

# MySQL

## PARA INICIANTES

Lui**z**Tools



# SUMÁRIO

<b>SOBRE O AUTOR</b>	3
<b>ANTES DE COMEÇAR</b>	7
<b>INTRODUÇÃO AO MYSQL</b>	9
Origem	10
Ambiente e Configuração	12
MySQL Workbench	13
<b>MYSQL NA PRÁTICA</b>	14
Criando nossa base de dados	17
O básico de SQL	22
<b>NODE.JS &amp; MYSQL</b>	28
Visual Studio Code	31
Node.Js +MySQL	34
<b>BOAS PRÁTICAS COM MYSQL</b>	45
Boas Práticas na Modelagem	47
Boas Práticas de Hardware	50
Boas Práticas de Performance	51
Boas Práticas de Segurança	56
<b>SEGUINDO EM FRENTE</b>	59

# **SOBRE O AUTOR**

---

Luiz Fernando Duarte Júnior é Bacharel em Ciência da Computação pela Universidade Luterana do Brasil (ULBRA, 2010) e Especialista em Desenvolvimento de Aplicações para Dispositivos Móveis pela Universidade do Vale do Rio dos Sinos (UNISINOS, 2013).

Carrega ainda um diploma de Reparador de Equipamentos Eletrônicos (SENAI, 2005), nove certificações em Métodos Ágeis de desenvolvimento de software por diferentes certificadoras (PSM-I, PSD-I, PACC-AIB, IPOF, ISMF, IKMF, CLF, DEPC, SFPC) e três certificações de coach profissional pelo IBC (Professional & Self Coach, Life Coach e Leader Coach).

Atuando na área de TI desde 2006, na maior parte do tempo como desenvolvedor, é apaixonado por desenvolvimento de software desde que teve o primeiro contato com a linguagem Assembly no curso de eletrônica. De lá para cá teve oportunidade de utilizar diferentes linguagens em diferentes sistemas, mas principalmente com tecnologias web, incluindo ASP.NET, JSP e, nos últimos tempos, Node.js.

**Foi amor à primeira vista e a paixão continua a crescer!**

Trabalhando com Node.js desenvolveu diversos projetos para empresas de todos os tamanhos, desde grandes empresas como Softplan até startups como Busca Acelerada e Só Famosos, além de ministrar palestras e cursos de Node.js para alunos do curso superior de várias universidades e eventos de tecnologia.

Um grande entusiasta da plataforma, espera que com esse livro possa ajudar ainda mais pessoas a aprenderem a desenvolver softwares com Node.js e aumentar a competitividade das empresas brasileiras e a empregabilidade dos profissionais de TI.

Além de viciado em desenvolvimento, como consultor em sua própria empresa, a DLZ Tecnologia e é autor do blog [www.luiztools.com.br](http://www.luiztools.com.br), onde escreve semanalmente sobre métodos ágeis e desenvolvimento de software, bem como mantenedor do canal [LuizTools](#), com o mesmo propósito.

Entre em contato, o autor está sempre disposto a ouvir e ajudar seus leitores.

## SOBRE O AUTOR



# MEUS CURSOS

## Curso online NODE.JS e MONGODB

[SAIBA MAIS...](#)

## Curso online Scrum e métodos Ágeis



Scrum

[SAIBA MAIS...](#)

## Curso online Jira



Jira

[SAIBA MAIS...](#)

## Curso online Web Full Stack JavaScript

[SAIBA MAIS...](#)

## Curso online React Native com Firebase

[SAIBA MAIS...](#)

Conheça todos os meus cursos

# MEUS LIVROS



Programação  
Web com Node.js

[SAIBA MAIS...](#)



Programação  
Web com Node.js

[SAIBA MAIS...](#)



NODEJS E  
MICROSERVICIOS  
UM GUIA PRÁTICO



Node.js e  
Microservices

[SAIBA MAIS...](#)



MongoDB  
para  
Iniciantes

POR LUIZTOOLS

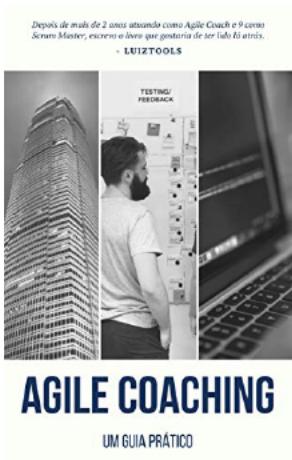
MongoDB  
para Iniciantes

[SAIBA MAIS...](#)



Scrum e  
Métodos Ágeis

[SAIBA MAIS...](#)



Agile Coaching  
UM GUIA PRÁTICO

[SAIBA MAIS...](#)



Criando apps  
para empresas  
com Android

[SAIBA MAIS...](#)



LUIZTOOLS  
KEEP CALM AND CODE JAVA  
JAVA PARA  
INICIANTES

Java para  
iniciantes

[SAIBA MAIS...](#)

## Conheça todos os meus livros

Aproveita e segue nas redes sociais:



# ANTES DE COMEÇAR

---

“

Without requirements and design, programming is  
the art of adding bugs to an empty text file.

- Louis Srygley

”

Antes de começarmos, é bom você ler esta seção para evitar surpresas e até para saber se este livro é para você.

## Para quem é este livro

Primeiramente, este livro vai lhe dar um primeiro contato com o banco MySQL e a criar uma aplicação de teste muito simples, usando JavaScript (com Node.js), mas não vai lhe ensinar lógica básica e algoritmos e nem os fundamentos de banco de dados, ele exige que você já saiba isso, ao menos em um nível básico (final do primeiro semestre da faculdade de computação, por exemplo).

Segundo, este livro exige que você já tenha conhecimento técnico prévio sobre computadores, que ao menos saiba mexer em um e que preferencialmente possua um.

Parto do pressuposto que você é ou já foi um estudante de Técnico em informática, Ciência da Computação, Sistemas de Informação, Análise e Desenvolvimento de Sistemas ou algum curso semelhante. Usarei diversos termos técnicos ao longo do livro que são comumente aprendidos nestes cursos e que não tenho o intuito de explicar aqui.

O foco deste livro é ensinar diversos aspectos do banco MySQL para quem nunca mexeu nele ou está apenas começando nessa tecnologia.

Ao término deste livro você estará apto a construir bancos de dados simples, com poucas tabelas e a criar aplicações igualmente simples usando Node.js (CRUD).

*Quer fazer um curso online de Node.js e MySQL com o autor deste livro?*

Acesse <https://www.luiztools.com.br/curso-fullstack>

---

# INTRODUÇÃO AO MYSQL

“

Good software, like wine, takes time.

- Joel Spolsky

”

Em 2007, mais ou menos, eu estava fazendo as cadeiras de Banco de Dados I e Banco de Dados II na faculdade de Ciência da Computação. Eu via como modelar um banco de dados relacional, como criar consultas e executar comandos SQL, além de álgebra relacional e um pouco de administração de banco de dados Oracle.

Isso tudo me permitiu passar a construir sistemas de verdade, com persistência de dados. A base de conhecimento aprendido em Oracle me permitiu mais tarde aprender o simplíssimo MS Access rapidamente e, mais tarde, migrar facilmente para o mais avançado, MS SQL Server. Posteriormente comecei a trabalhar com MySQL (que a Oracle acabou comprando), MS SQL Server Compact, Firebird (apenas estudos) e SQLite (para apps Android).

Mais tarde eu fui aprender ainda a utilizar bancos não-relacionais (os NoSQL) mas essa é uma história para outro momento.

Neste livro, a partir de agora, iremos utilizar o banco MySQL como tecnologia de persistência de dados, por vários motivos, que abordarei na sequência também.

## Origem

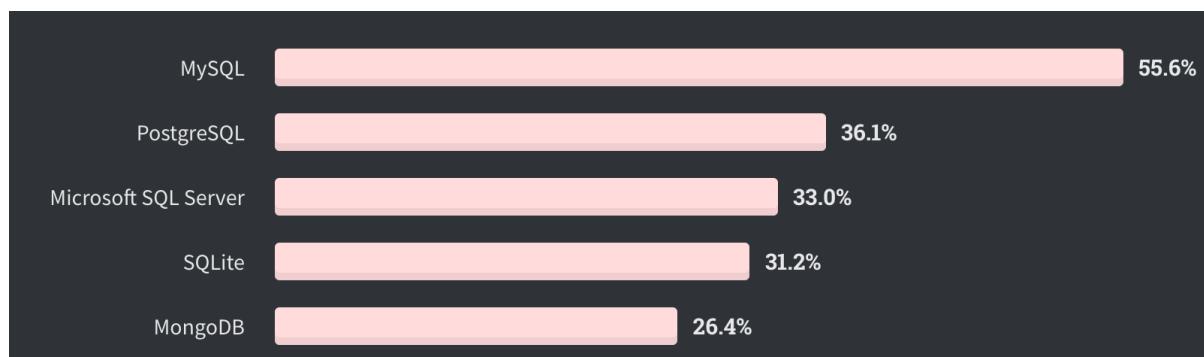
O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês Structured Query Language) como interface. É atualmente um dos sistemas de gerenciamento de bancos de dados mais populares da Oracle Corporation, com mais de 10 milhões de instalações pelo mundo.

O MySQL foi criado na Suécia por suecos e um finlandês: David Axmark, Allan Larsson e Michael “Monty” Widenius, desde a década de 80, como um projeto open-source (código aberto) e gratuito. Em 2008, a MySQL AB, desenvolvedora do MySQL foi adquirida pela Sun Microsystems, por US\$ 1 bilhão, um preço jamais visto no setor de licenças livres. E em 2009, a Oracle comprou a Sun Microsystems e todos os seus produtos, incluindo o MySQL.

Hoje seu desenvolvimento e manutenção empregam aproximadamente 400 profissionais no mundo inteiro, e mais de mil contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito dele. Como eu.

Entre os usuários do banco de dados MySQL estão: Facebook, Adobe, Alcatel, Zappos, 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.

Existem dezenas de tecnologias de bancos de dados no mundo, sendo que MySQL é, com grande margem, o mais popular dentre eles como o gráfico abaixo da pesquisa mais recente de bancos de dados utilizados pela audiência do StackOverflow em 2020 mostra. Um a cada dois desenvolvedores do mundo usam MySQL em seus projetos.



Fonte: <https://insights.stackoverflow.com/survey/2020#most-popular-technologies>

MySQL é gratuito, de código-aberto, leve e usa o modelo entidade-relacionamento com a linguagem de consulta SQL, características extremamente comuns entre os bancos de dados, facilitando muito o seu aprendizado e velocidade de desenvolvimento. O modelo relacional utilizado pelo MySQL é mundialmente conhecido como sendo de propósito geral e é usado como base para a imensa maioria dos sistemas existentes, de pequenas empresas a grandes corporações.

Basicamente, neste tipo de banco de banco de dados temos tabelas (entidades) organizadas em linhas e colunas, nas quais cada linha representa um registro da sua aplicação e cada coluna uma característica desse registro. Além disso, diferentes tabelas podem relacionar-se entre si através de “chaves”, para compor a estrutura de dados de uma empresa, por exemplo. Tudo isso serve para garantir o que chamamos de ACID: Atomicidade, Consistência, Isolamento e Durabilidade dos dados, características respeitadas pelo MySQL, a saber”

- » **Atomicidade:** um registro existe ou não existe; uma transação é feita por inteiro ou não é feita nada. O MySQL nunca vai salvar “meio-registro” em uma tabela. Se ele não conseguir salvar tudo, não vai salvar nada e vai te avisar.
- » **Consistência:** um novo registro e todas transações vão respeitar as regras de integridade definidas nas tabelas e relacionamentos. Se todo registro de cliente deve ter um CPF, então o MySQL não vai permitir que seja cadastrado um cliente sem essa informação.
- » **Isolamento:** muitos usuários podem acessar a aplicação, e consequentemente o banco de dados, ao mesmo tempo. O MySQL garante que um não vai atrapalhar o outro, isolando as suas atividades e garantindo sempre que o banco opere com os dados mais atuais.
- » **Durabilidade:** se um registro foi salvo, ele se manterá salvo, até que você o exclua. Não importa o que aconteça.

Obviamente isto tudo exige que você “modele” a sua base de dados corretamente no MySQL, para que ele consiga garantir o funcionamento esperado.

## Ambiente e Configuração

MySQL é um banco relacional ou SQL, ao usar o modelo de tabelas com linhas e colunas que se relacionam entre si. Em MySQL trabalhamos com linhas ou tuplas de registros cujas características são organizadas em colunas, o jeito mais tradicional, popular e utilizado no mercado de trabalho.

Falaremos melhor de MySQL quando chegar a hora, pois ele será o banco de dados que utilizaremos em nossas lições que exijam persistência de informações. Por ora, apenas baixe o servidor do MySQL para o seu sistema operacional. Você encontra a distribuição gratuita de MySQL para o seu sistema operacional no site oficial:

<https://dev.mysql.com/downloads/mysql/>

Entraremos nos detalhes da instalação quando chegar a hora certa.

