

Aprendendo a trabalhar com MySQL

Esse tutorial tem objetivo de ajudar VOCÊ à trabalhar com o MySQL, para um melhor entendimento, vamos utilizar um ambiente já configurado na WEB (o professor passará para você os procedimentos para realizar o acesso).

Antes de qualquer coisa, vamos fazer o download do MySQL e vamos entender o que ele é.

O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Structured Query Language ou Linguagem de Consulta Estruturada) como interface.

O MySQL atualmente é um dos maiores SGBD's do mundo, com mais de 10 milhões de instalações e vem sendo usado inclusive em projetos de grande porte em grandes empresas.

Entre essas empresas, estão: NASA, Friendster, Banco Bradesco, Dataprev, HP, Nokia, Sony, Lufthansa, U.S. Army, U.S. Federal Reserve Bank, Associated Press, Alcatel, Slashdot, Cisco Systems, Google, entre outros.

Não é a toa que o MySQL cresceu tanto assim nos últimos anos, abaixo segue uma lista de coisas que fizeram esse SGBD crescer tanto e continuar crescendo cada vez mais.

Portabilidade (suporta praticamente qualquer plataforma atual);

Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, C#, Visual Basic, Python, Perl, PHP, ASP e Ruby

Excelente desempenho e estabilidade;

Pouco exigente quanto a recursos de hardware;

Facilidade de uso;

É um Software Livre com base na GPL (entretanto, se o programa que acessar o Mysql não for GPL, uma licença comercial deverá ser adquirida);

Contempla a utilização de vários Storage Engines como MyISAM, InnoDB, Falcon, BDB, Archive, Federated, CSV, Solid...

Suporta controle transacional;

Suporta Triggers;

Suporta Cursors (Non-Scrollable e Non-Updatable);

Suporta Stored Procedures e Functions;

Replicação facilmente configurável;

Agora vamos entender um pouco a sintaxe do mysql.

Primeiro vamos ver como fazemos para criar um banco de dados pelo terminal do mysql. Porém já temos nosso banco de dados criado, mas fica abaixo a referência de como cria-lo por exemplo em sua casa para a prática de atividades.

Criando o banco de dados

Listagem 1: Criando banco de dados

```
CREATE DATABASE bancodeteste;
```

Como podemos ver, a sintaxe é bem intuitiva e com um leve conhecimento da língua inglesa fica mais fácil ainda de entender.

Após criar o bando de dados, precisamos avisar ao mysql que vamos usá-lo, para isso basta escrevermos.

Listagem 2: Usando o banco de dados

```
USE bancodeteste;
```

Feito isso o nosso banco de dados está criado, faltando apenas criar a nossa tabela. Para isso vamos usar o comando CREATE TABLE.

Criando tabelas

Listagem 3: Criando tabelas

```
CREATE TABLE fornecedores(  
codigo int(4) AUTO_INCREMENT,  
nome varchar(30) NOT NULL,  
email varchar(50),  
PRIMARY KEY (codigo)  
);
```

Como podemos ver, nossa tabela está criada.



```
Database changed  
mysql> CREATE TABLE fornecedores(codigo int(4) auto_increment, nome varchar(30)  
NOT NULL, email varchar(50), PRIMARY KEY(codigo));  
Query OK, 0 rows affected (0.14 sec)  
  
mysql> _
```

Figura 1: Tabela criada.

Agora vamos às explicações do que se tratam os comandos:

UC2: IMPLEMENTAR BANCO DE DADOS

AUTO_INCREMENT pode ser utilizado para automatizar um código que sirva de chave primária de uma tabela.

PRIMARY KEY define a chave primária da tabela, isto é, o campo que serve como chave da tabela e que não pode ser repetido.

NOT NULL define que um determinado campo seja de preenchimento obrigatório.

