sistemas de gerenciamento de banco de dados mais populares, tendo sido criado pela Oracle Corporation.
- Pode ser acessado pela interface gráfica MySQL Workbench ou, de forma mais conveniente, pela linha de comando.
- Podemos acessar um servidor local de MySQL com o seguinte comando:

```
lucy$ mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1426
Server version: 5.1.48 MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input.
```

Comandos SQL

- Comandos SQL terminam com um ponto e vírgula.
- Pressionando Enter sem um ponto e vírgula, você pode dividir um comando longo em várias linhas.
- Outra convenção é que palavras especiais, como funções e variáveis, geralmente são apresentadas em LETRAS MAIÚSCULAS.

Função DATABASES

- Por exemplo, a função DATABASES mostra os conjuntos de dados que estão presentes no servidor:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| test |
+-----+
3 rows in set (0.00 sec)
```

Comando CREATE DATABASE

- Ao iniciar um projeto, o primeiro passo é criar um banco de dados vazio usando o comando CREATE DATABASE:

```
mysql> CREATE DATABASE midwater;
Query OK, 1 row affected (0.09 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| midwater |
| mysql |
| test |
+-----+
4 rows in set (0.00 sec)
```

Comando USE

- Como o servidor é responsável por vários bancos de dados, é necessário especificar com qual deseja trabalhar. O comando USE seleciona um banco de dados:

```
mysql> USE midwater;  
Database changed  
mysql>
```

- Você pode alternar para outro banco de dados a qualquer momento emitindo outro comando USE.

Criando uma tabela

- Após criar um banco de dados vazio e selecioná-lo com o comando USE, o próximo passo é começar a criar tabelas.
- Tabelas são unidades de organização de dados bidimensionais, um tanto equivalentes a planilhas.
- Você pode criar uma tabela sem registros de dados, mas deve haver pelo menos uma coluna (também chamada de campo) para começar.
- Você sempre pode adicionar e remover colunas posteriormente, mas é melhor antecipar as necessidades da tabela e criar todas as colunas desde o início.

Criando uma tabela

- O primeiro campo a considerar ao projetar uma tabela é a chave primária, uma coluna especial que contém um valor que identifica exclusivamente cada linha.
- Por convenção, é melhor fazer da primeira coluna a chave primária e usar uma série de números inteiros únicos como entradas.
- É comum permitir que o banco de dados gere automaticamente um valor para a chave primária quando cada linha é criada, para que o valor não corresponda a um valor já presente no arquivo de entrada.

Criando uma tabela

- Os campos de cada coluna são delimitados por tabulação.
- Há uma linha de cabeçalho que descreve o conteúdo de cada coluna de dados. É importante rotular colunas com nomes informativos.
- Por convenção, a chave primária de uma tabela muitas vezes recebe um nome semelhante ao da própria tabela, combinado com o sufixo "`_id`".

Criando uma tabela

- Por exemplo, criaremos a tabela specimens:

Column name	Type
specimen_id	INTEGER ← This will be the primary key
vehicle	TINYTEXT
dive	INTEGER
date	DATE
lat	FLOAT
lon	FLOAT
depth	FLOAT
notes	TEXT

Criando uma tabela

- O comando `CREATE TABLE` é seguido pelo nome da tabela que você deseja criar e, em seguida, entre parênteses, as informações sobre cada coluna, separadas por vírgulas.
- Essas informações incluem o nome do campo, seu tipo e, opcionalmente, algumas informações adicionais.
- No caso do exemplo, a única coluna com parâmetros adicionais é a chave primária `specimen_id`:
 - `NOT NULL` indica que nenhuma linha pode estar sem um valor.
 - `AUTO_INCREMENT` especifica que um novo número inteiro único será automaticamente inserido cada vez que uma linha for adicionada.
 - `PRIMARY KEY` especifica que esta coluna é a chave primária.

Criando uma tabela

- Para criar tal tabela, utilizaremos os seguinte comando:

```
mysql> CREATE TABLE specimens (
    -> specimen_id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> vehicle TINYTEXT,
    -> dive INTEGER,
    -> date DATE,
    -> lat FLOAT,
    -> lon FLOAT,
    -> depth FLOAT,
    -> notes TEXT
    -> );
Query OK, 0 rows affected (0.10 sec)
```

Criando uma tabela

- A função SHOW TABLES mostra uma lista de tabelas disponíveis no banco de dados que encontra-se ativo:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_midwater |
+-----+
| specimens           |
+-----+
1 row in set (0.00 sec)
```