## MySQL Workbench

Para gerenciar o MySQL existem uma série de ferramentas de linha de comando disponíveis junto próprio servidor, mas se quiser ser mais produtivo geralmente se utilizam ferramentas visuais, recomendo a MySQL Workbench. É gratuito e você pode usar tanto para estudar quanto para trabalhar.

<https://www.mysql.com/products/workbench/>

Por que eu recomendo esta ferramenta ao invés de diversas outras? Usei ela em diversos projetos e além disso ela é mantida pelo mesmo time do MySQL e atualizando sua ferramenta constantemente.

A instalação dela é essencial para o avanço nas lições deste livro, pois a modelagem dos bancos de dados que usaremos será feita unicamente aqui.

Mais tarde, você pode ser solicitado a baixar e instalar outros softwares, mas por enquanto estes são o suficiente.

*Quer fazer um curso online de Node.js e MySQL com o autor deste livro?  
Acesse <https://www.luiztools.com.br/curso-fullstack>*

---

# MYSQL NA PRÁTICA

2

“

*Truth can only be found in one place: the code.*

*- Robert C. Martin*

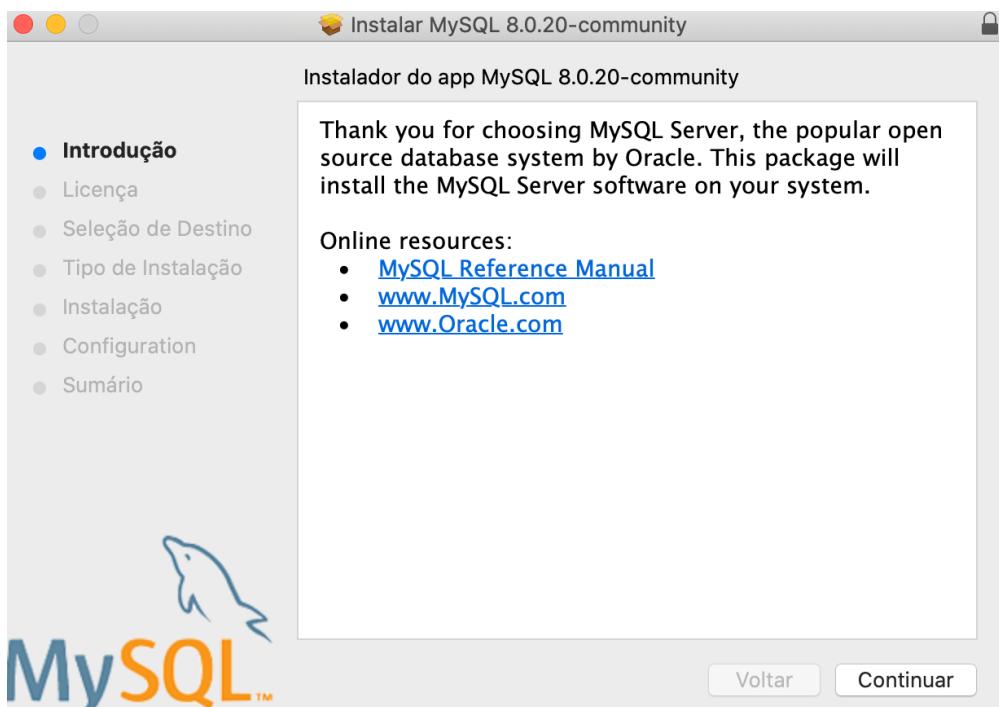
”

Diversos players de cloud computing fornecem versões de MySQL hospedadas e prontas para uso como **Umbler** e Amazon Web Services (AWS), no entanto é muito importante um conhecimento básico de administração local de MySQL para entender melhor como tudo funciona. Não focaremos aqui em nenhum aspecto de segurança, de alta disponibilidade, de escala ou sequer de administração avançada de MySQL. Deixo todas estas questões para você ver junto à documentação oficial no site do MySQL, onde inclusive você pode estudar e tirar as certificações.

Caso ainda não tenha feito isso, acesse o site oficial do MySQL e baixe gratuitamente a versão mais recente para o seu sistema operacional.

<https://dev.mysql.com/downloads/mysql/>

Baixe o arquivo compactado (ou não) e rode o executável que instalará os arquivos na sua pasta de Arquivos de Programas ou outra qualquer, dependendo do seu sistema operacional. A instalação é bem direta, basta ir avançando, concordar com os termos de uso e terá um momento onde o instalador vai dizer onde o MySQL será instalado, tome nota ou altere essa localização.



Em outro momento ela vai lhe pedir a senha do usuário root. Atenção aqui.

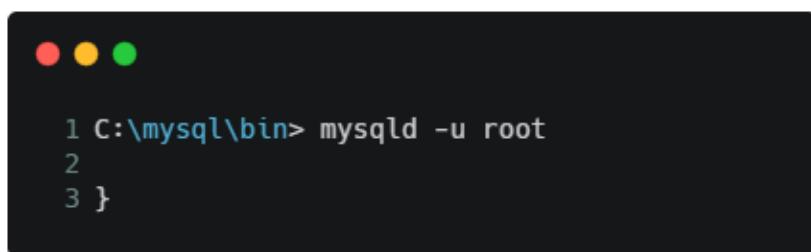
Esse é o administrador do seu servidor de banco de dados, então coloque uma senha forte mas que você consiga se lembrar. Eu coloquei apenas luiztools, pois em produção, minhas aplicações usam senhas muito mais fortes que isso, mas minha máquina local é apenas para estudos.

Se você ir até onde o MySQL foi instalado na sua máquina, vai encontrar uma estrutura de pastas onde a que mais nos interessa é a pasta bin. Nessa pasta estão uma coleção de utilitários de linha de comando que são o coração do MySQL (no caso do Windows, todos terminam com .exe):

- » **mysqld**: inicializa o servidor do MySQL;
- » **mysql**: inicializa o cliente de consulta a banco de dados;
- » **mysqladmin**: inicializa o cliente de administração do servidor MySQL;
- » **mysqldump**: realiza dump de um banco de dados (backup binário);
- » **mysqlpump**: realiza o backup lógico de um banco de dados (instruções);
- » **mysqlimport**: importa registros pro seu banco;
- » entre outros.

Para subir um servidor de MySQL na sua máquina é muito fácil: execute o utilitário mysqld via linha de comando como abaixo, sendo que você deve ter privilégios de administrador no prompt de comandos (sudo em sistemas Unix ou “Executar como Administrador” no Windows).

Código 2.1: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



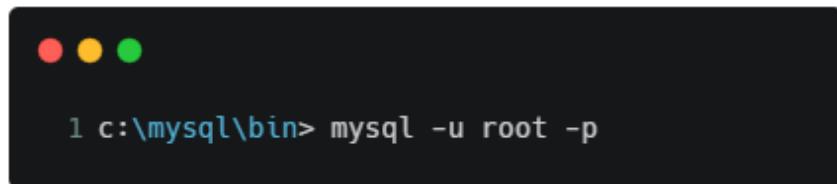
```
1 C:\mysql\bin> mysqld -u root
2
3 }
```

**Nota:** em sistemas Unix, os comandos devem conter ./ antes de cada utilitário, como ./mysqld, por exemplo. Em Windows, uma vez que você esteja dentro da pasta correta, apenas digite o nome do executável, sem a extensão ".exe".

Isso irá iniciar o servidor do MySQL como administrador. Uma vez que apareça no prompt “ready for connections. Version: x.x.x’ socket: ‘/tmp/mysql.sock’ port: 3306”, está pronto, o servidor está executando corretamente e você já pode utilizá-lo na porta padrão 3306.

Agora abra outro prompt de comando (o primeiro ficará executando o servidor) e novamente dentro da pasta bin do MySQL, digite:

Código 6.2: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 c:\mysql\bin> mysql -u root -p
```

O prompt irá te pedir a senha do root e após a conexão funcionar, verá que uma conexão foi estabelecida e um sinal de “>” indicará que você já pode digitar os seus comandos e queries para enviar à essa conexão.

Este utilitário ‘mysql’ pode ser útil para obter algumas informações ou fazer algumas consultas rapidamente, mas não o usaremos aqui no livro. Minha intenção em fazer uma conexão com ele foi mais para ver se o servidor estava rodando corretamente antes de avançarmos.

## Criando nossa base de dados

Já tem muitas décadas que os utilitários de linha de comando para banco de dados perderam terreno para os clientes gráficos, principalmente no terreno dos bancos relacionais, uma vez que uma parte essencial da administração de um SGBD relacional é a modelagem do esquema/schema do banco de dados. E você não vai querer modelar um banco de dados via linha de comando, não é mesmo?

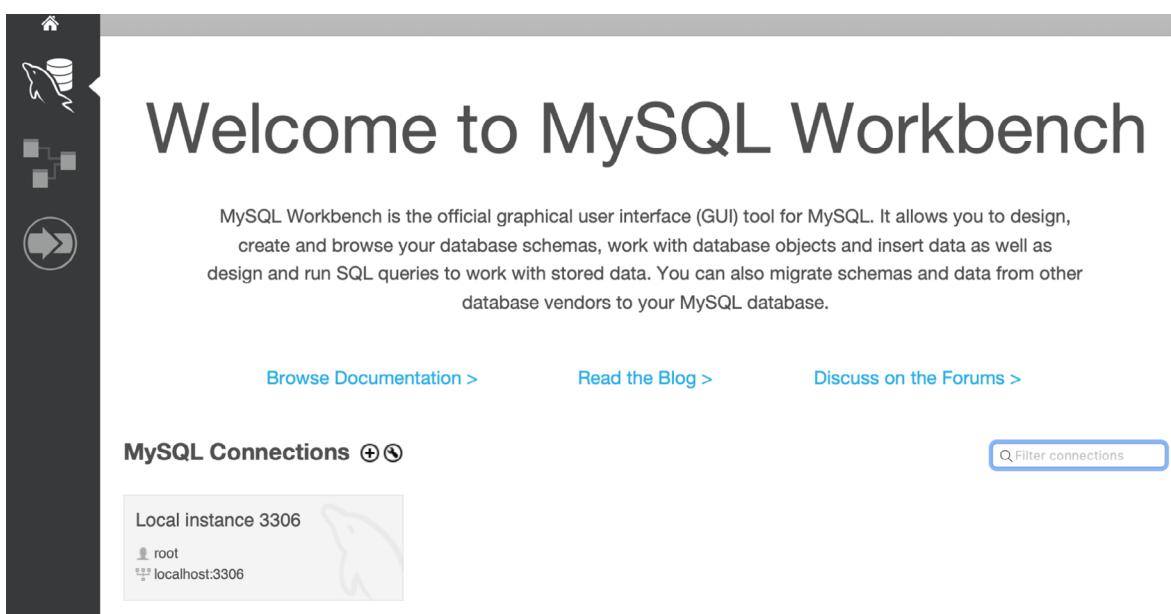
Sendo assim, se ainda não o fez, baixe do site oficial o cliente que eu

recomendo: MySQL Workbench!

<https://dev.mysql.com/downloads/workbench/>

O MySQL Workbench é um cliente gráfico para MySQL mantido pela mesma empresa do banco de dados e, embora não seja o primeiro ou o único existente, isso dá a ele o status de cliente gráfico oficial, motivo a minha recomendação. Ah, e ele é gratuito, é claro.

A instalação é muito direta, ainda mais fácil que a do servidor de MySQL. Após terminar todo o next-next-finish, execute o workbench e você verá a tela abaixo.

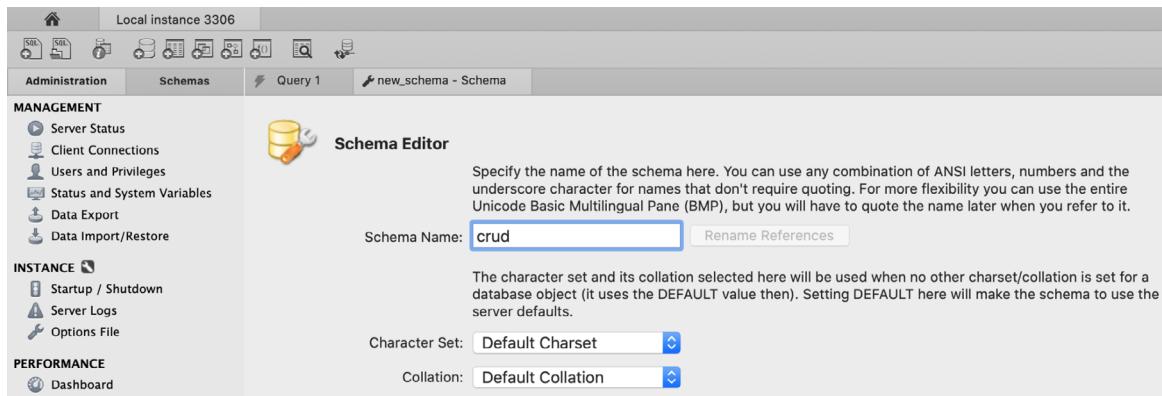


Note na parte mais inferior que ele já identificou que tem um servidor MySQL rodando na minha máquina (localhost), na porta 3306 e que clicando ali já podemos nos conectar no mesmo, apenas informando a senha de root do servidor.

Quando a conexão for efetuada, ele já deve abrir em um arquivo em branco para que você possa escrever consultas, do mesmo jeito que faria no client de linha de comando, mas com mais recursos como syntax-highlighting, autocomplete, etc. Mas não vamos escrever consultas, ao menos não por enquanto.

Nós vamos aqui criar um novo schema de banco de dados no servidor atual, clicando no ícone de banco de dados da toolbar superior (o quarto ícone da esquerda para direita). Como nome do schema, vou chamar

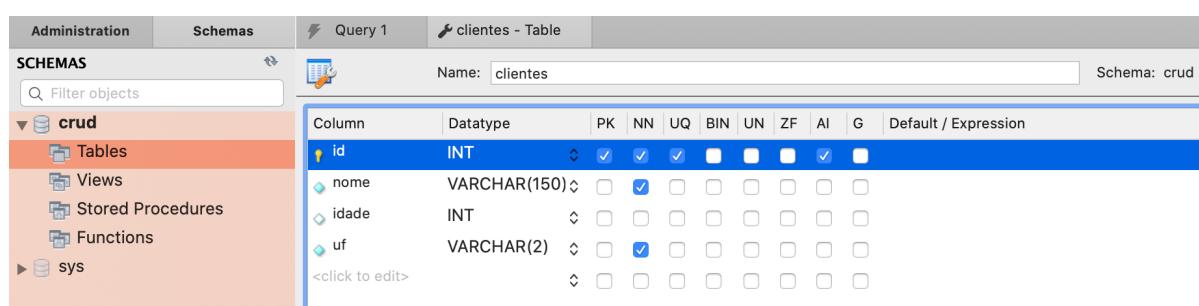
de crud (é o nome da nossa aplicação Node também, lembra?). Clique em Apply no canto inferior direito e depois o Workbench vai te mostrar o script SQL que será executado, pedindo a sua confirmação (Apply de novo).



**Atenção:** todos os objetos de bancos de dados (schemas, tabelas, colunas, índices, etc) devem ser escritos sem acentos, traços ou espaços, para evitar problemas. Tirando essa dica universal, cada empresa tem os seus próprios padrões de nomes de objetos do banco de dados, como underline ao invés de espaços e traços, tudo em caixa alta, prefixo 'TB\_' na frente de tabelas e por aí vai. Esses padrões só dizem respeito às empresas, não são padrões universais, mas a sugestão geral é que sejam consistentes.

Depois de termos criado o esquema, ele deverá aparecer na aba schema, da área esquerda do workbench, como mostra a imagem abaixo. Expandido o schema ‘crud’, você verá uma lista de opções, sendo que a que nos interessa aqui é Tables (tabelas).

Clique com o botão direito em Tables e escolha a opção ‘create table’, o que vai abrir o editor gráfico de tabelas como abaixo.



Vamos criar aqui uma tabela chamada clientes (name) para ser usada pela aplicação Node.js que começamos no capítulo anterior. Conforme o formulário que criamos anteriormente, ela deve ter as seguintes colunas (as mesmas que aparecem na imagem acima):

**id:**

- » Datatype INT (número inteiro);
- » Chave Primária (marque o campo PK - Primary Key);
- » Obrigatório (marque o campo NN - Not Null);
- » Auto-incremental (AI);

Meu intuito com este livro não é substituir as disciplinas de banco de dados dos cursos de computação, mas apenas frisando alguns pontos importantes, toda tabela precisa ter uma chave primária, que é o identificador único de cada registro daquela tabela. É prática comum que essa chave-primária seja numérica e inteira, por questões de armazenamento e performance, mas novamente, cada empresa tem os seus padrões.

**nome:**

- » Datatype VARCHAR com até 150 caracteres (string);
- » Obrigatório (NN)

O tipo de dado VARCHAR dos bancos SQL funciona com limitação de tamanho, então você deve pensar em um tamanho adequado ao seu cenário e ajustar a sua aplicação para que não envie textos maiores do que o permitido, ou dará erro ao salvar registros.

**idade:**

- » Datatype INT

Essa coluna da tabela clientes eu resolvi deixar bem simples, nem mesmo é obrigatória. Assim, caso o usuário da aplicação não informe a idade do cliente, não teremos qualquer erro na aplicação.

**uf:**

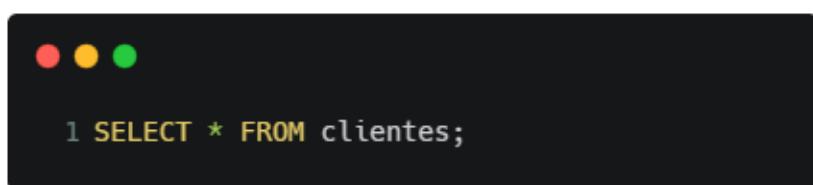
- » Datatype VARCHAR(2)
- » Obrigatório (NN)

E por fim, nosso UF (sigla da unidade federativa/estado brasileiro) terá apenas 2 caracteres e será obrigatório.

Clique em Apply no canto inferior direito do editor visual de tabela, o workbench vai te mostrar o comando SQL que será executado para confirmação.

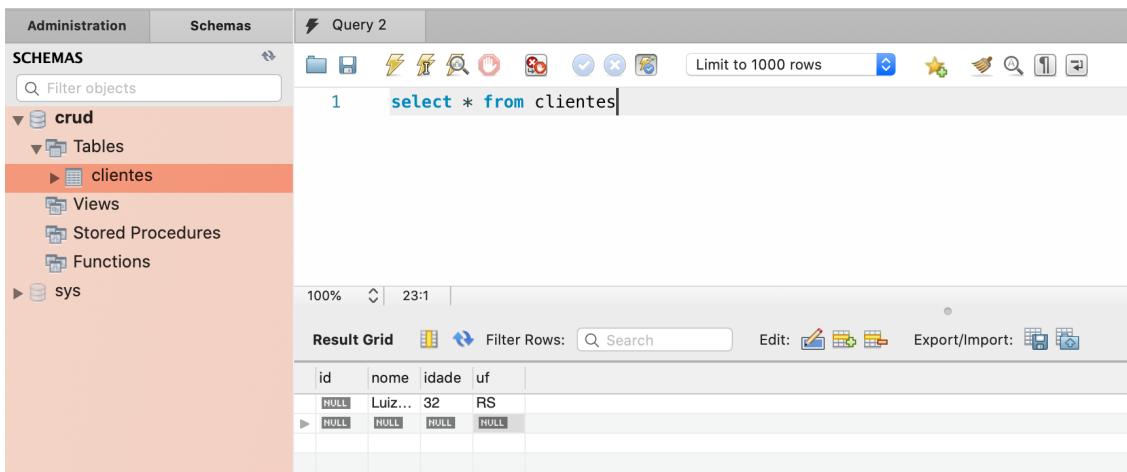
Agora, você pode escrever uma consulta SQL bem simples, apenas para ver o que tem dentro desta tabela que acabamos de criar. O botão para executar a consulta é o “Trovão”, na toolbar do editor de consultas (terceiro ícone da esquerda pra direita).

Código 2.3: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 SELECT * FROM clientes;
```

Isso deve listar uma tabela vazia no canto inferior do editor de consulta. Nesta tabela que se abrirá vazia, você pode inserir registros de maneira visual, como mostro na imagem abaixo (não esqueça de confirmar a inserção clicando em Apply, no canto inferior direito).



id	nome	idade	uf
NULL	Luiz...	32	RS
NULL	NULL	NULL	NULL

Obviamente você pode rodar comandos INSERT também, como no exemplo abaixo, chegando ao mesmo resultado. O uso mais visual ou mais textual da ferramenta vai depender mais dos seus conhecimentos e interesse em se aprofundar mais em uma ou outra abordagem.

Código 2.4: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 INSERT INTO clientes (nome, idade, uf) VALUES ('Luiz', 32,  
'RS');
```

A cláusula `INSERT` é bem simples: primeiro você lista entre parênteses logo após o nome da tabela os campos que serão inseridos. Depois, você lista entre parênteses logo após a cláusula `VALUES` os valores que serão inseridos nos campos anteriormente citados, na mesma ordem que referenciou eles. Vale ressaltar que valores textuais devem estar entre aspas simples (como no caso do nome e uf).

## O básico de SQL

Este não é um livro focado em MySQL, mas alguns comandos elementares são importantes que você conheça antes de voltarmos a codificar aplicações em Node.js, que a partir de agora terão persistência nesta fantástica tecnologia.

### **SELECT**

Fizemos uma consulta anterior bem simples: selecione todos os clientes do banco de dados. Mas e se não quisermos trazer todos, mas apenas aqueles que atendam a algumas características?

Por exemplo, e se quisermos filtrar pelo estado que o cliente mora? Podemos fazer esse e várias outras combinações de filtros usando a cláusula `WHERE` logo após o nome da tabela, como abaixo:

Código 2.5: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

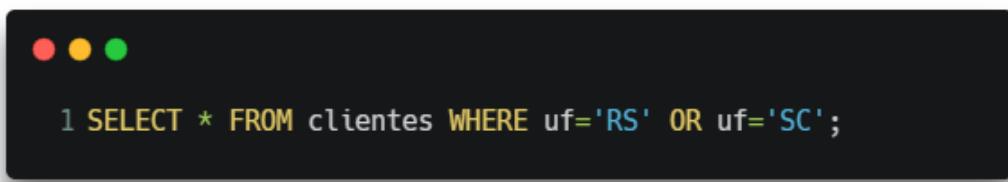
```
1 SELECT * FROM clientes WHERE uf='RS';
```

Na consulta acima, vamos trazer todas as colunas (\*) dos registros da tabela clientes que tenham a uf cadastrada como sendo ‘RS’. Sempre que usarmos ou passarmos valores textuais para o banco de dados, precisamos fazê-lo entre aspas (repare que no código do INSERT eu também usei aspas).

Note também que eu costumo escrever as cláusulas SQL em maiúsculo, enquanto que os objetos do meu banco de dados eu escrevo sempre em minúsculo. Esse é um padrão próprio, adote-o se fizer sentido para você.

Mas e se for todos que sejam de dois estados específicos?

Código 2.6: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 SELECT * FROM clientes WHERE uf='RS' OR uf='SC';
```

Aqui entra a lógica booleana/computacional que você já deve ter aprendido em outro momento da sua vida acadêmica. A linguagem SQL aceita alguns operadores lógicos como:

OR: os registros serão retornados se atenderem à expressão da esquerda ou da direita do OR. No caso acima, RS ou SC.

AND: os registros serão retornados se atenderem à ambas expressões, na esquerda e direita do AND. Se usássemos AND ao invés de OR no exemplo anterior, não seria retornado nenhum registro, pois não tem como um cliente morar em dois estados.

No caso de precisarmos encadear múltiplas condições OR (imagine que queremos clientes de 3 possíveis estados), podemos usar de outras cláusulas condicionais, como:

- » **IN:** os registros serão retornados se a coluna filtrada pertencer ao grupo citado entre parênteses;
- » **NOT IN:** os registros serão retornados se a coluna filtrada não pertencer ao grupo citado entre parênteses.

Assim, o mesmo filtro por dois estados, que fizemos anteriormente usando OR, também poderia ser feito usando IN, da seguinte forma:

Código 2.7: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 SELECT * FROM clientes WHERE uf IN ('RS', 'SC');
```

Quando o assunto são colunas numéricas, como a idade ou o id do cliente, podemos usar de operadores lógicos comuns na matemática como >, >=, < e <=. Assim, se quisermos retornar todos os clientes que são maior de idade, podemos fazer da seguinte forma:

Código 2.8: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 SELECT * FROM clientes WHERE idade >= 18;
```

E quando queremos pesquisar por parte de uma coluna textual (VARCHAR), podemos usar o operador LIKE, com o coringa %, como abaixo:

Código 2.9: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 SELECT * FROM clientes WHERE nome LIKE 'L%';
```

O % é um coringa e representa “quaisquer outros caracteres”. No exemplo acima, retornaremos todos os clientes cuja coluna nome comece com L e depois tenha quaisquer outros caracteres. Se quiséssemos nomes finalizados em L, bastaria iniciar a expressão entre aspas com o %. E por fim, se quiséssemos nomes que contenham L (em qualquer posição), poderíamos colocar % no início e no fim da expressão.

Mas e a consulta abaixo?

Código 2.10: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
 1 SELECT * FROM clientes WHERE nome LIKE 'L%' AND idade > 18;
```

Sim, todos os clientes que comecem com a letra L e que sejam maiores de idade. Ou seja, os operadores lógicos permitem que a gente combine tantos filtros quanto a gente quiser.