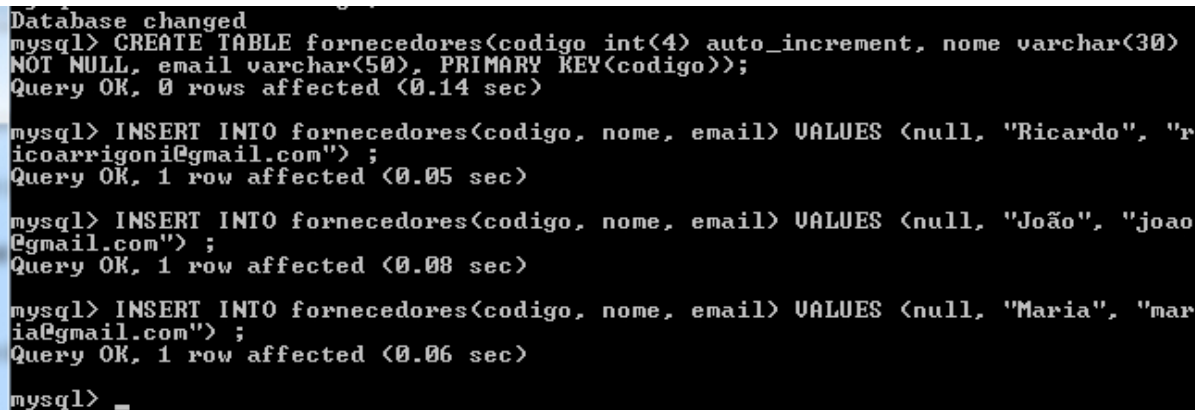
Agora já temos um banco de dados e uma tabela criada, com isso é possível manipular os dados do banco de dados.

INSERT

Agora vamos inserir alguma informação na nossa tabela, para isso vamos usar o comando INSERT, é bem simples também.

Listagem 4: Inserindo Dados

```
INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Ricardo",  
"ricoarrigoni@gmail.com") ;  
INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "João",  
"joao@gmail.com") ;  
INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Maria",  
"maria@gmail.com") ;
```



```
Database changed  
mysql> CREATE TABLE fornecedores(codigo int(4) auto_increment, nome varchar(30)  
NOT NULL, email varchar(50), PRIMARY KEY(codigo));  
Query OK, 0 rows affected (0.14 sec)  
  
mysql> INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Ricardo", "r  
icoarrigoni@gmail.com") ;  
Query OK, 1 row affected (0.05 sec)  
  
mysql> INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "João", "joao  
@gmail.com") ;  
Query OK, 1 row affected (0.08 sec)  
  
mysql> INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Maria", "mar  
ia@gmail.com") ;  
Query OK, 1 row affected (0.06 sec)  
  
mysql> _
```

Figura 2: Dados inseridos na tabela.