Criando uma tabela

- O comando DESCRIBE, seguido pelo nome da tabela, fornece um resumo da estrutura de uma tabela:

```
mysql> DESCRIBE specimens;
+-----+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| specimen_id | int(11)    | NO   | PRI | NULL    | auto_increment |
| vehicle     | tinytext   | YES  |     | NULL    |                 |
| dive        | int(11)    | YES  |     | NULL    |                 |
| date         | date       | YES  |     | NULL    |                 |
| lat          | float      | YES  |     | NULL    |                 |
| lon          | float      | YES  |     | NULL    |                 |
| depth        | float      | YES  |     | NULL    |                 |
| notes        | text        | YES  |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
8 rows in set (0.01 sec)
```

Inserindo dados manualmente em uma tabela

- Uma nova linha de dados é adicionada com o comando INSERT:

```
mysql> INSERT INTO specimens SET
-> vehicle='Tiburon',
-> dive=596,
-> date='2003-07-03',
-> lat=36.602,
-> lon=-122.375,
-> depth=1190,
-> notes='holotype';
Query OK, 1 row affected (0.01 sec)
```

Inserindo dados manualmente em uma tabela

- Para examinar o conteúdo da sua tabela até agora, utilize o comando SELECT com a seguinte sintaxe:

```
mysql> SELECT * FROM specimens;
+-----+-----+-----+-----+-----+-----+-----+
| specimen_id | vehicle | dive | date      | lat    | lon     | depth   | notes   |
+-----+-----+-----+-----+-----+-----+-----+
|       1 | Tiburon | 596 | 2003-07-03 | 36.602 | -122.375 | 1190    | holotype |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Inserindo dados manualmente em uma tabela

- O comando SELECT é muito poderoso. É o comando SELECT que posteriormente nos permitirá combinar dados entre tabelas e visualizar subconjuntos específicos de dados.
- Aqui, porém, o * indica que ele deve mostrar todas as colunas de dados na tabela, e a falta de outras cláusulas para refinar a busca faz com que ele exiba todas as linhas da tabela.

Inserindo dados automaticamente em uma tabela

- É possível automatizar interações com o banco de dados de diversas maneiras.
- É possível também decompor e realizar cálculos com os dados antes de carregá-los no banco de dados.
 - Remover código desnecessário de um arquivo KML.

Inserindo dados automaticamente em uma tabela

- Após a formatação adequada de todos os campos, os dados podem ser preparados para inserção no banco de dados.
- Cada linha de dados será adicionada ao banco de dados com um comando INSERT INTO.
- É recomendável imprimirá o comando SQL na tela antes de adicionar o código final para conectar-se ao banco de dados. Isso permite que você identifique problemas potenciais antes que erros difíceis de corrigir no banco de dados sejam cometidos.

MySQLdb

- O módulo de Python "MySQLdb" é capaz de gerir os bastidores da interação entre o Python e o MySQL.
- A parte mais complicada de conectar-se ao MySQL a partir do Python é instalar este módulo, mas você só precisa fazer isso uma vez.
- É uma boa ideia testar se o módulo está carregado corretamente usando o prompt interativo do Python antes de tentar usá-lo em um programa:

```
host:MySQL lucy$ cd ~/pcfbl/examples/ctd
host:ctd lucy$ python
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import MySQLdb
>>>
```

MySQLdb

- Depois de instalar o MySQLdb no seu computador, você ainda precisa importá-lo para o seu programa Python para usá-lo.

```
import MySQLdb
```

- Em seguida, crie uma conexão com o banco de dados MySQL. Você pode criar uma única conexão com o banco de dados no início do seu programa e usá-la sempre que desejar.