Outras cláusulas que você pode usar como filtro incluem:

- » **IS NULL**: para verificar se uma coluna não foi preenchida;
- » **IS NOT NULL**: para verificar se uma coluna foi preenchida;
- » entre outros!

Você também pode limitar o número de resultados que sua consulta vai trazer, se a base for muito grande, usando a cláusula LIMIT, da seguinte maneira:

Código 2.11: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
 1 SELECT * FROM clientes WHERE idade > 18 LIMIT 10;
```

No exemplo acima retornaremos os 10 primeiros clientes que sejam maiores de idade.

Mas como o MySQL sabe quais são os primeiros clientes? Pela chave primária!

Por padrão, toda consulta toma como base a ordenação da chave primária da tabela. Se essa ordenação não lhe atender, você pode

explicitamente declarar como deseja que sejam retornadas as informações usando a cláusula ORDER BY:

Código 2.12: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 SELECT * FROM clientes WHERE idade > 18 ORDER BY nome LIMIT 10;
```

O ORDER BY vem logo após o seu último filtro, mas antes do LIMIT. Assim, pegaremos os 10 primeiros clientes, ordenados por nome, que sejam maiores de idade.

Ok, vimos como usar o SELECT de maneiras bem interessantes e úteis, mas e os demais comandos de manipulação do banco?

UPDATE

Além do INSERT que vimos antes, também podemos atualizar registros já existentes, por exemplo usando o comando UPDATE, como abaixo;

Código 2.13: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 UPDATE clientes SET nome='Luiz Fernando', uf = 'RS' WHERE id=1;
```

O UPDATE espera sempre o nome da tabela, a cláusula SET com os campos que serão atualizados, separados por vírgula (campos não listados permanecerão com o valor atual) e o filtro WHERE para que não sejam alterados TODOS os registros da tabela.

**Atenção:** Sim, se não informar o WHERE ou informar incorretamente, você pode acabar alterando TODOS os registros da tabela. Então cuidado!

Como no exemplo acima usei a chave primária como filtro (id) e ela é única, esse UPDATE vai atualizar somente um registro, desde que exista um cliente com id =1 na tabela clientes.

A cláusula WHERE do UPDATE funciona de maneira exatamente igual quando no SELECT. No entanto, resista à tentação de colocar filtros abertos aqui, seja o mais específico possível.

## DELETE

Pra encerrar o nosso conjunto de comandos mais elementares do SQL falta o DELETE. E esse é um comando ainda mais perigoso que o UPDATE, pois se você não o usá-lo adequadamente, pode apagar todos os registros da sua tabela.

Assim, sempre use cláusula WHERE em seus Deletes e de preferência filtrando pela chave primária, que é única e garante a exclusão de apenas um registro da tabela, como abaixo:

Código 2.14: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 DELETE FROM clientes WHERE id=2;
```

Vai excluir todos os clientes cujo id seja igual a 2

Simples, não?!

Obviamente existem coisas muito mais avançadas do que esse rápido tópico de SQL. Lhe encorajo a dar uma olhada no site oficial do banco de dados onde há a seção de documentação, vários tutoriais e até mesmo a possibilidade de tirar certificações online para garantir que você realmente entendeu a tecnologia.

---

# NODE.JS & MYSQL

3

“

Some of the best programming is done on paper, really.  
Putting it into the computer is just a minor detail

- Max Kanat-Alexander

”

Você vai precisar ter o Node.js instalado se quiser realizar o exemplo prático de criação de uma aplicação simples com MySQL deste capítulo.

A plataforma Node.js é distribuída gratuitamente pelo seu mantenedor, Node.js Foundation, para diversos sistemas operacionais em seu website oficial Nodejs.org:

<https://nodejs.org>

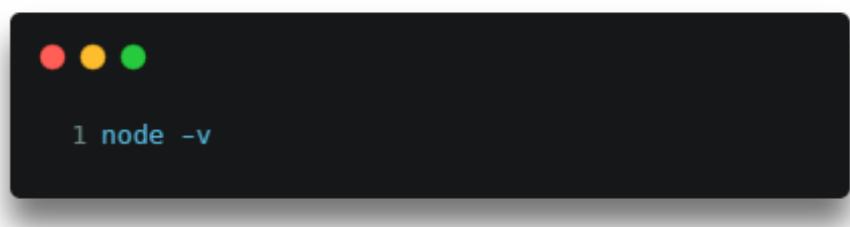
Na tela inicial você deve encontrar dois botões grandes e verdes para fazer download e a versão recomendada é a estável (LTS).

A instalação não requer nenhuma instrução especial, apenas avance cada uma das etapas e aceite o contrato de uso da plataforma.

Deve ser observado que o Node.js não é um ambiente visual ou uma ferramenta integrada de desenvolvimento, embora mesmo assim seja possível o desenvolvimento de aplicações complexas apenas com o uso do mesmo, sem nenhuma ferramenta externa.

Após a instalação do Node, para verificar se ele está funcionando, abra seu terminal de linha de comando (DOS, Terminal, Shell, bash, etc) e digite o comando abaixo:

Código 3.1: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



O resultado deve ser a versão do Node.js atualmente instalada na sua máquina. Isso mostra que o Node está instalado na sua máquina e funcionando corretamente.

Avançando nossos testes iniciais (apenas para nos certificarmos de que tudo está funcionando como deveria), vamos criar nosso primeiro programa JavaScript para rodar no Node.js com apenas um arquivo.

Abra o editor de texto mais básico que você tiver no seu computador (Bloco de Notas, vim, nano, etc) e escreva dentro dele o seguinte trecho de código:

Código 3.2: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 console.log('Olá mundo!');
```

Agora salve este arquivo com o nome de index.js (certifique-se que a extensão do arquivo seja “.js”, não deixe que seu editor coloque “.txt” por padrão) em qualquer lugar do seu computador, mas apenas memorize esse lugar, por favor. :)

Para rodar esse programa JavaScript, abra novamente o terminal de linha de comando e execute o comando abaixo:

Código 3.3: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 node /documents/index.js
```

Isto irá executar o programa contigo no arquivo /documents/index.js usando o runtime do Node. Note que aqui eu salvei meu arquivo .js na pasta documents, logo no seu caso, esse comando pode variar (no Windows inclusive usa-se \ ao invés de /, por exemplo). Uma dica é quando abrir o terminal, usar o comando ‘cd’ para navegar até a pasta onde seus arquivos JavaScript são salvos. Eu inclusive recomendo que você crie uma pasta NodeProjects ou simplesmente Projects na sua pasta de usuário para guardar todos os exemplos desse livro de maneira organizada. Assim, sempre que abrir um terminal, use o comando cd para ir até a pasta apropriada.

Se o seu terminal já estiver apontando para a pasta onde salvou o seu arquivo .js, basta chamar o comando ‘node’ seguido do respectivo nome do arquivo (sem pasta) que vai funcionar também.

Ah, o resultado da execução anterior? Apenas um ‘Olá mundo!’ (sem aspas) escrito no seu console, certo?!

Note que tudo que você precisa de ferramentas para começar a programar Node é exatamente isso: o runtime instalado e funcionando, um terminal de linha de comando e um editor de texto simples. Obviamente podemos adicionar mais ferramentas ao nosso arsenal, e é disso que trata a próxima sessão.

## Visual Studio Code

Ao longo deste capítulo vamos escrever uma série de códigos em JavaScript e recomendo que você use o editor de código Visual Studio Code, da Microsoft, para esta finalidade.

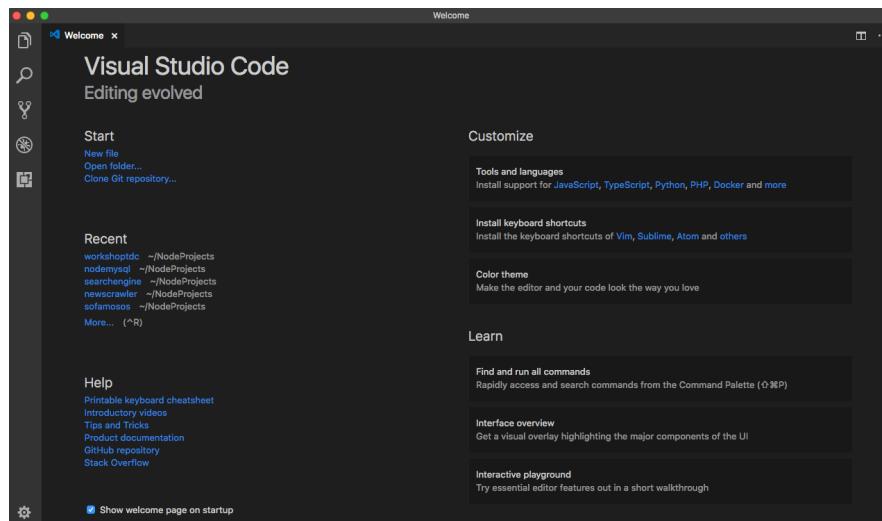
Esta não é a única opção disponível, mas é uma opção bem interessante e é a que uso, uma vez que reduz consideravelmente a curva de aprendizado, os erros cometidos durante o aprendizado e possui ferramentas de depuração muito boas, além de suporte a Git e linha de comando integrada. Apesar de ser desenvolvido pela Microsoft, é um projeto gratuito, de código-aberto, multi-plataforma e com extensões para diversas linguagens e plataformas, como Node.js. E diferente da sua contraparte mais “parruda”, o Visual Studio original, ele é bem leve e pequeno.

Para baixar e instalar o Visual Studio Code, acesse o seguinte link, no site oficial da ferramenta:

<https://code.visualstudio.com/>

Você notará um botão grande e verde para baixar a ferramenta para o seu sistema operacional. Apenas baixe e instale, não há qualquer preocupação adicional.

Após a instalação, mande executar a ferramenta Visual Studio Code e você verá a tela de boas vindas, que deve se parecer com essa abaixo, dependendo da versão mais atual da ferramenta. Chamamos esta tela de Boas Vindas (Welcome Screen).



No menu do topo você deve encontrar a opção File > New File, que abre um arquivo em branco para edição. Apenas adicione o seguinte código nele:

Código 3.4: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

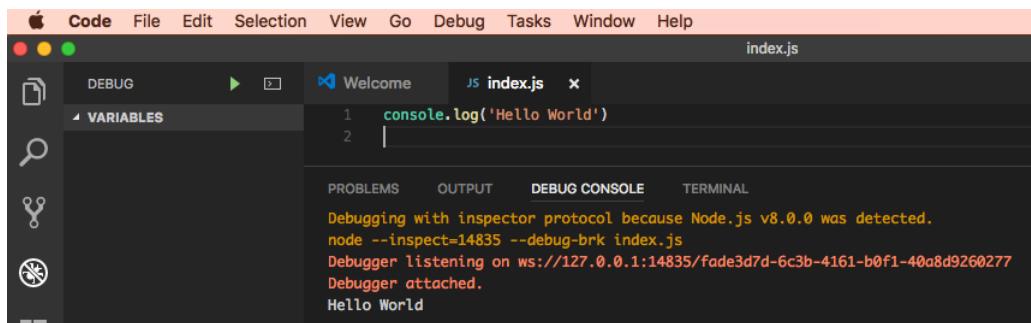
A screenshot of a terminal window. At the top, there are three colored window control buttons (red, yellow, green). Below them is a black terminal window containing the text: 1 console.log('Hello World'); followed by a new line.

JavaScript é uma linguagem bem direta e sem muitos rodeios, permitindo que com poucas linhas de código façamos coisas incríveis. Ok, um olá mundo não é algo incrível, mas este mesmo exemplo em linguagens de programação como C e Java ocuparia muito mais linhas.

Basicamente o que temos aqui é o uso do objeto 'console', que nos permite ter acesso ao terminal onde estamos executando o Node, e dentro dele estamos invocando a função 'log' que permite escrever no console passando um texto entre aspas (simples ou duplas, tanto faz, mas recomendo simples).

Salve o arquivo escolhendo File > Save ou usando o atalho Ctrl + S. Minha sugestão é que salve dentro de uma pasta NodeProjects/ HelloWorld para manter tudo organizado e com o nome de index.js. Geralmente o arquivo inicial de um programa Node.js se chama index, enquanto que a extensão ‘.js’ é obrigatória para arquivos JavaScript.

Para executar o programa escolha Debug > Start Debugging (F5). O Visual Studio Code vai lhe perguntar em qual ambiente deve executar esta aplicação (Node.js neste caso) e após a seleção irá executar seu programa com o resultado abaixo como esperado.