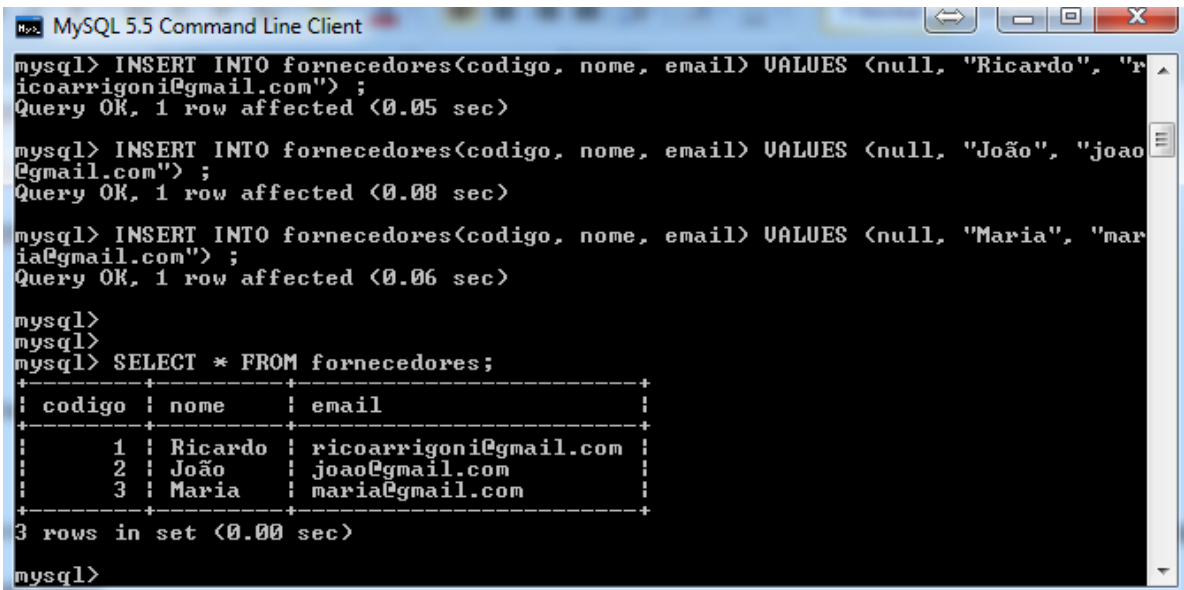
SELECT

Agora que nossa tabela está com alguns registros inseridos nela, nós vamos usar o comando SELECT pra poder selecionar e buscar esses registros. Continuaremos no mesmo terminal e iremos digitar o seguinte código:

Listagem 5: Usando o comando SELECT

```
SELECT * FROM fornecedores;
```

O Resultado desse SELECT nós vemos na Figura 5.

UC2: IMPLEMENTAR BANCO DE DADOS


```
mysql> INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Ricardo", "ricoarrigoni@gmail.com");
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "João", "joao@gmail.com");
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO fornecedores(codigo, nome, email) VALUES (null, "Maria", "maria@gmail.com");
Query OK, 1 row affected (0.06 sec)

mysql>
mysql>
mysql> SELECT * FROM fornecedores;