```
MyConnection = MySQLdb.connect( host = "localhost", \
user= "root", passwd = "", db = "midwater")
MyCursor = MyConnection.cursor()
```

MySQLdb

- Aqui criamos o objeto de conexão real, MyConnection. Ele precisa de informações sobre o endereço de rede do servidor MySQL como nome de usuário, senha e o banco de dados no servidor ao qual você deseja se conectar.
- Uma vez que você tenha o objeto de conexão, você o usa para criar um objeto de cursor. Você pode pensar neste cursor de banco de dados como o cursor da linha de comando, o qual é usado para enviar comandos e recuperar resultados.
- No final do programa, adicione duas linhas para fechar o cursor e a conexão com o banco de dados:

```
MyCursor.close()  
MyConnection.close()
```

Executando comandos com o MySQLdb

- A execução do comando é apenas uma linha. Adicione-a imediatamente após a linha que imprime a variável SQL na tela:

```
MyCursor.execute(SQL)
```

- Esta linha executa a string SQL que você criou anteriormente, usando o cursor do banco de dados que você abriu antes do loop. A cada passagem pelo loop, ele executa um novo comando SQL e adiciona mais uma linha de dados ao banco de dados.
- Acesse um exemplo de um script SQL em Python no seguinte link: mylatlon_4.py

Comando SELECT COUNT(*)

- Aqui estão mais alguns usos do SELECT e maneiras de refinar a saída. Uma pergunta básica sobre um banco de dados é: "Quantas linhas minha tabela contém?" Podemos usar uma declaração SELECT para obter uma resposta. Substituindo o * por COUNT(*), você recupera uma única linha que contém a contagem do número de linhas recuperadas pelo SELECT, em vez de todas as linhas de dados em si.

```
mysql> SELECT COUNT(*) FROM specimens;
+-----+
| COUNT(*) |
+-----+
|      10 |
+-----+
1 row in set (0.00 sec)
```

Comando SELECT

- Também é possível extrair dados de colunas específicas com o SELECT. Em vez de usar *, que é um curinga para todas as colunas, você pode especificar as colunas desejadas, separando-as por vírgulas:

```
mysql> SELECT vehicle,date FROM specimens;
```

vehicle	date
Tiburon	2003-07-19
JSL II	1986-09-16
JSL II	1984-08-18
Ventana	1999-03-11
Ventana	2000-06-16
Ventana	2002-09-09
Tiburon	2002-11-24
Tiburon	2003-03-13
Tiburon	2003-03-31
JSL II	2003-09-26

```
10 rows in set (0.00 sec)
```

Comandos SELECT DISTINCT e GROUP BY

- Uma tarefa comum é ver quantos valores distintos uma determinada coluna possui. Isso pode ser feito de algumas maneiras diferentes.
 - O comando SELECT DISTINCT é o mais simples de digitar, mas não mostra quantas linhas existem para cada tipo de veículo.
 - Para obter a contagem para cada variável, use o comando alternativo que inclui a cláusula GROUP BY. Ele agrupa as linhas por valores de veículo compartilhados e, em seguida, conta o número de linhas em cada um desses grupos com COUNT(*) .

Comandos SELECT DISTINCT e GROUP BY

```
mysql> SELECT DISTINCT vehicle FROM specimens;
+-----+
| vehicle |
+-----+
| Tiburon |
| JSL II  |
| Ventana |
+-----+
3 rows in set (0.03 sec)

mysql> SELECT vehicle,COUNT(*) FROM specimens GROUP BY vehicle;
+-----+-----+
| vehicle | COUNT(*) |
+-----+-----+
| JSL II  |      3 |
| Tiburon |      4 |
| Ventana |      3 |
+-----+-----+
3 rows in set (0.21 sec)
```

Operações matemáticas e estatísticas

- O SQL pode realizar operações matemáticas e estatísticas nos dados que são recuperados:

TABLE 15.2 Selected SQL math and statistical operators and functions

Function or operator	Meaning
+, -, *, /	Basic math operators
AVG	Average of the values
COUNT	Count of the values
MAX	Maximum value
MIN	Minimum value
STD	Standard deviation
SUM	Sum of the values

Operações matemáticas e estatísticas

- Para usar esses operadores, construa uma fórmula entre parênteses, conectando os nomes dos campos com símbolos matemáticos, como `(depth * 3)`.
- Você também pode usar as funções estatísticas colocando um nome de campo entre parênteses após o nome do parâmetro.
- Por exemplo, para obter a profundidade média para mergulhos agrupados por cada veículo, você pode usar o seguinte comando:

```
mysql> SELECT vehicle, AVG(depth) FROM specimens GROUP BY vehicle;
+-----+-----+
| vehicle | avg(depth) |
+-----+-----+
| JSL II  | 688.666666666667 |
| Tiburon | 1154          |
| Ventana | 900.666666666667 |
+-----+-----+
3 rows in set (0.06 sec)
```

Cláusula WHERE

- Além de isolar colunas específicas, você também pode isolar linhas específicas de uma tabela. Isso é feito com a cláusula WHERE:

```
mysql> SELECT * FROM specimens WHERE vehicle='Tiburon';
+-----+-----+-----+-----+-----+-----+-----+-----+
| specimen_id | vehicle | dive | date | lat | lon | depth | notes |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 4 | Tiburon | 596 | 2003-07-19 | 36.602 | -122.375 | 1190 | holotype |
| 10 | Tiburon | 515 | 2002-11-24 | 36.7 | -122.033 | 1156 | |
| 11 | Tiburon | 531 | 2003-03-13 | 24.317 | -109.203 | 1144 | |
| 12 | Tiburon | 547 | 2003-03-31 | 24.234 | -109.667 | 1126 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

- Observe que o operador de igualdade no SQL é um único sinal de igual (=), e não == como em Python e muitas outras linguagens.

Cláusula WHERE

- A cláusula WHERE é muito flexível e você pode usá-la com correspondências aproximadas usando LIKE ou até mesmo expressões regulares com REGEXP.
 - Em vez de testar uma correspondência exata com uma string usando =, você pode usar LIKE para recuperar correspondências a uma parte da string. LIKE usa % como um caractere curinga da mesma maneira que * é usado no shell bash e como * é usado em expressões regulares:

```
mysql> SELECT vehicle,dive FROM specimens WHERE vehicle LIKE 'TIB%';
+-----+-----+
| vehicle | dive |
+-----+-----+
| Tiburon | 596 |
| Tiburon | 515 |
| Tiburon | 531 |
| Tiburon | 547 |
+-----+-----+
```

Cláusula WHERE

- Para realizar uma pesquisa de expressão regular, use WHERE field REGEXP query, e query é uma string contendo o termo de pesquisa. A sintaxe de expressão regular SQL não inclui todos os curingas como e , mas você pode usar a notação com colchetes (e.g., [A-z]):

```
mysql> SELECT vehicle,dive FROM specimens WHERE vehicle REGEXP '^V';
+-----+-----+
| vehicle | dive |
+-----+-----+
| Ventana | 1575 |
| Ventana | 1777 |
| Ventana | 2243 |
+-----+-----+
```

Cláusula WHERE

- Expressões construídas com o WHERE também usam operadores de comparação numérica e declarações lógicas com frequência. Este é um dos usos mais comuns de uma declaração SELECT. Por exemplo:

```
mysql> SELECT vehicle,dive FROM specimens WHERE dive < 1000;
+-----+-----+
| vehicle |  dive |
+-----+-----+
| Tiburon |    596 |
| Tiburon |    515 |
| Tiburon |    531 |
| Tiburon |    547 |
| JSL II  |   930 |
+-----+-----+
```

AND e OR

- Se você construir uma sequência lógica de comparações, certifique-se de pensar no uso de AND e OR.
- Duas comparações ligadas por um OR retornarão o conjunto combinado de valores quando esses testes forem verdadeiros. Por exemplo:

```
SELECT vehicle, dive from specimens
WHERE vehicle LIKE "Tib%" OR vehicle LIKE "JSL%";
```

- Usaremos WHERE com AND para retornar um subconjunto desejado.

UPDATE

- Depois que os dados forem carregados em um banco de dados, você frequentemente desejará modificá-los.
- O comando UPDATE pode alterar as tabela. Por exemplo:

```
mysql> UPDATE ctd SET vehicle='TIBURON' WHERE vehicle='tibr';
Query OK, 3085 rows affected (0.07 sec)
Rows matched: 3085  Changed: 3085  Warnings: 0

mysql> UPDATE ctd SET vehicle='VENTANA' WHERE vehicle='vnta';
Query OK, 653 rows affected (0.05 sec)
Rows matched: 653  Changed: 653  Warnings: 0
```

- Aqui, a cláusula WHERE restringe o comando a um subconjunto de linhas em que os critérios especificados são verdadeiros. Já cláusula SET atribui um valor específico a um campo específico.