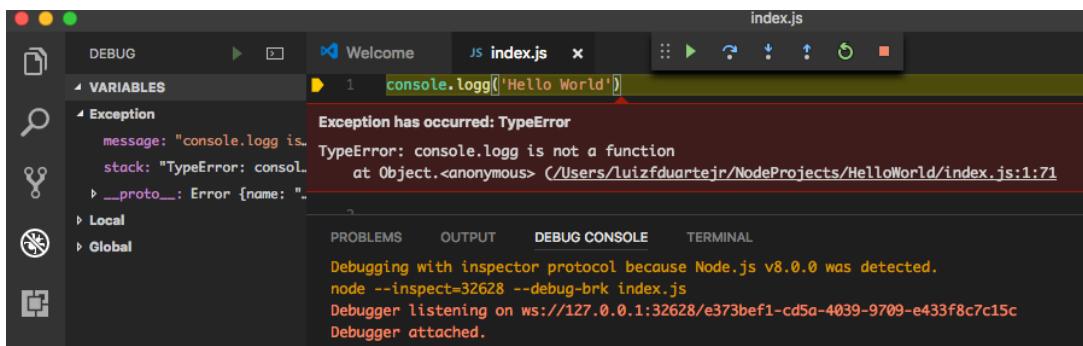
```
index.js
1 console.log('Hello World')
2 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Debugging with inspector protocol because Node.js v8.0.0 was detected.
node --inspect=14835 --debug-brk index.js
Debugger listening on ws://127.0.0.1:14835/fade3d7d-6c3b-4161-b0f1-40a8d9260277
Debugger attached.
Hello World
```

Parabéns! Seu programa funciona!

Se houver erros de execução, estes são avisados com uma mensagem vermelha indicando qual erro, em qual arquivo e em qual linha. Se mais de um arquivo for listado, procure o que estiver mais ao topo e que tenha sido programado por você. Os erros estarão em Inglês, idioma obrigatório para programadores, e geralmente possuem solução se você souber procurar no Google em sites como StackOverflow entre outros. No exemplo acima eu digitei erroneamente a função ‘log’ com dois ‘g’s



```
index.js
1 console.log('Hello World')
```

Exception has occurred: TypeError

```
TypeError: console.log is not a function
at Object.<anonymous> (/Users/luizfdurantejr/NodeProjects>HelloWorld/index.js:1:71)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Debugging with inspector protocol because Node.js v8.0.0 was detected.
node --inspect=32628 --debug-brk index.js
Debugger listening on ws://127.0.0.1:32628/e373bef1-cd5a-4039-9709-e433f8c7c15c
Debugger attached.
```

(TypeError = erro de digitação). Conceitos mais aprofundados sobre JavaScript serão vistos posteriormente.

Caso note que sempre que manda executar o projeto (F5) ele pergunta

qual é o ambiente, é porque você ainda não configurou um projeto corretamente no Visual Studio Code. Para fazer isso é bem simples, vá no menu File > Open e selecione a pasta do seu projeto, HelloWorld neste caso. O VS Code vai entender que esta pasta é o seu projeto completo.

Agora, para criarmos um arquivo de configuração do VS Code para este projeto, basta ir no menu Debug > Add Configuration, selecionar a opção Node.js e salvar o arquivo, sem necessidade de configurações adicionais. Isso irá criar um arquivo launch.json, de uso exclusivo do VS Code, evitando que ele sempre pergunte qual o environment que você quer usar.

E com isso finalizamos a construção do nosso Olá Mundo em Node.js usando Visual Studio Code, o primeiro programa que qualquer programador deve criar quando está aprendendo uma nova linguagem de programação!

## Node.js + MySQL

Agora que entendemos, em seções anteriores, como criar um programa Node.js usando JavaScript e em capítulo anterior como usar MySQL, é hora de juntar as duas pontas!

O que vamos fazer agora é uma aplicação de linha de comando bem simples que vai conectar no banco de dados que criamos no capítulo anterior e realizar uma série de operações na mesma tabela de clientes. Você verá que com um pouco de criatividade isso será o bastante para que, mais tarde você consiga fazer sistemas completos com listagem, cadastro, edição e exclusão de clientes a partir do nosso banco MySQL.

Certifique-se de que o seu servidor MySQL está rodando e que ele contém alguns clientes de exemplo, conforme os exercícios que fizemos anteriormente. Abra também outro terminal e navegue até a pasta do projeto, começaremos este tutorial a partir daí.

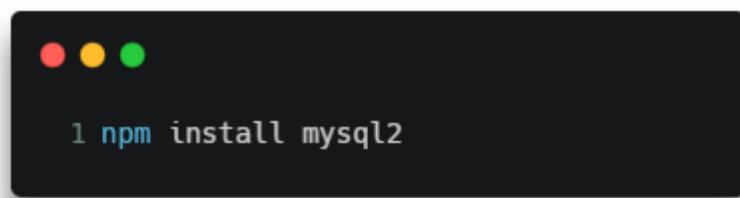
### MySQL Driver

Existem diferentes formas de se conectar a bancos de dados usando Node.js, usarei aqui um dos vários drivers existentes para MySQL que

não é o oficial, mas é o mais moderno e com a melhor performance, chamado mysql2.

Para instalar essa dependência no seu programa Node.js, que na verdade é apenas um arquivo index.js por enquanto, rode o seguinte comando na mesma pasta onde está o index.js. Além de baixar a dependência nesta pasta, o comando também salva a mesma em um arquivo local de configuração chamado packages.json:

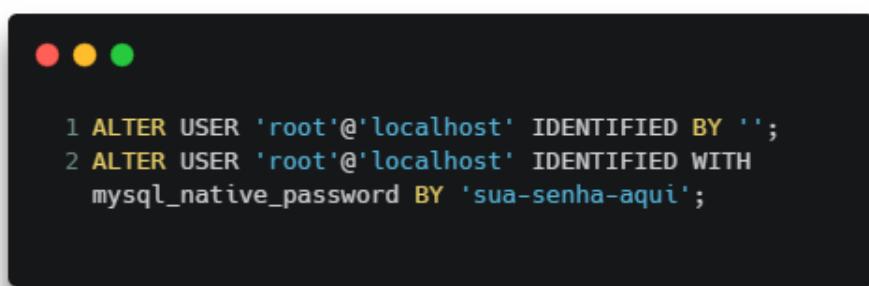
Código 3.5: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 npm install mysql2
```

**Atenção:** na data que escrevo este livro o MySQL mudou algumas questões de segurança e os pacotes do mysql e mysql2 para Node.js ainda não se adaptaram. Sendo assim, caso você tenha um erro ER\_NOT\_SUPPORTED\_AUTH\_MODE ao conectar-se no banco de dados mais à frente, execute os seguintes comandos no Workbench:

Código 3.6: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

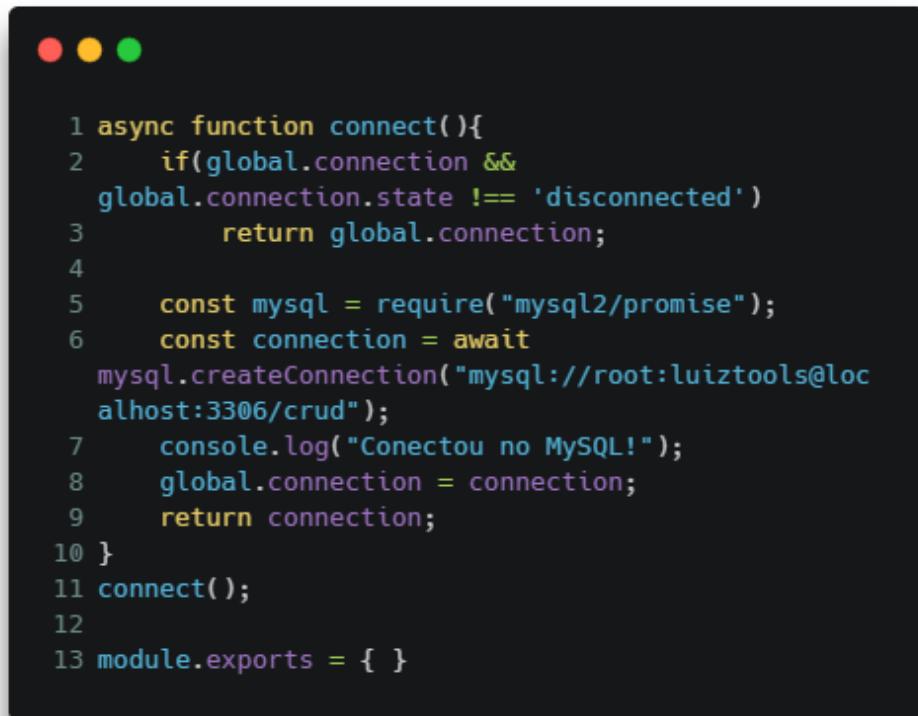


```
1 ALTER USER 'root'@'localhost' IDENTIFIED BY '';
2 ALTER USER 'root'@'localhost' IDENTIFIED WITH
  mysql_native_password BY 'sua-senha-aqui';
```

Para deixar nossa aplicação minimamente organizada não vamos sair por aí escrevendo lógica de acesso à dados. Vamos centralizar tudo o que for responsabilidade do MySQL dentro de um novo módulo chamado db.js, que nada mais é do que um arquivo db.js na raiz do nosso projeto.

Esse arquivo será o responsável pela conexão e manipulação do nosso banco de dados, usando o driver do MySQL. Adicione estas linhas nele:

Código 3.7: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 async function connect(){
2     if(global.connection &&
3         global.connection.state !== 'disconnected')
4         return global.connection;
5
6     const mysql = require("mysql2/promise");
7     const connection = await
        mysql.createConnection("mysql://root:luiztools@loc
        alhost:3306/crud");
8     console.log("Conectou no MySQL!");
9     global.connection = connection;
10    return connection;
11 }
12
13 module.exports = { }
```

Ignore o primeiro if, vamos para o ‘const mysql’. Nós começamos a conexão carregando o nosso pacote, mais especificamente o wrapper mysql2/promise que é justamente o que nos dá o suporte a Promises que comentei antes.

Com esta constante mysql, consigo chamar a function createConnection que espera a nossa connection string com o banco. Aqui você deve substituir pelos dados da sua instalação de MySQL, no formato mysql://usuario:senha@servidor:porta/banco

Note que usei a palavra reservada await antes do createConnection. Isso porque o createConnection é assíncrono, o que me obriga a usar callback, promise ou async/await para garantir que a instrução a seguir só aconteça depois da conexão já estabelecida.

Optei pelo await por ser a forma mais moderna (ES6), mas ele me obriga a usar a keyword async antes da declaração da minha function, como deve ter reparado também.

No fim da function de conexão, eu armazeno ela em uma variável global e agora sim faz sentido eu lhe explicar o primeiro if do código.

Como eu não quero ficar criando um monte de conexões com o MySQL (isso é lento e consome muitos recursos do servidor), eu vou criar apenas uma, compartilhar em uma variável global e, quando a conexão for chamada pela segunda vez, ela vai cair naquele if inicial que verifica se já não existe uma conexão global conectada. Se existir, ela vai ser reaproveitada ao invés de criar uma nova.

Uma técnica mais avançada que essa é usando pool de conexões, ou mesmo um ORM, mas que fogem do escopo deste ebook mais introdutório.

Para testar este código, criei uma chamada a esta conexão ao fim do módulo e em seguida você deve ir no arquivo index.js para importar o nosso módulo db.js, como abaixo:

Código 3.8: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 const db = require('./db');
```

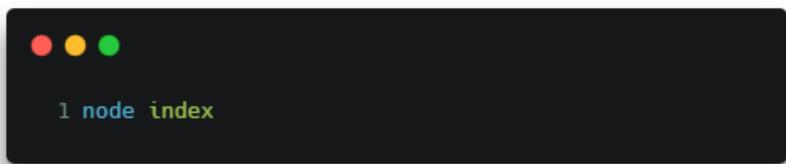
**Nota:** Nota: módulos do Node.js podem ser carregados apenas com o nome do módulo pois o Node procura primeiro na pasta node\_modules. Já módulos criados por você devem ser carregados passando o caminho relativo até eles, neste caso usei './' pois eles estão na mesma pasta.

Nesta linha nós estamos carregando o módulo db que acabamos de criar e guardamos o resultado dele em uma constante.

Ao carregarmos o módulo db em memória, ele vai percorrer todo o seu conteúdo uma única vez antes de colocá-lo em cache, isso vai fazer com que aquela chamada solta do connect() que deixei lá no db.js seja acionada, estabelecendo uma primeira conexão logo de largada na aplicação. Por enquanto esta é a única coisa que vai acontecer e você

consegue ver isso na prática rodando o comando abaixo dentro da mesma pasta do projeto.

Código 3.9: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

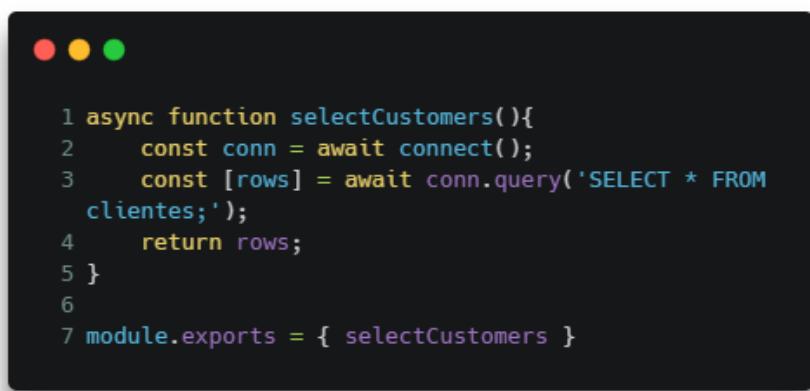
A screenshot of a terminal window with a black background and three colored window control buttons (red, yellow, green) at the top. The text 'node index' is displayed in white font in the center of the window.

A seguir, vamos modificar a nossa aplicação para que ela mostre os clientes cadastrados no banco de dados no console, usando esse db.js que acabamos de criar.

## Listando os clientes

Para conseguirmos fazer uma listagem de clientes, o primeiro passo é ter uma função que retorne todos os clientes em nosso módulo db.js, assim, adicione a seguinte função ao seu db.js, no final do arquivo (substituindo a linha do module.exports):

Código 3.10: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

A screenshot of a terminal window with a black background and three colored window control buttons (red, yellow, green) at the top. The text shows the code for the selectCustomers function in db.js, which uses async/await to connect to the database and execute a SELECT query.

Nesta função selectCustomers nós começamos chamando a função connect que, internamente, vai realizar uma conexão com o banco ou me retornar a conexão já se existente, se houver. Minha única preocupação é chamar ela e tenho a certeza que terei uma conexão para usar. Note o uso do await novamente aqui, pois a linha de baixo depende que a conexão seja estabelecida para avançar e isso me obriga a ter o async antes da function também.

A consulta aqui é bem direta: usamos a conexão pré-estabelecida para fazer uma consulta SQL na tabela de clientes sem filtro algum. O resultado desse SELECT é um array com rows (linhas) e fields (campos/colunas), eu quero só as rows que são retornadas logo em seguida.

Atenção ao fato de que eu modifiquei o module.exports, para agora retornar esta nova função criada, permitindo que a mesma seja chamada por outros módulos da nossa aplicação Node:

```
1 module.exports = { selectCustomers }
```

Agora , vamos programar a lógica que vai usar esta função. Abra o arquivo index.js e edite o código para que fique como abaixo, explicarei na sequência:

Código 6.21: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 (async () => {
2     const db = require("./db");
3     console.log('Começou!');
4
5     console.log('SELECT * FROM CLIENTES');
6     const clientes = await db.selectCustomers();
7     console.log(clientes);
8 })()
```

Note que dei uma incrementada boa aqui, mas nada muito complicado.

Primeiro, temos o require do nosso db, uns logs de marcação e na sequência chamo a função de selectCustomers exportada no módulo db que vai retornar o nosso array de clientes, que vou só jogar no console, mas que em uma aplicação real tenho certeza que você será mais criativo.

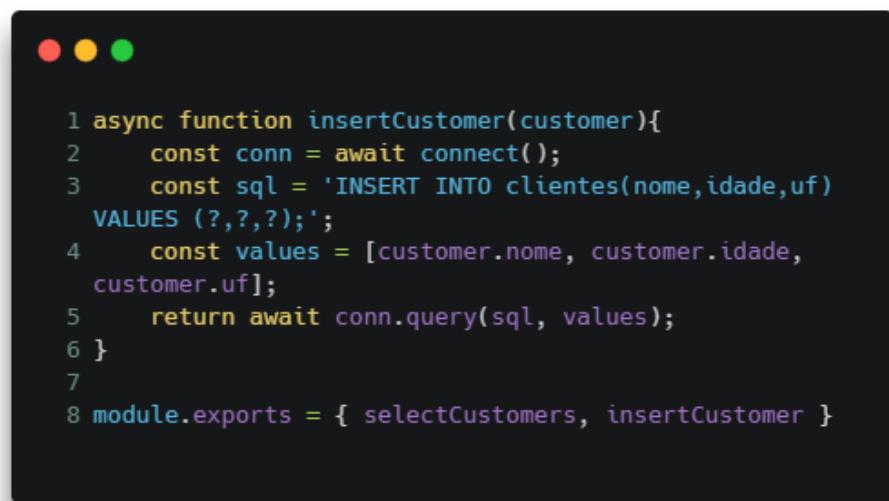
Como estou usando o await aqui também, e ele existe ser usado em funções async, criei uma função async anônima ao redor de todo código que automaticamente é chamada quando mandarmos executar o index.js.

Faça isso agora, rode usando ‘node index’ no terminal e verá que funciona como deveria, printando no console todos os clientes do seu banco de dados, tabela clientes.

## Cadastrando novos clientes

Listar dados é moleza e salvar dados no MySQL não é algo particularmente difícil. Vamos alterar nosso db.js para incluir uma nova função, desta vez para inserir clientes usando a conexão e, novamente, aguardando o seu término:

Código 3.12: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 async function insertCustomer(customer){
2     const conn = await connect();
3     const sql = 'INSERT INTO clientes(nome,idade,uf)
4         VALUES (?,?,?);';
5     const values = [customer.nome, customer.idade,
6         customer.uf];
7     return await conn.query(sql, values);
8 }
9
10 module.exports = { selectCustomers, insertCustomer }
```

Note que já deixei o module.exports ajustado também, pois cada função que criarmos, devemos expor para que ela seja usada em outro arquivo do nosso projeto.

O formato desta função é bem semelhante ao da anterior, de consulta. A novidade aqui é a forma como juntei a string SQL com os campos do objeto cliente recebido por parâmetro. Esta forma de concatenar string é a recomendação oficial do pacote MySQL por ser à prova de SQL Injection, mas antes vou te explicar como ela funciona.

Em cada local da expressão SQL que deve ir uma variável, usamos o

símbolo ‘?’ (sem aspas). Depois, no comando conn.query, passamos como segundo argumento o array de variáveis que vamos usar, na ordem e quantidade necessárias para a expressão funcionar.

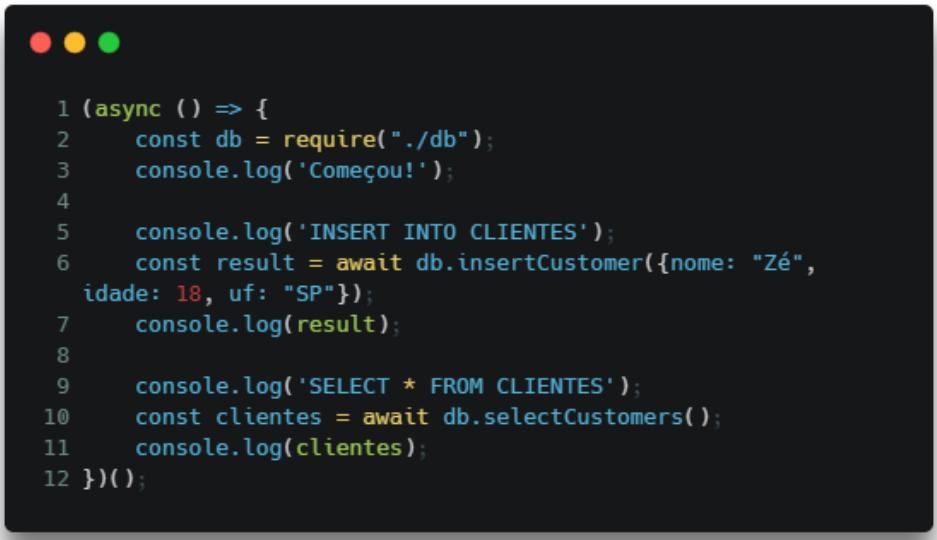
Quando executamos a query desta forma, o pacote mysql2 trata os argumentos recebidos para evitar ataques de SQL Injection.

**Atenção:** Um ataque de SQL Injection é quando um invasor envia comandos SQL através de campos de um formulário, o que faz com que estes comandos burlem a nossa expressão, gerando resultados indesejados e até destruição da nossa base de dados em produção. Mais sobre SQL Injection em [https://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o\\_de\\_SQL](https://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_SQL)

Sendo assim, JAMAIS concatene os campos recebidos diretamente no SQL que será executado, sob risco de deixar seu sistemas vulneráveis a este tipo de ataque.

Agora vamos voltar ao index.js para adicionar uma chamada a esta nova função:

Código 3.13: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
1 (async () => {
2     const db = require("./db");
3     console.log('Começou!');
4
5     console.log('INSERT INTO CLIENTES');
6     const result = await db.insertCustomer({nome: "Zé",
7         idade: 18, uf: "SP"});
8     console.log(result);
9
10    console.log('SELECT * FROM CLIENTES');
11    const clientes = await db.selectCustomers();
12 })();
```

Optei por colocá-lo antes do SELECT, para que fique mais fácil de ver se ele funcionou ou não.

Também optei por printar o seu resultado, para você ver o que ele devolve, embora a maior parte das informações não seja muito útil, somente o número de linhas afetadas na tabela talvez lhe interesse pois, no fundo, se der algum erro na execução do SQL, vai disparar uma exception que vai derrubar essa nossa aplicação bem simples. Sim, você pode embrulhar tudo com try/catch para tratar os erros.

Rode novamente com node index e veja o resultado não apenas no console, mas indo lá no seu banco de dados também.

Result Grid Filter Rows:  Edit:

	id	nome	idade	uf	
▶	2	Luiz	32	RS	
	3	Diego	18	SC	
	4	Pedro	6	RS	
	6	Zé José	19	SP	
	8	Zé	18	SP	
	9	Zé	18	SP	
	10	Zé	18	SP	
		HULL	HULL	HULL	HULL

## Atualizando clientes

Para atualizar clientes (o U do CRUD, Update) não é preciso muito esforço diferente do que estamos fazendo até agora. Primeiro precisamos criar uma nova função no db.js para fazer update, como abaixo:

Código 3.14: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
1 async function updateCustomer(id, customer){  
2     const conn = await connect();  
3     const sql = 'UPDATE clientes SET nome=?, idade=?, uf=?  
    WHERE id=?';  
4     const values = [customer.nome, customer.idade,  
    customer.uf, id];  
5     return await conn.query(sql, values);  
6 }  
7  
8 module.exports = { selectCustomers, insertCustomer,  
    updateCustomer }
```

O processo não é muito diferente do insert, apenas temos de passar o filtro do SQL update para saber qual documento será afetado (neste caso somente aquele que possui o id específico). Também já inclui o module.exports atualizado na última linha para que você não esqueça.

E no index.js, seguimos a mesma estrutura do INSERT também, apenas tome cuidado para informar um ID existente na sua base de dados ao invés do meu '6', ou até mesmo modifique o código para pegar um dos ids retornados pelo SELECT anterior.

Código 3.15: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
● ● ●  
1 console.log('UPDATE CLIENTES');  
2 const result2 = await db.updateCustomer(6, {nome: "Zé José",  
    idade: 19, uf: "SP"});  
3 console.log(result2);
```

E com isso finalizamos o update!

## Excluindo um cliente

Agora em nossa última parte do tutorial, faremos o D do CRUD, D de Delete!

Vamos no db.js adicionar nossa última função, de delete:

Código 3.16: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>

```
● ● ●  
1 async function deleteCustomer(id){  
2     const conn = await connect();  
3     const sql = 'DELETE FROM clientes where id=?;';  
4     return await conn.query(sql, [id]);  
5 }  
6  
7 module.exports = {selectCustomers, insertCustomer,  
    updateCustomer, deleteCustomer}
```

Essa função é bem simples de entender se você fez todas as operações anteriores. Na sequência, vamos criar a chamada para esta função no index.js:

Código 3.17: disponível em <https://www.luiztools.com.br/ebook-mysql-fontes>



```
● ● ●

1 console.log('DELETE FROM CLIENTES');
2 const result3 = await db.deleteCustomer(7);
3 console.log(result3);
```

O resultado de todos esses códigos e testes?

Uma tabela clientes cheia de Zés,ahaha

E com isso finalizamos nosso CRUD!

Espero que tenha gostado desse capítulo, porque eu gostei bastante de escrevê-lo!

*Quer fazer um curso online de Node.js e MySQL com o autor deste livro?*

*Acesse <https://www.luiztools.com.br/curso-fullstack>*

---

# BOAS PRÁTICAS COM MYSQL

4

“

I'm not a great programmer;  
I'm just a good programmer with great habits.

- Kent Beck

”

Quando estamos começando com uma nova tecnologia sempre tudo é muito confuso, complexo e, por que não dizer, assustador?

Trabalho desde 2006 com programação e antes disso eu já programava de maneira não profissional, desde um técnico em eletrônica que fiz em 2004 que incluía programação de microcontroladores para a indústria. Nesse ínterim tive de trocar sucessivas vezes de tecnologias, seja por questões acadêmicas, seja por questões de mercado: Assembly, C/C++, Lisp, Prolog, Visual Basic, Java, Lua, .NET, PHP e, mais recentemente, Node.js.