+----+-----+-----+
| codigo | nome  | email                |
+----+-----+-----+
| 1      | Ricardo | ricoarrigoni@gmail.com |
| 2      | João   | joao@gmail.com        |
| 3      | Maria  | maria@gmail.com       |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Figura 5:

Resultado do SELECT global.

Como podemos perceber, todos os registros da tabela foram retornados. Isso se deu porque o uso do SELECT * faz com que a consulta retorne todos os valores da tabela.

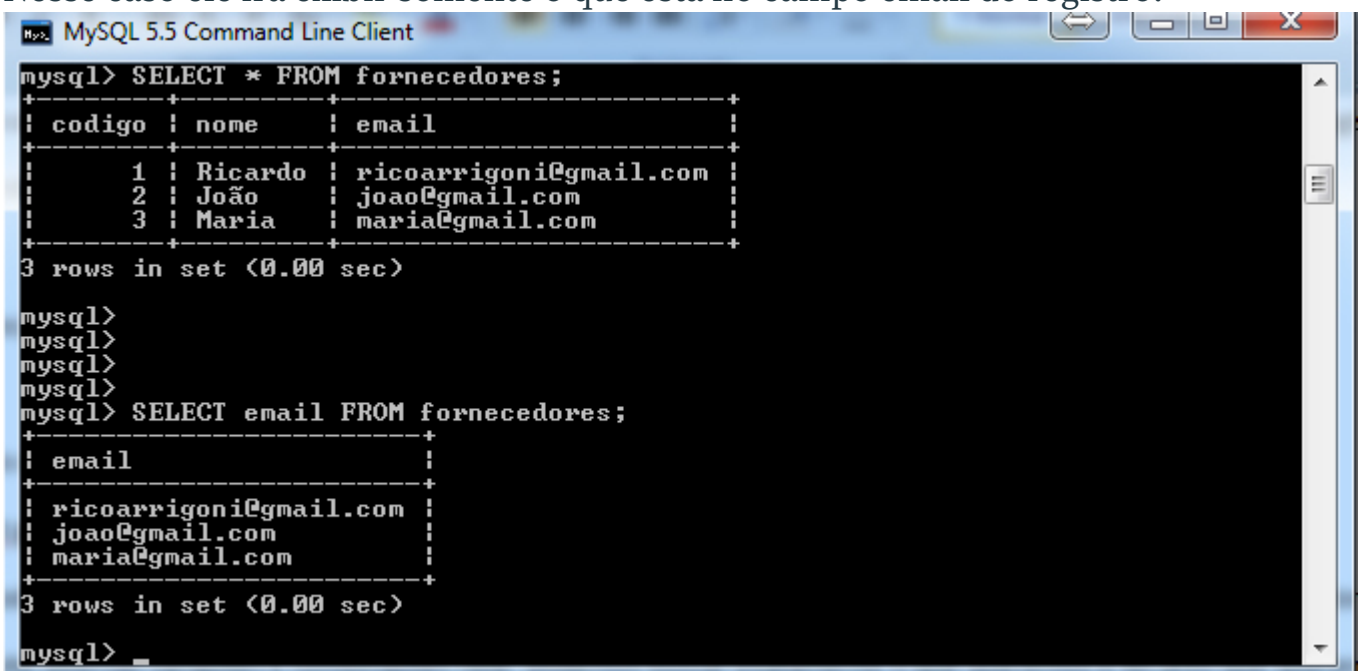
Mas o comando SELECT permite diversas variações e combinações nele, podemos buscar exatamente o que queremos e do jeito que queremos, por exemplo.

Se quisermos buscar apenas o e-mail do fornecedor nós podemos, basta fazer da seguinte forma:

Listagem 6: Usando o `SELECT` específico

```
SELECT email FROM fornecedores;
```

Nesse caso ele irá exibir somente o que está no campo email do registro.



```
mysql> SELECT * FROM fornecedores;
+----+-----+-----+
| codigo | nome  | email                |
+----+-----+-----+
| 1      | Ricardo | ricoarrigoni@gmail.com |
| 2      | João   | joao@gmail.com        |
| 3      | Maria  | maria@gmail.com       |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql>
mysql>
mysql>
mysql> SELECT email FROM fornecedores;
+-----+
| email                |
+-----+
| ricoarrigoni@gmail.com |
| joao@gmail.com        |
| maria@gmail.com       |
+-----+
3 rows in set (0.00 sec)

mysql>
```

Figura 3: `SELECT` específico.

ORDER BY

Se em nossa consulta sql nós quisermos que os registros retornados venham ordenados, nós podemos usar o comando ORDER BY, basta dizer pelo que você quer ordenar que ele traz o registro ordenado.

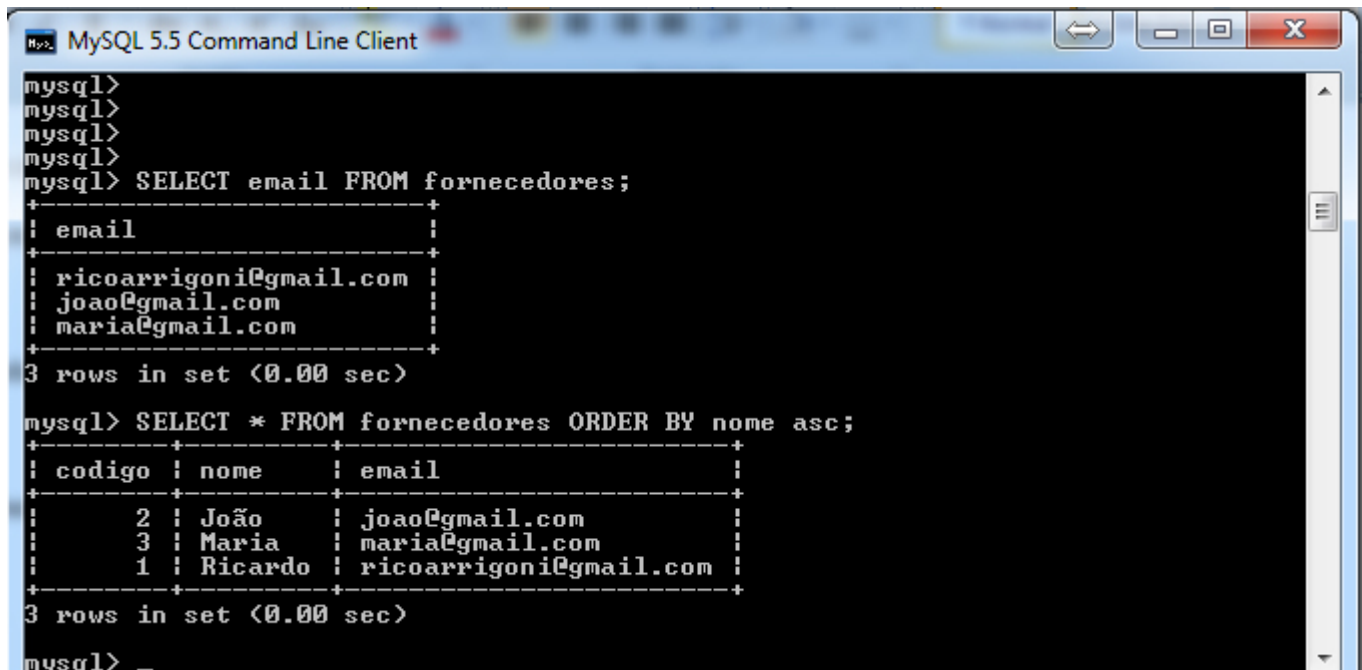
Listagem 7: SELECT usando ORDER BY

```
SELECT * FROM fornecedores ORDER BY nome asc;
```

Dessa forma, os registros retornados virão ordenados pelo campo nome em ordem alfabética. Mas como eu sei que virá em ordem alfabética?

Quando eu uso o termo “asc” eu digo que quero em ordem ascendente, se eu quiser fazer um SELECT que venha em ordem descendente, basta utilizarmos o “desc”.

O resultado da nossa busca pode ser visto aqui:



```
mysql>
mysql>
mysql>
mysql> SELECT email FROM fornecedores;
+-----+
| email |
+-----+
| ricoarrigoni@gmail.com |
| joao@gmail.com         |
| maria@gmail.com        |
+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM fornecedores ORDER BY nome asc;
+-----+-----+-----+
| codigo | nome   | email |
+-----+-----+-----+
| 2      | João  | joao@gmail.com |
| 3      | Maria | maria@gmail.com |
| 1      | Ricardo | ricoarrigoni@gmail.com |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Figura 4: SELECT ordenado por ordem alfabética.

UPDATE

Para alterar um registro, usamos o comando UPDATE, com ele é possível editar os campos de sua tabela e colocar outro valor neles.

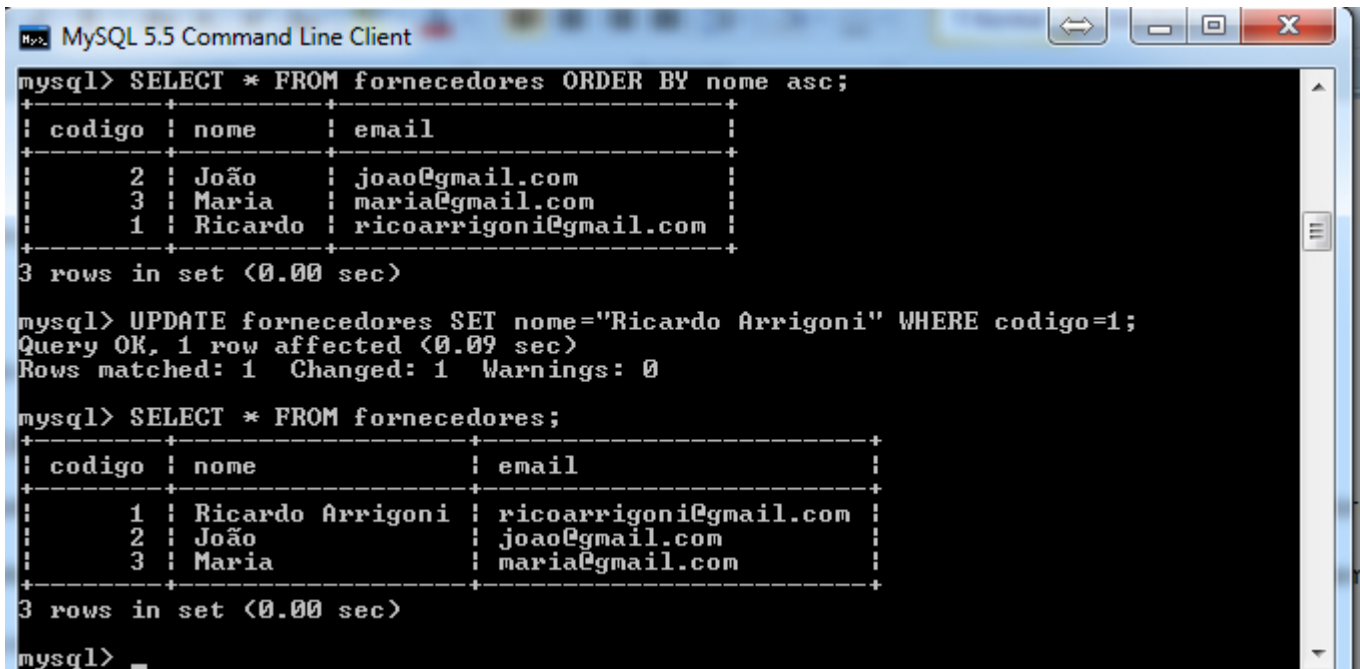
Listagem 8: Editando registros

```
UPDATE fornecedores SET nome="Ricardo Arrigoni" WHERE codigo=1;
```

Após esse comando, o nome “Ricardo” será alterado para “Ricardo Arrigoni”, vamos novamente dar um SELECT na tabela para poder ver o registro alterado

Listagem 9: `SELECT` para ver os dados atualizados

```
SELECT * FROM fornecedores;
```



```
mysql> SELECT * FROM fornecedores ORDER BY nome asc;
+----+-----+-----+
| codigo | nome      | email                |
+----+-----+-----+
| 2      | João     | joao@gmail.com       |
| 3      | Maria    | maria@gmail.com      |
| 1      | Ricardo  | ricoarrigoni@gmail.com |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> UPDATE fornecedores SET nome="Ricardo Arrigoni" WHERE codigo=1;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM fornecedores;
+----+-----+-----+
| codigo | nome      | email                |
+----+-----+-----+
| 1      | Ricardo Arrigoni | ricoarrigoni@gmail.com |
| 2      | João     | joao@gmail.com       |
| 3      | Maria    | maria@gmail.com      |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

Figura 5: Editando registros.

Como vimos no exemplo anterior, estamos usando uma cláusula chama de WHERE em nosso SQL, essa cláusula é responsável por definir qual registro específico da tabela vai ser afetado pelo comando SQL.

DELETE

Por último e não menos importante, vamos falar do comando DELETE. Ele é o responsável por remover todo e qualquer registro do bando de dados.

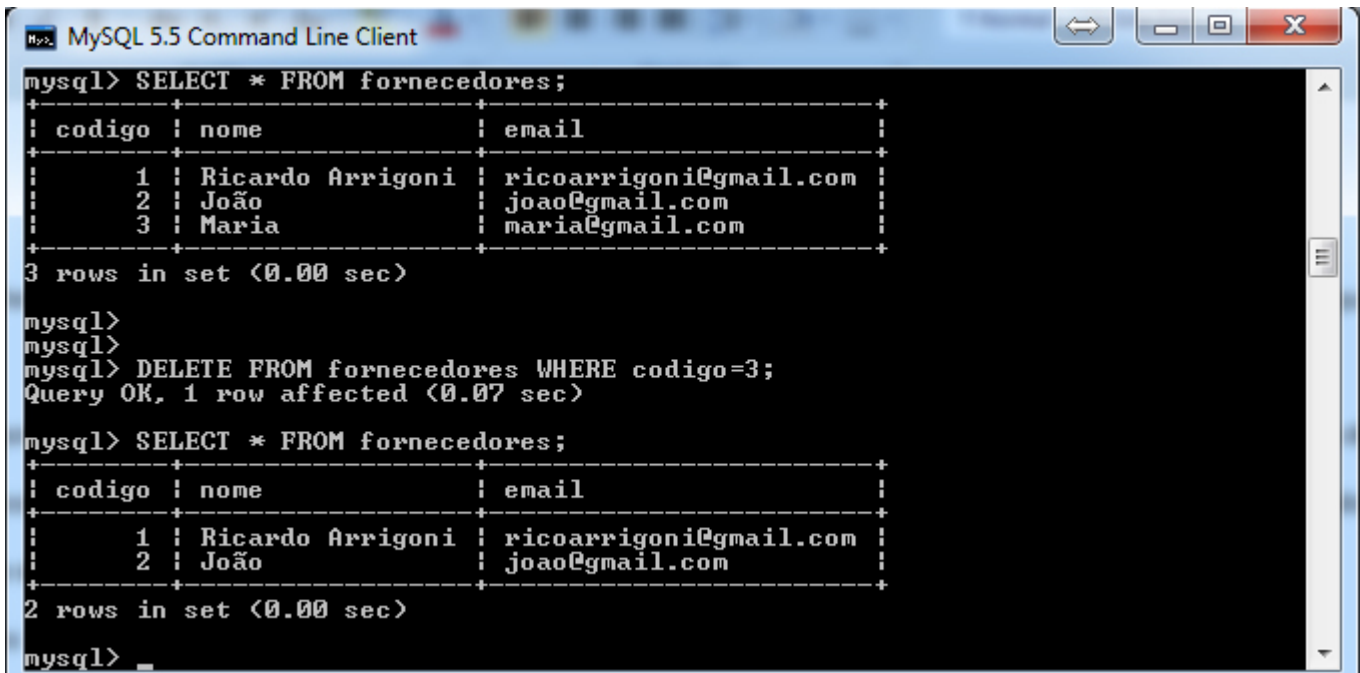
OBS : Uma vez executado, esse comando não é reversível, portanto tome bastante cuidado ao deletar algum registro de seu banco de dados.

Agora que já sabemos o que ele faz e que devemos tomar muito cuidado ao utilizá-lo, vamos ver como usar ele em nosso exemplo.

Listagem 10: Deletando dados

```
DELETE FROM fornecedores WHERE codigo=3;
```

Depois de executar esse comando o nosso registro de codigo=3 foi excluído do banco de dados e ele não poderá ser recuperado, ficamos então com apenas 2 registros em nossa tabela.



```
mysql> SELECT * FROM fornecedores;
+----+-----+-----+
| codigo | nome      | email                |
+----+-----+-----+
| 1      | Ricardo Arrigoni | ricoarrigoni@gmail.com |
| 2      | João        | joao@gmail.com        |
| 3      | Maria       | maria@gmail.com       |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql>
mysql>
mysql> DELETE FROM fornecedores WHERE codigo=3;
Query OK, 1 row affected (0.07 sec)

mysql> SELECT * FROM fornecedores;
+----+-----+-----+
| codigo | nome      | email                |
+----+-----+-----+
| 1      | Ricardo Arrigoni | ricoarrigoni@gmail.com |
| 2      | João        | joao@gmail.com        |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

Figura 6: Deletando dados da tabela.

Como pudemos ver, o MySQL é um banco de dados bem simples e fácil de se usar, além de ser bastante leve e o melhor de tudo, gratuito.

Tipos de dados numéricos no MySQL

O MySQL tem todos os tipos numéricos possíveis, o que inclui exatos, aproximados, inteiros, de ponto fixo, ponto flutuante etc. A lista, abaixo, mostra um resumo dos tipos de dados possíveis no MySQL:

TINYINT — número inteiro muito pequeno (tiny);

SMALLINT — número inteiro pequeno;

MEDIUMINT — número inteiro de tamanho médio;

INT — número inteiro de tamanho comum;

BIGINT — número inteiro de tamanho grande;

DECIMAL — número decimal, de ponto fixo;

FLOAT — número de ponto flutuante de precisão simples (32 bits);

DOUBLE — número de ponto flutuante de precisão dupla (64 bits);

BIT — um campo de um bit.

Tipos de dados em strings

Strings são cadeias de caracteres. No MySQL, uma string pode ter qualquer conteúdo, desde texto simples a dados binários – tais como imagens e arquivos. Cadeias de caracteres podem ser comparadas e ser objeto de buscas.

CHAR — uma cadeia de caracteres (string), de tamanho fixo e não-binária;

VARCHAR — uma string de tamanho variável e não-binária;

BINARY — uma string binária de tamanho fixo;

VARBINARY — uma string binária de tamanho variável;

BLOB — um BLOB (Binary Large Object – Objeto Grande Binário) pequeno;

TINYBLOB — um BLOB muito pequeno;

MEDIUMBLOB — um BLOB de tamanho médio;

LONGBLOB — um BLOB grande;

TINYTEXT — uma string não-binária e de tamanho bem reduzido;

TEXT — uma string não-binária e pequena;

MEDIUMTEXT — uma string de tamanho comum e não-binária;

LONGTEXT — uma string não-binária de tamanho grande;

ENUM — de acordo com o manual do MySQL, é uma string, com um valor que precisa ser selecionado de uma lista predefinida na criação da tabela;

SET — é um objeto que pode ter zero ou mais valores – cada um dos quais precisa ser escolhido de uma lista de valores predeterminados quando da criação da tabela.

Armazenamento de data e hora

Há várias opções para armazenar dados relacionados a data e hora. Se você quiser apenas armazenar o ano referente a um evento, pode usar o tipo YEAR. O tipo TIMESTAMP pode ser usado para acompanhar as mudanças ocorridas em um campo de uma tabela. Veja os tipos e suas descrições:

DATE — o valor referente a uma data no formato 'CCYY-MM-DD'. Por exemplo 1985-11-25 (ano-mês-dia). O 'CC' se refere aos dois dígitos do século (Century, em inglês);

TIME — um valor horário no formato 'hh:mm:ss' (hora:minutos:segundos);

TIMESTAMP — timestamp é uma sequência de caracteres ou informação codificada que identifica uma marca temporal ou um dado momento em que um evento ocorreu. No MySQL, ele tem o formato 'CCYY-MM-DD hh:mm:ss' – neste caso, seguem a padronização ISO 8601;

YEAR — armazena um ano no formato 'CCYY' ou 'YY';

Dados especiais

O MySQL tem suporte a tipos de dados que correspondem às classes OpenGIS.

Alguns destes carregam valores geométricos simples:

GEOMETRY

POINT

LINESTRING

POLYGON

O GEOMETRY pode armazenar qualquer tipo de valor geométrico. Os outros valores simples (POINT, LINESTRING e POLYGON) têm seus valores restritos aos tipos geométricos a que se referem.

Os outros, que seguem listados, carregam valores relativos a coleções/coletivos:

GEOMETRYCOLLECTION

MULTILINESTRING

MULTIPOINT

MULTIPOLYGON

Assim, GEOMETRYCOLLECTION pode armazenar coletâneas de objetos de qualquer tipo. Os outros tipos coletivos (MULTILINESTRING, MULTIPOLYGON e GEOMETRYCOLLECTION) restringem-se a cada forma geométrica particular.