Igualmente troquei entre vários bancos de dados e em alguns momentos tive até de usar mais de um ao mesmo tempo, como MS SQL Server, MySQL, Oracle, Redis, SQLite, Firebird e MongoDB.

E a cada troca eu volto a ser um calouro, tendo de aprender não apenas sintaxe, semântica, todo o “ferramental” relacionado à tecnologia, mas o mais difícil de tudo: boas práticas!

Como saber se a modelagem da minha tabela está de acordo com os padrões de mercado?

Como tirar o máximo proveito da performance do meu banco de dados através do uso das melhores consultas?

Como ...

São muitas as dúvidas sobre banco de dados e, mesmo com o todo-poderoso Google em nossas mãos, parece que essas respostas estão desconectadas e jogadas cada uma em um canto obscuro da Internet, fazendo com que dê um trabalho enorme encontrá-las.

Não trago aqui verdades universais ou respostas para todas as perguntas que você vai ter sobre MySQL, mas trago um apanhado da experiência coletiva de milhares de desenvolvedores ao redor do globo para formar um guia de boas práticas com MySQL. Pessoas com muito mais conhecimento deste banco do que eu, e algumas dicas minhas também, afinal eu também sou um desses milhares de desenvolvedores!

## Boas Práticas na Modelagem

Vamos começar do começo, ok?

Quando estiver modelando um banco relacional como o SQL Server, você vai estar também definindo boa parte das limitações de performance que seu banco terá durante sua vida útil. A modelagem correta das tabelas (veja bem, correta, não necessariamente normalizada) irá influenciar pesadamente no desempenho do seu sistema, principalmente considerando que a maior parte do tempo de resposta de um sistema é devido aos acessos ao banco de dados.

A regra aqui é: não confundam modelagem ER com um diagrama de classes ou algo do gênero.

Embora muitos desenvolvedores atuais sejam adeptos de teorias “pós-modernistas” de desenvolvimento Code-First, onde primeiro se criam as classes para então gerar as tabelas a partir delas (o que não há nada de errado especificamente), vale ressaltar que a mais elaborada biblioteca de classes, usando e abusando de regras como herança, não-repetição de código, especialização de classes, agregação e composição, não dará um bom banco de dados.

Seguem alguns exemplos.

Primeiro, vamos pegar como base o famigerado cadastro de clientes. Os clientes têm endereço certo?

Muitos programadores já pensam logo de cara em fazer uma tabela de endereços separada da tabela de clientes. Se em seu sistema existe a necessidade de múltiplos endereços para um mesmo cliente (comum em e-commerces), ok. Caso contrário, é inútil. E não vale a desculpa de “mas se eu tiver outra tabela, como fornecedor, que precise de endereço...”, pois a menos que alguns fornecedores morem no mesmo lugar que alguns clientes, não haverá ganho algum com essa abordagem. Isso se chama patternite (doença causa pelo excesso de estudo de design patterns).

Esse pensamento leva a um aumento na complexidade de queries simples como trazer todos os dados de um cliente e preste atenção, quando falo em complexidade não estou falando de ser difícil de programar, mas custoso para o banco de dados em tempo de

processamento, acesso à disco. Toda vez que quiser trazer todos os dados de um cliente, terá de usar um JOIN, o que é desnecessário se você modelar o banco otimizado para as consultas que vai fazer (também falaremos disso mais tarde).

Eu dei um exemplo tosco, de composição, mas um caso mais grave é o de querer usar herança em tabelas. Criar uma tabela Pessoa, com colunas comuns às tabelas Cliente e Fornecedor, por exemplo, levará novamente a trabalho desnecessário na construção e execução de consultas. Não há problema em ter colunas repetidas em tabelas diferentes, o problema é quando se tem dados repetidos em tabelas diferentes.

## Nomenclatura

Aqui a regra geral é: escolha um padrão e siga em frente.

Simplesmente não existe um consenso sobre nomenclatura de bancos, tabelas e colunas. Isso varia de tecnologia para tecnologia, de empresa para empresa e de profissional para profissional, some tudo isso junto e o número de combinações possíveis é infinito.

No fim do dia, o que importa é conseguir se achar sem passar mais tempo procurando o nome da tabela do que escrevendo a consulta para utilizá-la.

Se me permite dizer como eu faço, gosto de bancos com o mesmo nome do sistema (sem acentos e espaços, óbvio) e tabelas no plural, em Pascal Case. Para colunas, também gosto de Pascal Case. Eu sei, na hora de consultar não muda em nada, mas fica mais fácil para eu ler e entender do que se trata.

Para chaves primárias, uso sempre ID, para aquela chave primária auto-incremental marota que não pode faltar nas minhas tabelas. Sim, só ID, se estou na tabela Clientes e tem um ID, de quem você acha que é esse ID?

Para chaves estrangeiras, costumo usar IDNomeTabela, porque aí sim eu preciso saber de quem é esse ID.

Prefixos em tabelas e colunas acho besteira, fazia sentido lá atrás quando tudo era tela preta e não se tinha ícones e pastas para saber o

quê é o quê. Estamos em 2020 e todo mundo deveria usar ferramentas gráficas de gestão de banco, porque usar terminal é antiprodutivo e ARRISCADO. Para fazer uma besteira é muito mais fácil que visualmente e a cores.

Minha opinião.

## Desnormalização

Aprenda as formas normais. Depois esqueça as formas normais. Soa grosseiro, mas em projetos de uso intenso de dados temos de pensar dessa forma. Nada contra a forma acadêmica de projetar bancos, muito pelo contrário, um projeto bem estruturado de um grande ERP ou CRM é digno de colocar em um quadro na parede. Em alguns casos nem caberia em uma única parede, hehehehe. A questão aqui é que quanto mais se divide um banco em tabelas, mais JOINs serão necessários para juntar os dados novamente. Inversamente, quanto mais dados colocarmos em uma mesma tabela mais demorada serão as consultas nesta tabela.

E agora José?

O negócio aqui é analisar cada caso e usar cada técnica de acordo com a sua necessidade para aquele cenário. Ou seja, se os JOINs a um grupo de tabelas estão sendo usados com muita frequência ou até mesmo sempre tem de ser consultados juntos, considere a hipótese de transformar tudo em um tabelão. Como sei que isso soa estranho para muita gente, experimente ao menos criar uma View e notará a diferença.

Como exemplo gosto de citar a tabela de Endereco e de Estado. No Brasil os estados podem ser representados com apenas 2 caracteres, e criando um campo UF no endereço ao invés de um IDEstado tornará seu sistema muito mais eficiente do que fazer JOINs o tempo todo entre Endereco e Estado. Inversamente evite tabelas genéricas usando campos de tipo para definir a categoria do registro armazenado, pois isso geralmente costuma inflar demais as tabelas sem necessidade.

Outra coisa, ao contrário do que muitos acreditam, nem sempre criar chaves estrangeiras para todos os campos em comum entre tabelas é um bom negócio. Os relacionamentos servem para garantir integridade referencial e possuem como efeito colateral o aumento no tempo dos INSERTs, DELETEs e UPDATEs nas tabelas relacionadas. Então pense

bem antes de sair criando chaves a torto e à direito.

## Boas Práticas de Hardware

Para os desenvolvedores, hardware é algo que não recebe tanta importância (até apresentar problemas!). Não colocar o seu banco MySQL em um servidor apropriado pode causar muitas dores de cabeça com a tecnologia. É como diz o ditado: o barato sai caro!

Ao contratar serviços em nuvem, como os da Umbler e outros players, procure se informar do tipo de hardware utilizado, como tecnologias de disco e memória, além de dimensionar corretamente a instância de servidor onde estará o MySQL, é claro.

Ah, e não use ambientes compartilhados de MySQL para aplicações em produção. Nunca! Prefira sempre instâncias e containers dedicados para este serviço ficar executando.

### Memória RAM

Como a maioria dos bancos de dados, o MySQL funciona melhor quando os seus índices e queries mais utilizadas cabem completamente na memória RAM do seu servidor.

Ao lado de um bom SSD (adiante), a quantidade de memória RAM é o fator mais importante quando o assunto é hardware. O resultado de outras otimizações nem se compara com a melhoria de performance que ter RAM suficiente proporciona ao seu sistema.

### Disco

A maioria dos acessos a dados em bases MySQL não são sequenciais e, como resultado, são muito mais velozes em discos de acesso aleatório como SSDs, tanto SATA, PCIe e NVMe. Mesmo unidades flash domésticas possuem uma performance superior quando utilizados em uma máquina com boa quantidade de memória RAM (vide dica anterior) frente aos discos rígidos de alta performance de servidores como SAS e SCSI.

No entanto, enquanto os arquivos de dados beneficiam-se dos SSDs, os arquivos de log do MySQL são bons candidatos para serem

armazenados em discos tradicionais devido ao seu perfil de escrita sequencial, sendo que, quando necessário, o uso de RAID-10 é o mais indicado para a maioria das aplicações.

## Boas Práticas de Performance

Aqui vão algumas dicas gerais que podem auxiliar na performance do banco de dados.

Claro, imagino que você já passou pela seção de hardware, pois boa parte do quesito performance depende de um bom hardware.

Além disso, durante o desenvolvimento da nossa aplicação em Node.js eu falei de reaproveitamento de conexão e de pool de conexões, vale a pena você estudar esses conceitos pois ficar abrindo e fechando conexão o tempo todo, para cada consulta, é um problema grande de performance que você vai ter se colocar em produção assim.

Só não é pior do que ficar abrindo várias conexões e nunca fechando... Esse deveria dar demissão por justa causa, hahaha

### JOINS

Na maioria dos casos em que tenha que juntar dados de duas ou mais tabelas em uma mesma consulta, dê preferência ao uso de JOINs ao invés de subconsultas e tabelas aninhadas. Usando esta dica com a dos índices mais abaixo, lhe darão um ganho muito grande em performance.

Entretanto, o uso de múltiplos JOINs em uma mesma consulta podem acabar com sua performance devido ao excesso de cruzamento entre diferentes tabelas, neste caso, dê uma olhada no item sobre desnormalização de tabelas, no assunto modelagem.

JOINs também são muito beneficiados pela dica a seguir: índices.

## Índices

Boa parte dos problemas de desempenho se resolvem com índices bem construídos. Assim como os livros, que são um conjunto de capítulos com muitas páginas cada, os bancos são conjuntos de tabelas como muitos registros cada. Em ambos nós precisamos ter índices, que são estruturas de apoio para encontrar rapidamente o que estamos procurando.

Note que índices e chaves são coisas diferentes, embora a chaves primárias sejam os únicos índices clusterizados das tabelas. A ideia aqui não é ensinar como construir índices (use o MySQL Workbench e não terá problemas), é dar dicas sobre porque construí-los e quando fazê-lo. Primeiro, toda tabela deve ter um índice clusterizado, que é o índice que rege a ordenação das linhas da tabela. Ou seja, a chave primária.

Fora as regras tradicionais de chaves primárias, como não poder haver repetição, dê preferência por usar um campo numérico como chave, pois os índices trabalham melhor como estes campos (i.e. as consultas serão mais rápidas). Este é o mais básico e elementar dos índices e ele é criado mesmo que você não saiba disso.

Segundo, crie índices nas suas chaves estrangeiras. Não necessariamente em todas, afinal o uso excessivo de índices prejudica a performance dos INSERTs, UPDATEs e DELETEs, mas nas FKs que você costuma fazer JOIN.

Terceiro, crie índices para garantir integridade. Quando você cria um índice para uma coluna (ou grupo delas) e diz que ele é UNIQUE, você não apenas garante que não haja elementos repetidos em sua tabela, como aumenta significativamente a performance de consultas que envolvam as colunas do índice. Isso porque quando o motor de consulta do MySQL sabe que um campo não se repete, ele vai parar de procurar por mais ocorrências daquele campo tão logo encontre a primeira ocorrência, entende?

## Manutenção de Índices

Dê manutenção nos seus índices periodicamente. Conforme você vai inserindo, removendo e atualizando elementos da sua tabela, seus índices vão ficando fragmentados, o que pode prejudicar a performance de uma maneira até pior do que a ausência de índices. Desta forma, lembre-se de dar um Rebuild nos seus índices quando eles estiverem fragmentados, preferencialmente em horários de pouco acesso ao seu sistema para evitar problemas com os usuários. O uso de índices segue os mesmos princípios em todos os bancos relacionais: melhoram a leitura e prejudicam a escrita dos dados. Sendo assim, caso possua índices que não estão sendo utilizados, remova-os, para garantir que eles não atrapalhem a escolha dos planos de execução pelo motor de consulta do MySQL e principalmente não prejudique a performance de inserts, updates, etc.

Outra dica é sempre verificar se a sua tabela não deveria ter um índice composto ao invés de vários índices individuais. Caso você possua consultas que sempre usam as mesmas colunas como filtro, e estas colunas possuem índices que não são usados individualmente, opte por condensá-los em um único índice composto que será muito mais performático.

## Comandos Precisos

Por mais que seja chato escrever comandos completos de SQL, quanto mais preciso e completo for seu comando, mais veloz será sua interpretação pelo motor de busca e consequentemente sua consulta terá melhor desempenho.

Sim, estou falando de milissegundos, mas não esqueça que a cada 1000 milissegundos economizados, é 1 segundo a menos de encheção de paciência! A dica aqui é bem simples, não tenha preguiça ao escrever as queries, você o fará uma vez, mas elas serão executadas milhares de vezes, então vale a pena.

Digite completamente o nome de tabelas, incluindo o schema, evite algumas palavras-chave as quais cito mais tarde e nos SELECTs retorne somente as colunas que serão utilizadas na aplicação.

## Boas Keywords

A regra é que não existem balas de prata. Entretanto, existem diversas palavras-chave que em 90% dos casos podem resolver grandes problemas de performance e às vezes são ignoradas pelos desenvolvedores. Brevemente não esqueça de:

**LIMIT** - retorna um número limitado de registros, muito útil para trazer somente a quantidade de dados que faz sentido para sua aplicação. A menos que queira todas as ocorrências de uma condição consultada, use LIMIT 1, 5, 10, etc conforme o número de registros que queira retornar.

**DISTINCT** - retorna os elementos sem repeti-los, muito útil para eliminar repetições de valores em uma mesma coluna e consequentemente dados inúteis sendo transferidos pela rede.

**COUNT** - retorna a quantidade de elementos para uma dada condição (WHERE). Se quer saber apenas se um elemento existe na tabela, ou quantos elementos de um tipo, use COUNT ao invés de mandar retornar todos os dados dos registros que atendem à sua condição.

## Más Keywords

Da mesma forma que mesmo as palavras acima Não são garantia de sucesso na escrita de boas consultas SQL, as palavras abaixo nem sempre são vilãs.

Cito elas aqui apenas para que se lembre de sempre ponderar sobre sua utilização, para evitar surpresas quando seu sistema começar a ser utilizado de verdade, afinal, quando só você está testando o sistema tudo funciona com boa performance, não é mesmo?!

**LIKE** - compara um trecho de texto dentro de um bloco maior. Se tiver de usar o LIKE evite usar mais de um coringa (%) para que a perda de performance não seja tão grande.

Outra dica que pode ajudar é o uso de índices no campo de texto onde precisará usar likes, como citado no tópico anterior sobre índices. Se você está se perguntando como fazem os buscadores como o Google, tenha a certeza de que eles não usam LIKE em uma grande tabela que

possuía a web inteira dentro. O segredo aqui está em estruturas de dados específicas para armazenar textos de forma distribuída como índices invertidos, matrizes de dispersão, árvores de conhecimento, entre outras. O buraco é muito mais embaixo que um mero LIKE!

**IN** - se você quer retornar os registros cujas condições são múltiplos identificadores, como todos os empregados cujos IDs sejam 1,3,7,45,100, você irá usar um IN, certo?

O problema aqui é que o IN é interpretado pelo motor de busca como uma junção de ORs, ou seja, 'WHERE ID IN (1,3,7,45,100)' é a mesma coisa que 'WHERE ID=1 OR ID=3 OR ID=7 OR ID=45 OR ID=100'. Isto não chega a ser um problema em uma consulta com poucas dezenas de valores no IN, mas tome cuidado quando você chega nas centenas deles. Já experimentei outras abordagens com tabelas temporárias, tabelas em memória, entre outras, com resultados semelhantes. A única solução foi encontrada nos algoritmos que precedem a consulta ao banco, fugindo do uso intensivo de IN.

\* - o famigerado asterisco deve ser evitado pelo simples motivo de que é muito fácil retornar mais colunas do que o necessário usando esse recurso. Outro problema é que o interpretador SQL deve buscar no esquema da tabela as colunas que terão de ser retornadas, antes de realizar a consulta propriamente dita. No final das contas é uma economia porca de tempo e que não traz benefício algum nem mesmo a curto prazo.

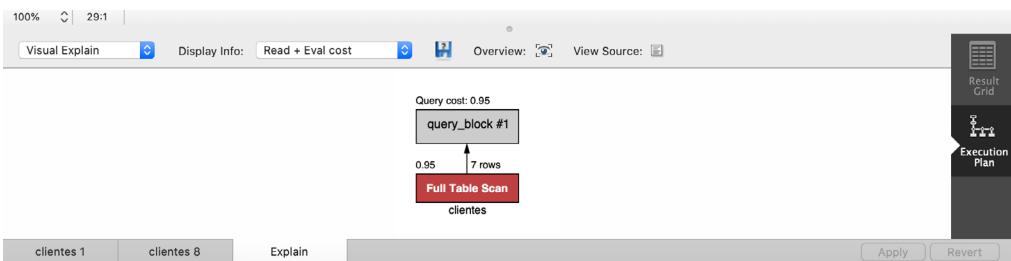
## Evite Negações

Como a maioria das bases de dados, o MySQL não indexa a ausência de valores e condições de negação podem exigir que todos as linhas de uma tabela sejam analisadas para dar um retorno à query. Se a negação é a única condição e ela não é seletiva (por exemplo, pesquisar uma coleção de pedidos onde 99% dos pedidos estão finalizados para encontrar aqueles que não estão), todos os registros serão escaneados em busca das linhas que preencham a negação.

Embora não tenha como você fugir de algumas negações de vez em quando, experimente usar mais o operador de igualdade do que o de diferença, sendo o mais seletivo possível se quiser uma performance maior nas consultas. Modelar o banco já pensando nas consultas que irá fazer ajuda bastante também.

## Plano de Execução

Para cada query importante da sua aplicação e/ou queries que estejam com mau desempenho, analise o plano de execução da mesma usando ferramentas visuais como a fornecida pelo Workbench. Ela permite visualizar planos de execução de uma maneira bem prática e didática, sendo que cada estágio do pipeline de execução é exibido como um nó em uma árvore, como mostra a figura abaixo.



## Boas Práticas de Segurança

Eu não sou especialista em segurança da informação e a minha dica de ouro aqui é ter alguém no seu time que seja, principalmente se a sua aplicação estiver na Internet.

Ainda assim, acho que consigo lhe ajudar com algumas dicas básicas.

### Segurança pelo Ambiente

Escolha empresas de confiança para hospedar o seu banco de dados, sendo que recomendo primariamente os três maiores players de cloud do mundo, nesta ordem: Amazon AWS, Microsoft Azure e Google Cloud Platform.

Eles vão te dar as melhores ferramentas e as configurações iniciais mais seguras existentes.

Geralmente existem dois produtos para cada banco de dados: o com gerenciamento por você e o gerenciado por eles. Na Amazon AWS, você pode pegar qualquer servidor, que eles chamam de EC2, e colocar um MySQL dentro. Aí é você que vai ter de gerenciar tudo.

Ou então, contratar o serviço de banco gerenciado, que eles chamam de RDS, que automaticamente vai ter backup, replicação e muitas outras configurações adicionais que você iria sofrer para fazer na mão.

Obviamente um serviço desses possui seu preço...

Outras dicas envolvem você não abrir o seu MySQL para a Internet, ou seja, somente permitir no firewall do seu servidor a entrada de conexão específicas, geralmente do servidor onde está a aplicação, supondo aqui que você hospede em servidores separados as camadas da sua aplicação. Se está tudo no mesmo servidor, simplesmente bloqueeie no firewall TODAS requisições de entrada que não sejam nas portas 80 (HTTP) e 443 (SSL).

Não, você não deve acessar o seu servidor MySQL de produção da sua casa o tempo todo. Se acontecer uma emergência, vá no firewall, crie uma liberação para o seu IP e faça o acesso. Depois remova esta liberação.

## Segurança pela Obscuridade

Talvez você tenha ouvido falar na faculdade que segurança pela obscuridade é tolice.

Pois é, ela não garante nada, mas dificulta. Tipo aquelas trancas de volante para carros, sabe? Não impede, mas dificulta e talvez faça o bandido escolher outro mais fácil.

Aqui as dicas que dou é que o host do seu banco de dados não seja algo óbvio (`mysql.minhaempresa.com`) e que a porta não seja a padrão (3306). Principalmente este último.

A porta 3306 de qualquer MySQL aberto para a Internet é atacada constantemente por robôs de hackers, mesmo se sua empresa for pequena. Sério, acredite em mim, e não faço ideia de como eles descobrem tão rápido que você tem um MySQL exposto.

Novamente, mudar a porta do seu banco de dados não impede os atacantes, mas dificulta o seu trabalho pois eles sempre vão DIRETO na 3306 na esperança de que tenha um MySQL aberto lá.

## Use os drivers mais recentes

MySQL possui drivers de conexão para dezenas de linguagens de programação. Estes drivers são criados pelos mesmos engenheiros que mantém o kernel do banco de dados, mas tendem a ser atualizados de maneira mais constante do que o mesmo, tipicamente a cada dois meses. Sempre use a versão mais recente dos drivers para aproveitar ao máximo a estabilidade, segurança, performance e recursos implementados regularmente.

## Backup

Faça backup. Ponto.

Novamente citando os grandes players de cloud, eles permitem a você ter rotinas de snapshots, que são backups do seu servidor inteiro, inclusive diariamente.

Novamente, isso tem um custo.

Sim, backup é item de segurança, chamado Disponibilidade.

*Quer fazer um curso online de Node.js e MySQL com o autor deste livro?*

Acesse <https://www.luiztools.com.br/curso-fullstack>

# SEGUINDO EM FRENTE

---

“

A code is like love, it has created with clear intentions at the beginning, but it can get complicated.

- *Gerry Geek*

”

Este livro termina aqui.

Pois é, certamente você está agora com uma vontade louca de aprender mais e criar aplicações incríveis com MySQL, que resolvam problemas das empresas e de quebra que o deixem cheio de dinheiro na conta bancária, não é mesmo?

Pois é, eu também! :)

Este livro é pequeno se comparado com o universo de possibilidades que este banco de dados nos traz. Como professor, costumo dividir o aprendizado de alguma tecnologia (como MySQL) em duas grandes etapas: aprender o básico e executar o que foi aprendido no mercado, para alcançar os níveis intermediários e avançados. Acho que este guia atende bem ao primeiro requisito, mas o segundo só depende de você.

De nada adianta saber muita teoria se você não aplicar ela. Então agora que terminou de ler este livro e já conhece uma série de formas de criar aplicações com esta fantástica tecnologia, inicie hoje mesmo (não importa se for tarde) um projeto de aplicação que a use. Caso não tenha nenhuma ideia, cadastre-se agora mesmo em alguma plataforma de freelancing. Mesmo que não ganhe muito dinheiro em seus primeiros projetos, somente chegarão os projetos grandes, bem pagos e realmente interessantes depois que você tiver experiência.

Me despeço de você leitor com uma sensação de dever cumprido. Caso acredite que está pronto para ainda mais tutoriais bacanas, sugiro dar uma olhada em meu blog <https://www.luiztools.com.br>.

Caso tenha gostado do material, indique esse ebook a um amigo que também deseja aprender a programar com MySQL. Não tenha medo da concorrência e abrace a ideia de ter um sócio que possa lhe ajudar nos projetos.

Caso não tenha gostado tanto assim, envie suas dúvidas, críticas e sugestões para  [contato@luiztools.com.br](mailto: contato@luiztools.com.br) que estou sempre disposto a melhorar.

Um abraço e até a próxima!

# MEUS CURSOS

## Curso online NODE.JS e MONGODB

[SAIBA MAIS...](#)

## Curso online Scrum e métodos Ágeis

[SAIBA MAIS...](#)

## Curso online Jira

[SAIBA MAIS...](#)

## Curso online Web Full Stack JavaScript

[SAIBA MAIS...](#)

## Curso online React Native com Firebase

[SAIBA MAIS...](#)

Conheça todos os meus cursos

# MEUS LIVROS



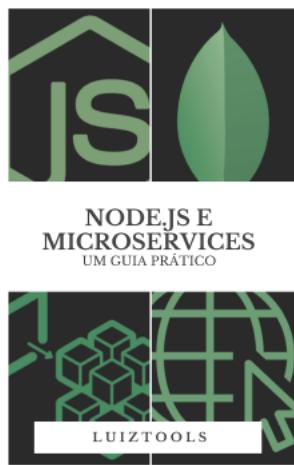
Programação  
Web com Node.js

[SAIBA MAIS...](#)



Programação  
Web com Node.js

[SAIBA MAIS...](#)



NODEJS E  
MICROSERVICES  
UM GUIA PRÁTICO



MongoDB  
para  
Iniciantes

POR LUIZTOOLS

MongoDB  
para Iniciantes

[SAIBA MAIS...](#)



Scrum e  
Métodos Ágeis

[SAIBA MAIS...](#)



Agile Coaching

[SAIBA MAIS...](#)



Criando apps  
para empresas  
com Android

[SAIBA MAIS...](#)



Java para  
iniciantes

[SAIBA MAIS...](#)

## Conheça todos os meus livros

Aproveita e segue nas redes sociais:

