

**UNISENAC-PELOTAS**

# **FUNDAMENTOS DO BACKEND**

**PROF ALINE TIMM**



# O QUE É BACKEND?

## 1. Modelo Cliente-Servidor e a Arquitetura de 3 Camadas:

- Analogia: O Restaurante.
  - Cliente (Frontend): O salão, onde o cliente senta, olha o cardápio (site) e faz um pedido. É o que o usuário vê no navegador (HTML, CSS, JS).
  - Servidor (Backend): A cozinha. Fica escondida, recebe os pedidos do salão, prepara a comida e a envia de volta. É o nosso código Node.js.
  - Banco de Dados: A despensa/cofre. Onde os ingredientes (dados) são armazenados de forma segura. Apenas a cozinha tem acesso.
- Fluxo: O cliente faz um pedido (requisição HTTP) -> O garçom leva para a cozinha -> A cozinha (backend) pega os ingredientes (dados) na despensa (banco de dados), prepara o prato (processa a lógica) e entrega ao garçom -> O garçom entrega o prato (resposta HTTP) ao cliente, que o vê em sua mesa (tela).

# O QUE É UMA API?

- **API (Application Programming Interface):**

Uma API, ou Interface de Programação de Aplicações, é formalmente definida como um conjunto de regras e rotas que um sistema de backend expõe. Sua finalidade é permitir que aplicações cliente, como o frontend, saibam precisamente quais operações podem ser solicitadas e como estruturar essas solicitações.

- **API REST (Representational State Transfer):**

API REST é um estilo de arquitetura utilizado para a criação de APIs de maneira padronizada e previsível, para realizar a comunicação entre sistemas através do protocolo HTTP, utilizando métodos como GET, POST, PUT e DELETE para realizar operações em recursos. Exemplos de uso incluem a exibição de um feed de notícias (GET), a publicação de um novo post em uma rede social (POST), a atualização de um perfil de usuário (PUT) e a exclusão de um comentário (DELETE). As APIs REST enviam e recebem informações em formatos como JSON.



# O PROTOCOLO HTTP:

Em uma definição formal, é um protocolo de comunicação que aceita **métodos** como GET, e POST.

**1. Método:** O que o cliente quer fazer?

- a. GET: "Me traga informações." (Ex: Ver um perfil, buscar um produto).
- b. POST: "Anote este novo pedido." (Ex: Criar um novo usuário, postar uma foto).
- c. PUT/PATCH: "Altere este pedido que já fiz." (Ex: Atualizar um endereço).
- d. DELETE: "Cancele este pedido." (Ex: Apagar uma conta).

**2. URL/Endpoint:** O "endereço" da cozinha para o qual o pedido é direcionado (Ex: /usuarios, /produtos/123).

**3. Cabeçalhos (Headers):** Informações extras sobre o pedido (Ex: "O cliente prefere o prato em formato JSON").

**4. Corpo (Body):** Os detalhes do pedido (usado em POST e PUT). (Ex: Os dados de um novo usuário).

# O QUE VEM NA RESPOSTA?

1. **Resposta (Response):** O "prato" que o servidor devolve. Contém:
  - a. Status Code: Uma nota sobre o resultado do pedido.
  - b. 2xx (Sucesso):
  - c. 200 OK: "Pedido entregue com sucesso."
  - d. 201 Created: "Novo item criado com sucesso."
  - e. 4xx (Erro do Cliente):
  - f. 400 Bad Request: "Seu pedido veio com informações erradas."
  - g. 404 Not Found: "O prato que você pediu não existe no nosso cardápio."
  - h. 5xx (Erro do Servidor):
  - i. 500 Internal Server Error: "Deu um problema na nossa cozinha! Não foi sua culpa."
2. **Cabeçalhos (Headers):** Informações sobre a resposta.
3. **Corpo (Body):** O "prato" em si (HTML, JSON, etc.).

# O QUE SÃO ENDPOINTS?

Um endpoint é a combinação de uma URL (o caminho para um recurso) e um método HTTP (a ação a ser executada). A URL especifica "onde" a operação está localizada (ex:/usuarios), enquanto o método especifica "o que" se deseja fazer.

Exemplo:

- Endpoint
  - Método: GET (Ação: "Me traga...")
  - URL: /tarefas (Endereço: "o que quero que traga")
  - Resultado: GET /tarefas é, por exemplo o endpoint que lista todas as tarefas.





# ATIVIDADE PRÁTICA

- Explorando APIs Reais: Usar o próprio navegador para:
  - Acessar a API do GitHub: <https://api.github.com/users/google>. Analisar o JSON retornado.
- - Abrir as "Ferramentas de Desenvolvedor" (F12 ou Ctrl+Shift+i) em qualquer site (ex: senac.rs), ir na aba "Rede" (Network), recarregar a página e analisar as requisições HTTP que o site faz para carregar seus dados.

# TÓPICOS IMPORTANTES:

## 1. Síncrono vs. Assíncrono:

- Ainda usando como analogia um restaurante:
  - Síncrono: O atendente pega seu pedido, prepara seu café, te entrega e SÓ ENTÃO atende a próxima pessoa. A fila inteira fica bloqueada.
  - Assíncrono: O atendente pega seu pedido e te dá uma senha. Enquanto seu café está sendo feito, ele já vai atendendo outras pessoas. Quando seu café fica pronto, ele chama sua senha. Ninguém fica bloqueado. **Node.js funciona assim!**





# INTEGRAÇÃO COM BANCO DE DADOS

- Vamos agora mudar de exemplo, vamos supor que a biblioteca do SENAC está passando por uma atualização e ganhará um novo site, e para isso precisará de um banco de dados integrado.

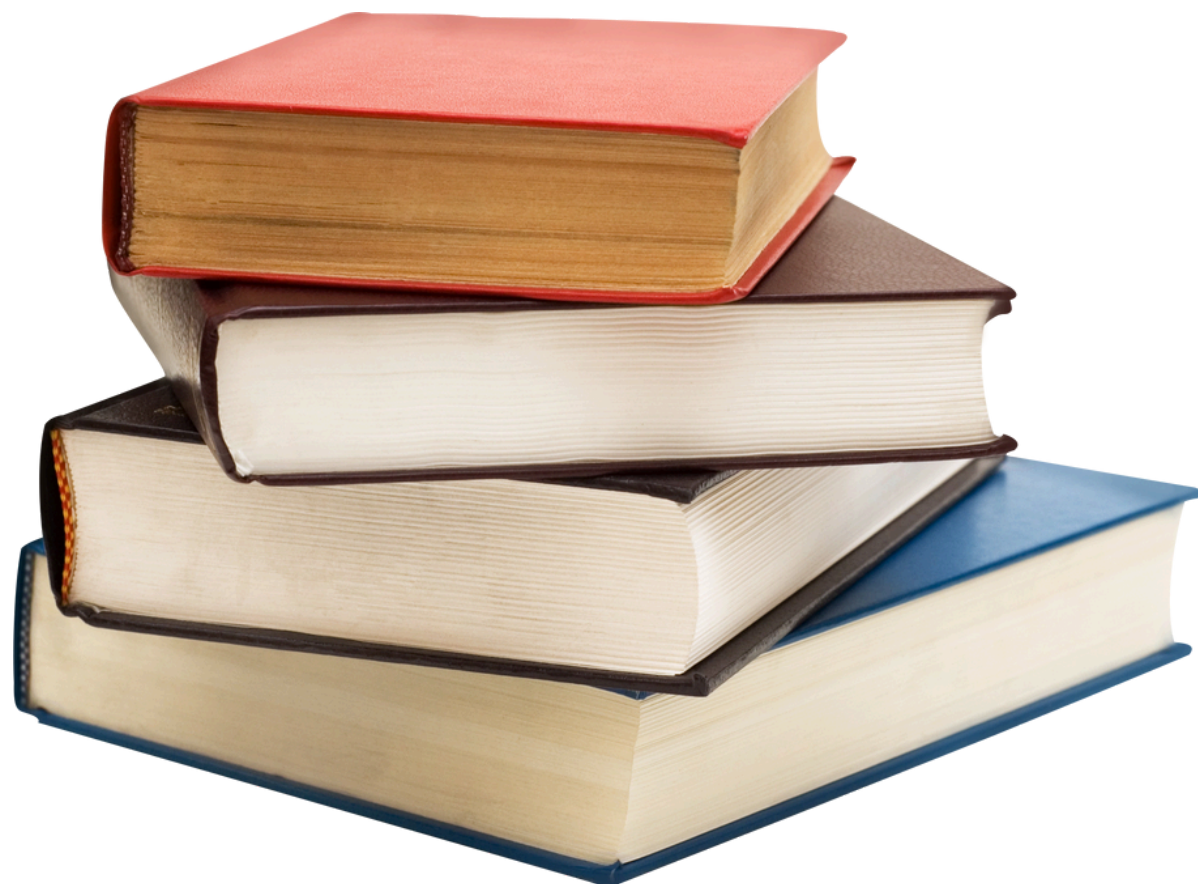
Todo banco de dados começa com uma necessidade do mundo real. A bibliotecária nos diz que precisa organizar os livros, autores e editoras. Antes de escrever qualquer linha de código ou criar qualquer tabela, precisamos entender o que ela precisa armazenar

- A Necessidade: Um sistema para gerenciar os livros de uma biblioteca.
- Os Requisitos: Precisamos guardar informações sobre os livros, quem os escreveu (autores) e quem os publicou (editoras).



# O MODELO CONCEITUAL

Modelo Conceitual, e a ferramenta que usamos para isso é o Diagrama Entidade-Relacionamento (DER). Ele foi definido por Peter Chen em 1976 para nos ajudar a visualizar os "objetos" do nosso sistema e como eles se conectam.



# DESENHANDO A IDEIA (O MODELO CONCEITUAL)

Entidades são os principais "objetos" sobre os quais queremos guardar informações. No nosso caso da biblioteca, as entidades são claras:

- LIVRO
- AUTOR
- EDITORA

No diagrama, representamos cada entidade com um retângulo.

b) Definindo os Atributos:

Atributos são as características de cada entidade. Por exemplo, o que precisamos saber sobre um livro?

- LIVRO: ISBN, título, número de páginas, preço.
- AUTOR: Código, nome, formação.
- EDITORA: Código, nome, endereço.

Um atributo especial é o identificador (ou chave), que é um valor único para cada registro, como o ISBN de um livro.

c) Criando os Relacionamentos:

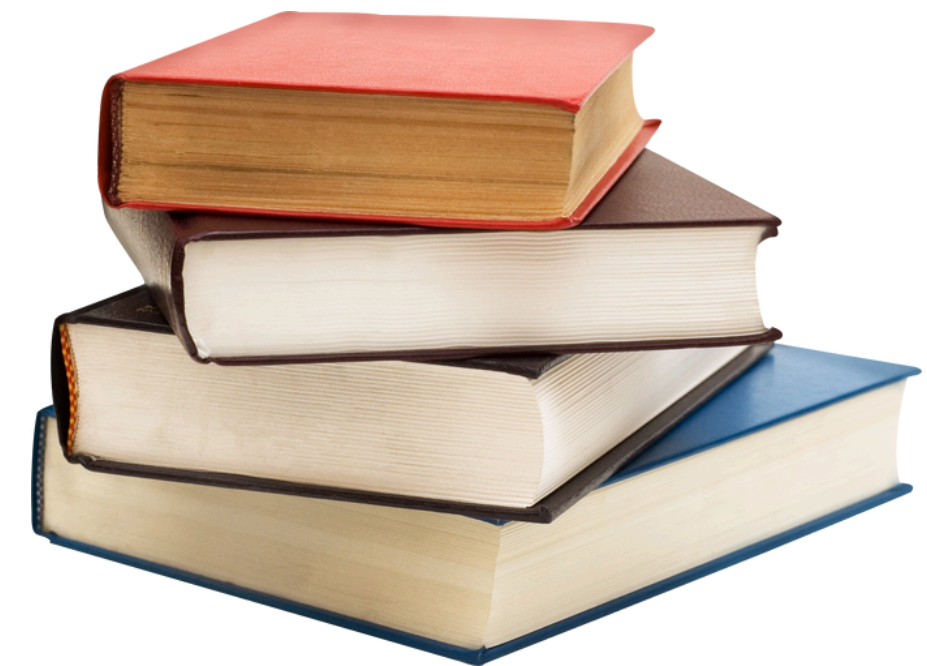
Como essas entidades se conectam?

Um

- AUTOR escreve um ou vários LIVROS.

Uma

- EDITORA publica vários LIVROS, mas um livro é publicado por apenas uma editora.



# CRIANDO O MODELO LÓGICO

Precisamos traduzir nosso diagrama para algo que o banco de dados entenda: tabelas. Esse processo é chamado de: Modelo Lógico.

**Aqui estão as regras básicas para essa tradução:**

**Cada entidade vira uma tabela:**

Teremos uma tabela LIVRO, uma tabela AUTOR e uma tabela EDITORA.

**Cada atributo vira uma coluna:**

A tabela LIVRO terá as colunas ISBN, Título, Num\_Paginas, etc...

**O identificador vira a Chave Primária (PK):**

Esta é uma coluna (ou conjunto de colunas) que identifica unicamente cada linha da tabela, como a coluna ISBN na tabela LIVRO. Ela não pode ter valores repetidos ou nulos.



# EXEMPLO: ENTIDADE LIVRO VIRA TABELA LIVRO

Atributos agora são  
colunas

Estas colunas são  
nossas chaves  
primárias

ISBN	TITULO	NUM_PAGINAS	GÊNERO	CLASSIFICAÇÃO
9788535914849	Sapiens: Uma Breve História da Humanidade – Yuval Noah Harari	472	História / Ciências Humanas	★ 4,4
9788543101770	O Poder do Hábito – Charles Duhigg	408	Desenvolvimento Pessoal / Psicologia	★ 4,2
9788535933949	A Revolução dos Bichos – George Orwell	152	Ficção / Distopia	★ 4,2
9786559790246	A Guerra dos Tronos (As Crônicas de Gelo e Fogo, vol. 1) – George R. R. Martin	600	Fantasia	★ 4,5
9788535919714	Ensaio sobre a Cegueira – José Saramago	312	Ficção Literária	★ 4,3



# E OS RELACIONAMENTOS?

Eles são implementados usando Chaves Estrangeiras (FK). Uma chave estrangeira é uma coluna em uma tabela que faz referência à chave primária de outra tabela, criando uma ligação.

No nosso caso, um livro é publicado por uma editora. Para representar isso, a tabela LIVRO terá uma coluna extra, por exemplo Cod\_Editora\_FK, que guardará o código da editora que o publicou.



EXEMPLO: NA TABELA LIVRO

ADICIONAMOS O CÓDIGO DA EDITORA

ISBN	TÍTULO	NUM_PAGINAS	GÊNERO	CLASSIFICAÇÃO	Cod_Editora_FK
9788535914849	Sapiens: Uma Breve História da Humanidade – Yuval Noah Harari	472	História / Ciências Humanas	4,4	2
9788543101770	O Poder do Hábito – Charles Duhigg	408	Desenvolvimento Pessoal	4,2	4
9788535933949	A Revolução dos Bichos – George Orwell	152	Ficção / Distopia	4,2	1
9786559790246	A Guerra dos Tronos (As Crônicas de Gelo e Fogo, vol. 1) – G. R. R. M	600	Fantasia	4,5	4
9788535919714	Ensaio sobre a Cegueira – José Saramago	312	Ficção Literária	4,3	5



E NA TABELA EDITORA

TEMOS OS DADOS CORRESPONDENTES

Cod_Editora	Nome_Editora	País
1	Companhia das Letras	Brasil
2	Objetiva	Brasil
3	Civilização Brasileira	Portugal
4	HarperCollins	EUA
5	Relógio d'Água	Portugal



# O CÓDIGO SQL – DDL

Finalmente, chegamos ao ponto de criar nosso banco de dados! Para isso, usamos uma linguagem chamada SQL (Structured Query Language). O SQL tem subconjuntos de comandos, e o que usamos para criar e definir a estrutura do banco é o DDL (Data Definition Language).

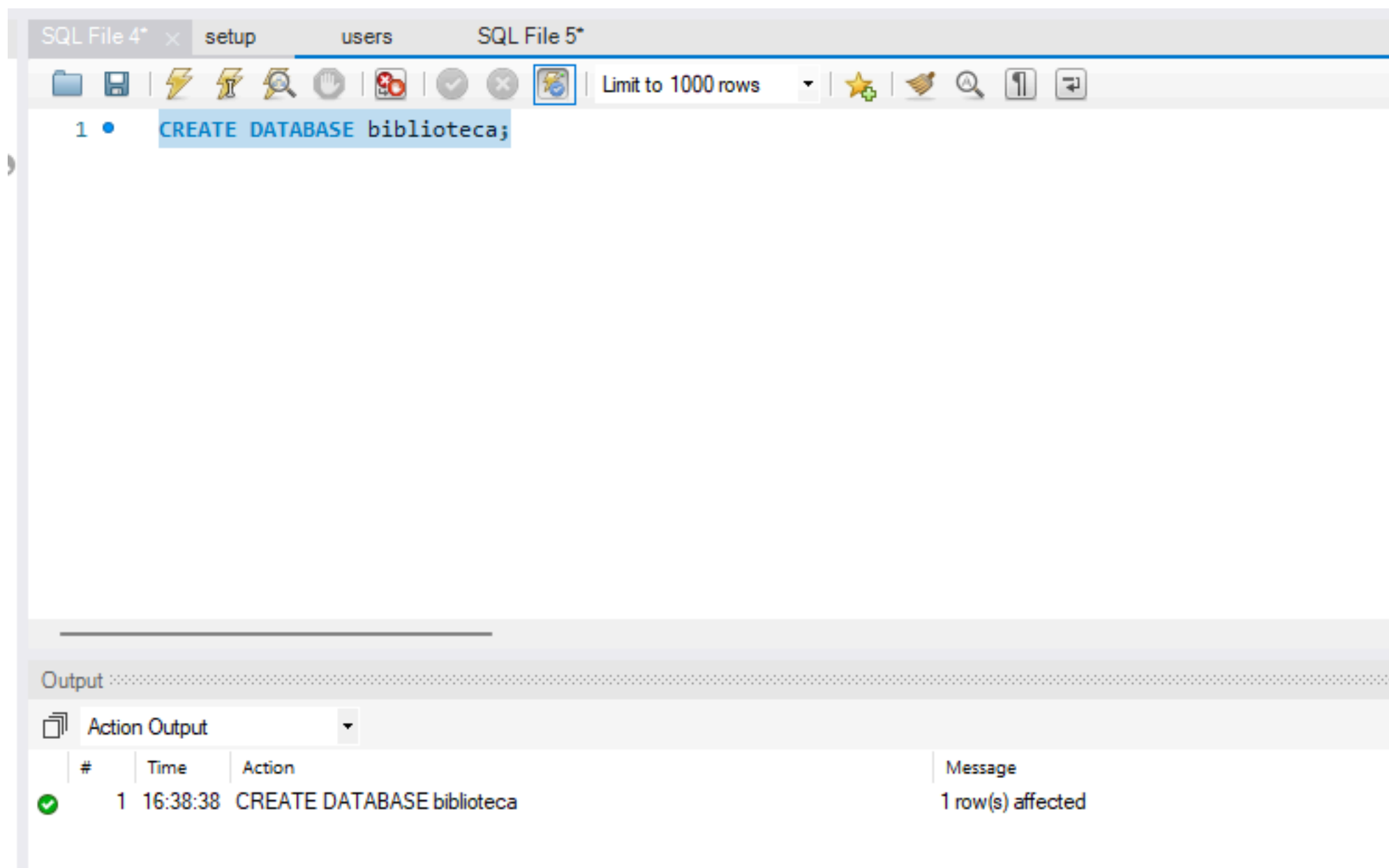
## **a) Criando o Banco de Dados:**

Primeiro, consiga uma interface, a maneira mais fácil é baixar e instalar o MySQL Workbench junto com o MySQL Community Server do site oficial da Oracle/MySQL. O instalador oficial geralmente oferece a opção de instalar os dois juntos. Após instalar, abra o MySQL Workbench, conecte-se ao seu banco de dados local (o instalador te ajudará a configurar isso com um usuário e senha) e você verá uma tela pronta para receber seus comandos SQL. É ali que você vai digitar e executar:

# O CÓDIGO SQL – DDL

## b) Execute o Comando:

Após instalar, abra o MySQL Workbench, conecte-se ao seu banco de dados local (o instalador te ajudará a configurar isso com um usuário e senha) e você verá uma tela pronta para receber seus comandos SQL. É ali que você vai digitar e executar: **CREATE DATABASE biblioteca;**



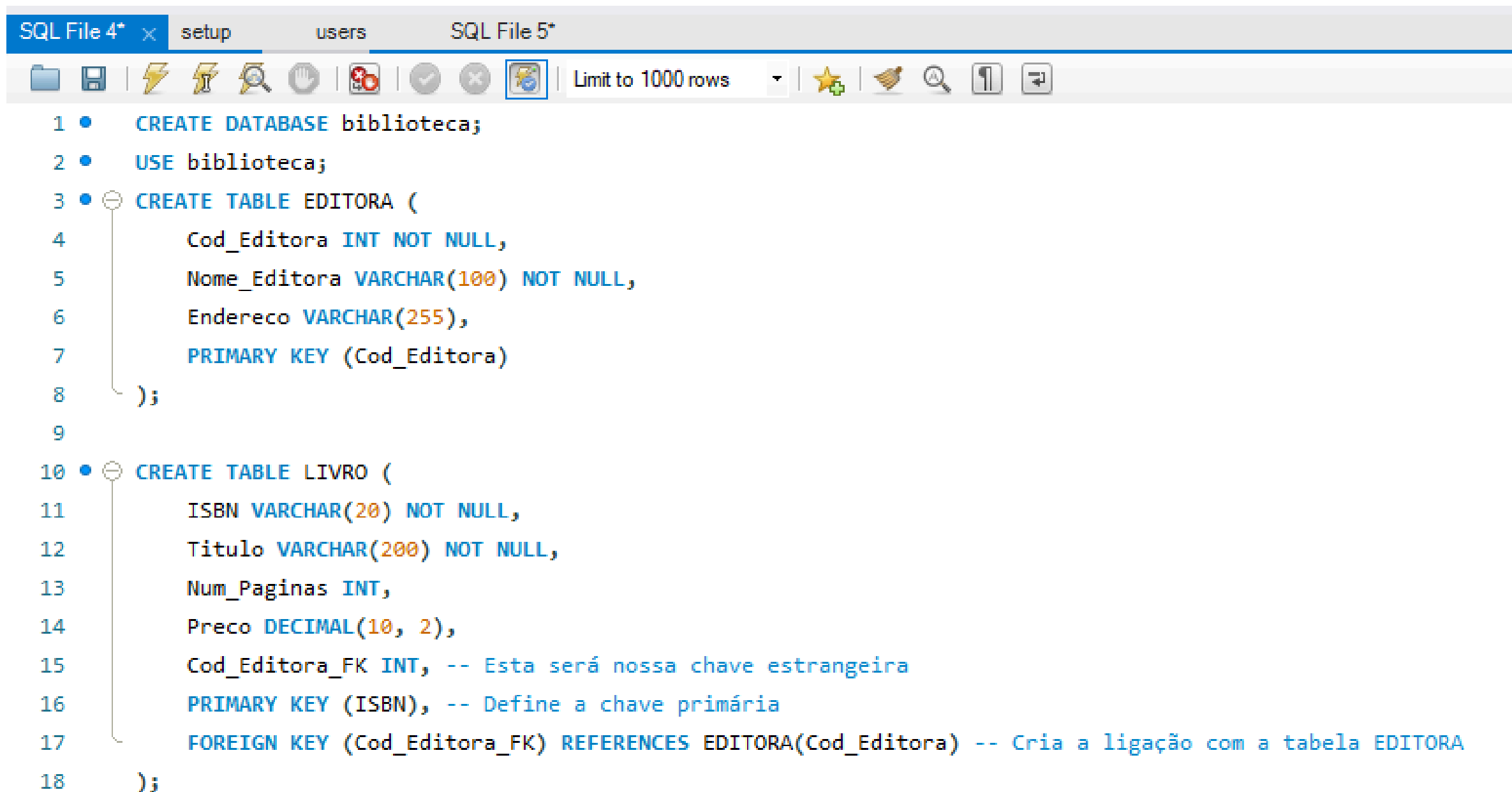


# O CÓDIGO SQL – DDL

## b) Criando as Tabelas:

Agora, usamos o comando

CREATE TABLE para construir cada tabela que planejamos, definindo suas colunas e os tipos de dados que elas armazenarão. Os tipos de dados especificam que tipo de informação uma coluna pode conter, como INT para números inteiros, VARCHAR(100) para textos de até 100 caracteres ou DATE para datas.

A screenshot of a SQL IDE interface. The top bar shows three tabs: 'SQL File 4\*', 'setup', and 'users'. Below the tabs is a toolbar with various icons for file operations, execution, and search. The main area displays SQL code for creating a database and two tables. The code is as follows:

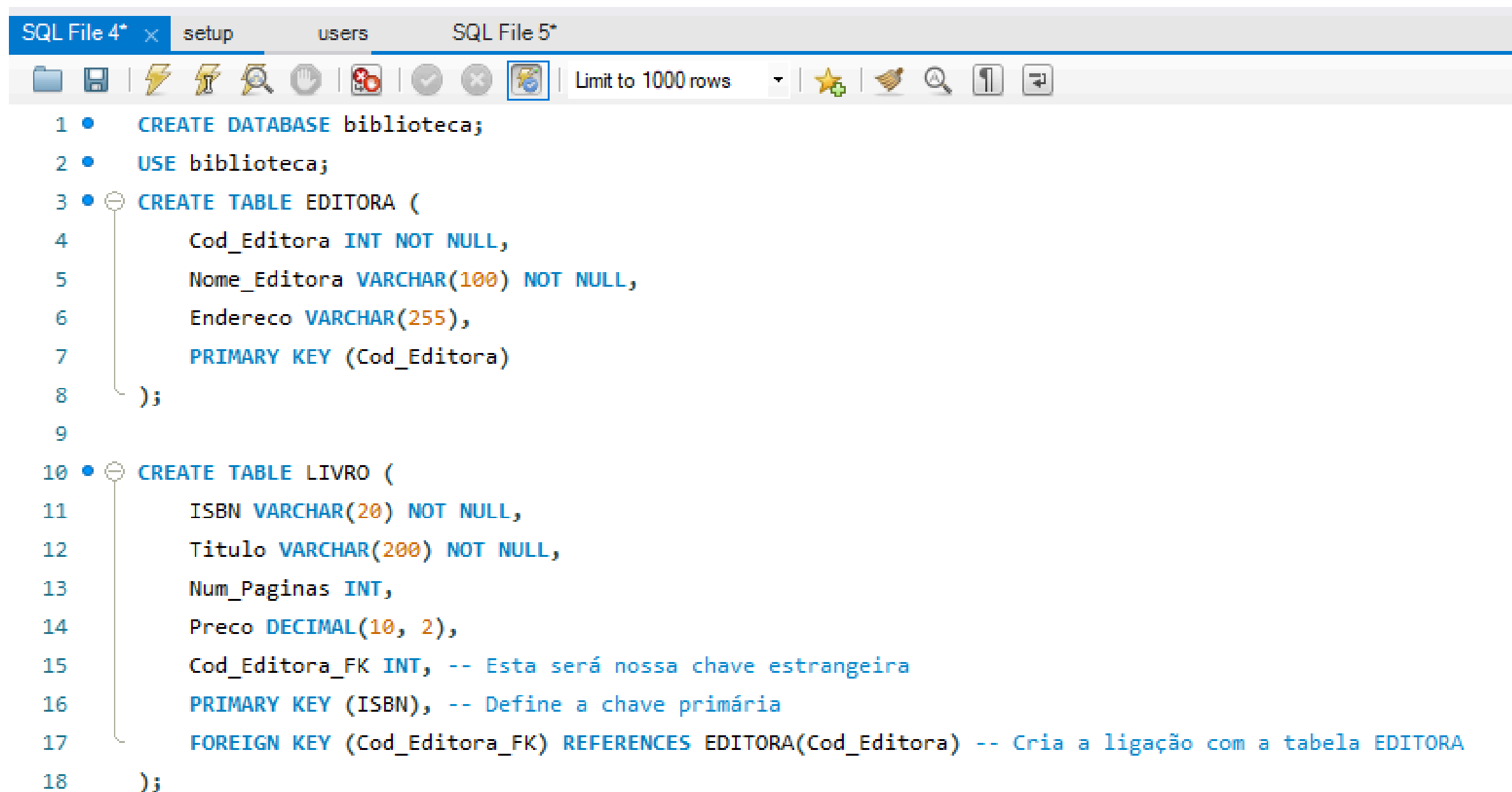
```
1 • CREATE DATABASE biblioteca;
2 • USE biblioteca;
3 • CREATE TABLE EDITORA (
4     Cod_Editora INT NOT NULL,
5     Nome_Editora VARCHAR(100) NOT NULL,
6     Endereco VARCHAR(255),
7     PRIMARY KEY (Cod_Editora)
8 );
9
10 • CREATE TABLE LIVRO (
11     ISBN VARCHAR(20) NOT NULL,
12     Titulo VARCHAR(200) NOT NULL,
13     Num_Paginas INT,
14     Preco DECIMAL(10, 2),
15     Cod_Editora_FK INT, -- Esta será nossa chave estrangeira
16     PRIMARY KEY (ISBN), -- Define a chave primária
17     FOREIGN KEY (Cod_Editora_FK) REFERENCES EDITORA(Cod_Editora) -- Cria a ligação com a tabela EDITORA
18 );
```

# O CÓDIGO SQL – DDL

## b) Criando as Tabelas:

Agora, usamos o comando

CREATE TABLE para construir cada tabela que planejamos, definindo suas colunas e os tipos de dados que elas armazenarão. Os tipos de dados especificam que tipo de informação uma coluna pode conter, como INT para números inteiros, VARCHAR(100) para textos de até 100 caracteres ou DATE para datas.



```
SQL File 4* x  setup  users  SQL File 5*
Limit to 1000 rows

1 • CREATE DATABASE biblioteca;
2 • USE biblioteca;
3 • CREATE TABLE EDITORA (
4     Cod_Editora INT NOT NULL,
5     Nome_Editora VARCHAR(100) NOT NULL,
6     Endereco VARCHAR(255),
7     PRIMARY KEY (Cod_Editora)
8 );
9
10 • CREATE TABLE LIVRO (
11     ISBN VARCHAR(20) NOT NULL,
12     Titulo VARCHAR(200) NOT NULL,
13     Num_Paginas INT,
14     Preco DECIMAL(10, 2),
15     Cod_Editora_FK INT, -- Esta será nossa chave estrangeira
16     PRIMARY KEY (ISBN), -- Define a chave primária
17     FOREIGN KEY (Cod_Editora_FK) REFERENCES EDITORA(Cod_Editora) -- Cria a ligação com a tabela EDITORA
18 );
```

# MAS E COMO FAZER A INTEGRAÇÃO?

Até agora o que vimos foi o que chamamos de: Modelagem de Dados

## O Processo de Modelagem (Revisão):

- Modelo Conceitual: A ideia inicial. Identificar as Entidades (Livro, Autor), Atributos (título, nome) e Relacionamentos.
- Modelo Lógico: Transformar as entidades em Tabelas com Chaves Primárias e Estrangeiras.

O resultado dessa modelagem é o CREATE TABLE que você executa no MySQL. Ele prepara-se para receber os dados.



# AGORA, VAMOS FOCAR NO COMO A INTEGRAÇÃO ACONTECE

O backend não guarda os dados. Ele os busca no banco de dados em tempo real toda vez que o frontend pede.

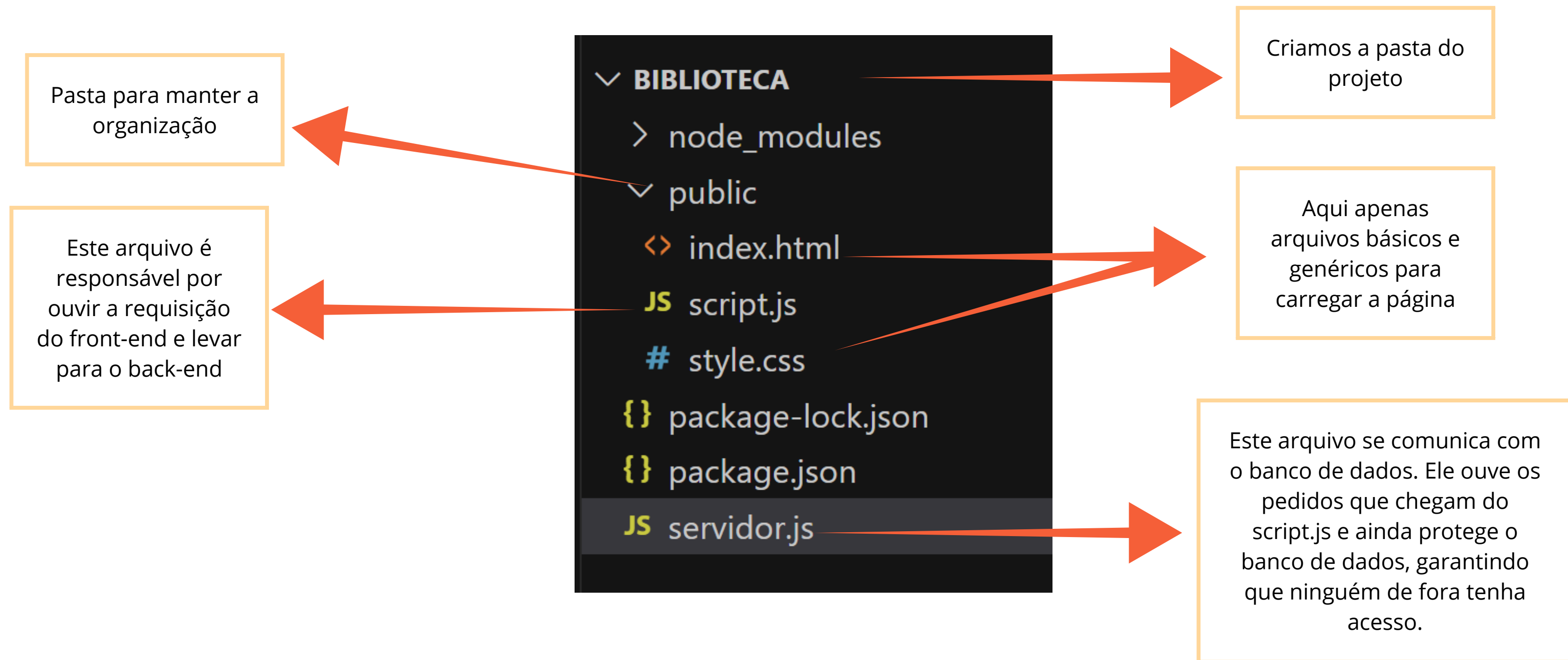
O Fluxo de uma Requisição de Busca e o que chamamos de:

## INTEGRAÇÃO COM O BANCO DE DADOS



Para integrar um banco de dados na web, a abordagem mais comum envolve o uso de uma linguagem de programação server-side (como PHP, Python, Java, ou Node.js)

# VAMOS USAR COMO EXEMPLO O NOSSO SISTEMA DE BUSCA DE LIVROS DE UMA BIBLIOTECA:





# NO DETALHE: FOCANDO EM QUEM FAZ A COMUNICAÇÃO – SCRIPT.JS

O arquivo script.js contém o código JavaScript que é executado no lado do cliente (client-side), ou seja, diretamente no navegador do usuário. Sua função principal é prover interatividade e comportamento dinâmico a um documento HTML, que, por si só, é estático. Ele é o responsável por toda a lógica de interface do usuário (UI) e pela comunicação assíncrona com o servidor.

- **Para que ele serve?**

Ele serve para manipular o Document Object Model (DOM), gerenciar eventos de usuário e iniciar requisições de rede para buscar ou enviar dados, desacoplando a interface do usuário da lógica de negócio que reside no backend.

# O QUE COLOCAMOS DENTRO DELE E POR QUÊ? - SCRIPT.JS

## 1. Seletores de Elementos (getElementById):

- Terminologia Técnica: Referenciamento de Nós do DOM (DOM Node Selection).
- Função: Antes de poder interagir com a página, o script precisa de referências aos objetos que representam os elementos HTML. Para isso, utilizamos métodos da API do DOM, como `document.getElementById()` ou `document.querySelector()`. Esses métodos percorrem a árvore do DOM para localizar e retornar um objeto de nó específico. Armazenamos essas referências em variáveis para acessá-las eficientemente, seja para ler o valor de um campo de entrada (`<input>`) ou para inserir conteúdo em uma `<div>`.
- 

## 2. Ouvintes de Evento (addEventListener):

- Terminologia Técnica: Manipulação de Eventos (Event Handling).
- Função: A interatividade em uma página web é orientada a eventos. O método `addEventListener()` anexa uma função (callback function) a um elemento do DOM, que será invocada de forma assíncrona sempre que o evento especificado (ex: `click`, `keyup`, `submit`) ocorrer naquele elemento. Isso estabelece um mecanismo de escuta que permite ao nosso código reagir às ações do usuário em tempo real, sem bloquear a thread principal do navegador.

# O QUE COLOCAMOS DENTRO DELE E POR QUÊ? - SCRIPT.JS

- A Chamada para a API (fetch):
  - Terminologia Técnica: Requisição HTTP Assíncrona e Comunicação com API.
  - Função: Esta é a principal forma de integração do client-side com o server-side. A API fetch() inicia uma requisição HTTP assíncrona para um endpoint específico no backend (ex: /api/livros/pesquisar). A requisição envia dados, como termos de busca, através de query parameters na URL. Por ser uma operação assíncrona, a interface do usuário permanece responsiva enquanto o script aguarda a resposta do servidor. O fetch() retorna uma Promise, que é resolvida com o objeto Response assim que o servidor responde. A comunicação é baseada no protocolo HTTP, e os dados trocados são comumente serializados em formato JSON.
  -
- Manipulação do DOM (innerHTML, createElement):
  - Terminologia Técnica: Manipulação Dinâmica do DOM (Dynamic DOM Manipulation).
  - Função: Após receber a resposta do servidor e converter o corpo da resposta (geralmente de JSON para um objeto JavaScript), o script precisa refletir essa nova informação na interface do usuário. Isso é feito através da manipulação do DOM. O script pode criar novos elementos (document.createElement()), definir seus atributos e conteúdo textual (textContent), e inseri-los na árvore do DOM (appendChild()). Alternativamente, pode modificar o conteúdo de um nó existente através da propriedade innerHTML. Esse processo atualiza a visualização da página dinamicamente, sem a necessidade de um recarregamento completo (full page reload), proporcionando uma experiência de usuário fluida e moderna, característica de uma Single-Page Application (SPA).

# O QUE É O **SERVIDOR.JS**

- O arquivo `servidor.js` é o ponto de entrada (entry point) da nossa aplicação backend. É um script JavaScript executado em um ambiente de servidor através do runtime Node.js. Ele opera no lado do servidor (server-side), o que significa que seu código é processado na máquina servidora, sendo completamente inacessível e invisível para o navegador do cliente. Ele é responsável por toda a lógica de negócio, processamento de dados e, crucialmente, pela comunicação segura com o banco de dados.

- **Para que ele serve?**

Sua principal função é atuar como um mediador seguro e inteligente entre o frontend (cliente) e as fontes de dados (banco de dados).



# PARA QUE ELE SERVE?

- Expor uma API: Criar um conjunto de endpoints (URLs) controlados que o frontend pode requisitar para obter ou manipular dados.
- Abstrair a Lógica de Negócio: O cliente não precisa saber como os dados são estruturados, buscados ou validados. Toda essa complexidade reside no backend.
- Garantir a Segurança: Ele protege as credenciais do banco de dados e implementa regras de acesso, garantindo que o cliente só possa realizar as operações permitidas.



# VAMOS USAR COMO EXEMPLO UM SISTEMA DE BUSCA DE LIVROS DE UMA BIBLIOTECA:

- Vamos seguir o caminho de um usuário que busca por "Machado de Assis" no site da biblioteca:

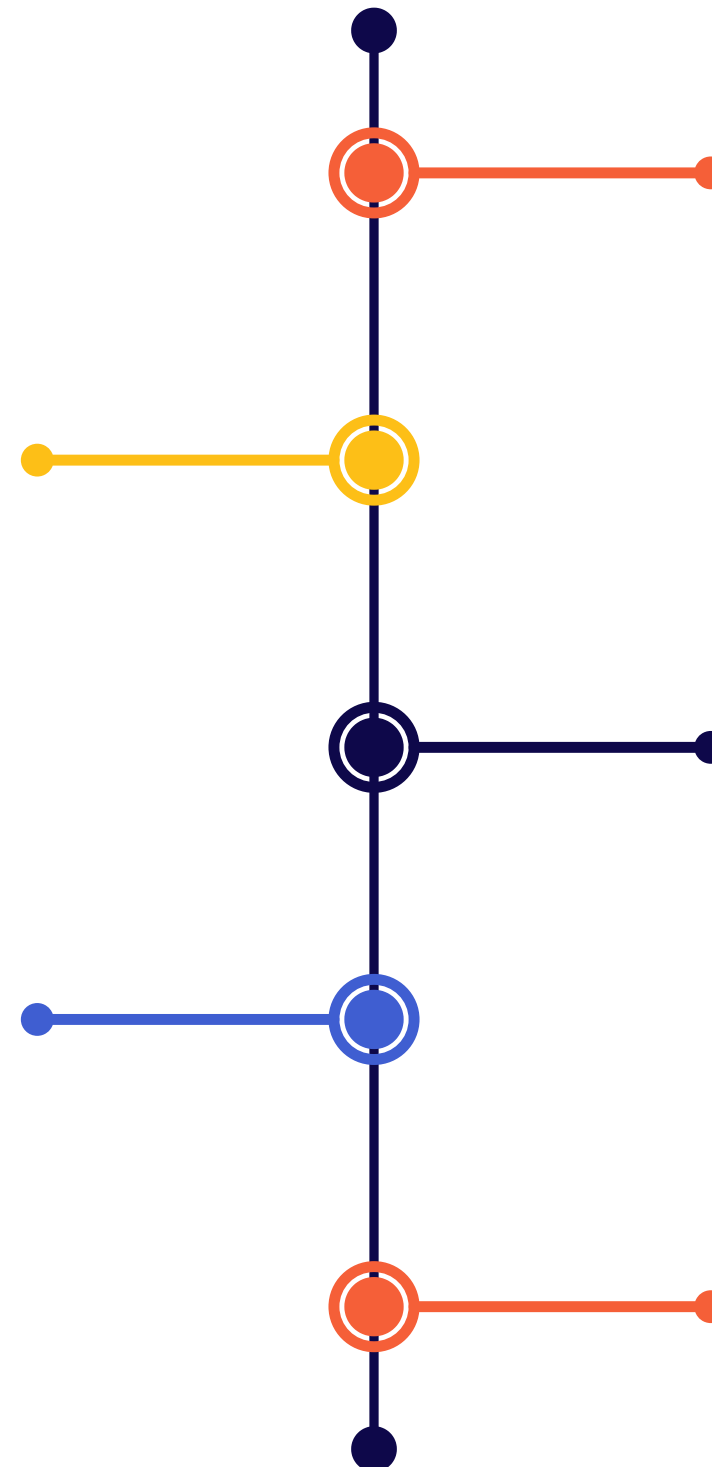
- O JavaScript no navegador cria uma requisição para a API do backend: GET /api/livros/pesquisar? termo=Machado%20de%20Assis.

- O backend "reformula" o pedido simples do usuário em uma consulta complexa que o banco de dados entende. Ele monta o comando SELECT com JOINS, WHERE, etc.

- O usuário digita "Machado de Assis" e clica em "Buscar".

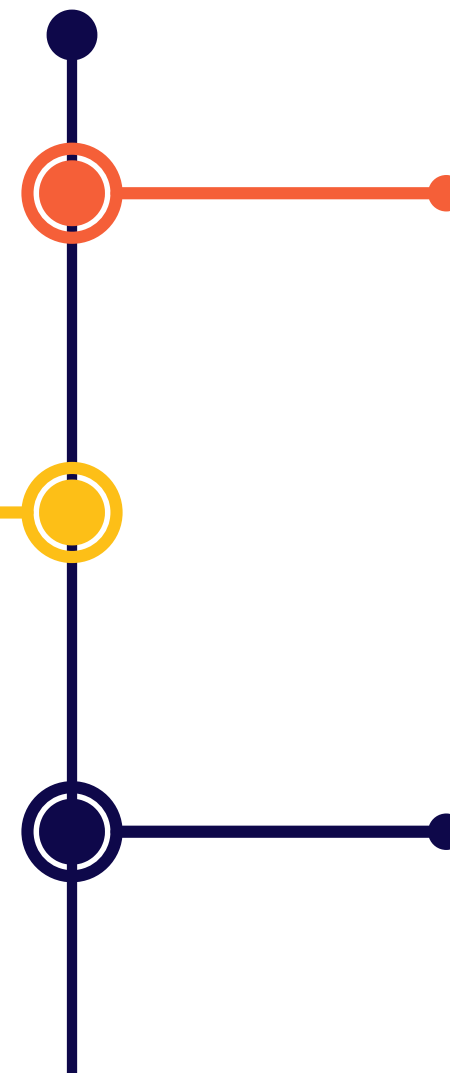
- Nosso servidor (servidor.js) está ouvindo. A rota `app.get('/api/livros/pesquisar', ...)` é acionada. Ele extrai o termo "Machado de Assis" da URL.

- Usando o driver do MySQL (mysql2), o backend estabelece uma conexão segura com o banco de dados e executa a consulta SQL.



# CONTINUANDO O PROCESSO DE BUSCA DE LIVROS DE UMA BIBLIOTECA:

- O servidor envia essa estrutura JSON de volta para o navegador que fez a requisição inicial.



- O banco de dados retorna o resultado como uma lista de linhas e colunas. O backend pega essa resposta "crua" e a "traduz" para o formato universal JSON.
- O JavaScript (script.js) recebe a lista de livros em JSON. Ele percorre essa lista e cria dinamicamente os elementos HTML (os <div> dos livros) para mostrar os resultados de forma amigável para o usuário.



# VAMOS DAR UMA OLHADA NO CÓDIGO?

[https://github.com/alinetimm/Aula\\_Integra-  
o\\_Banco\\_de\\_Dados.git](https://github.com/alinetimm/Aula_Integra-o_Banco_de_Dados.git)





# OBRIGADA!

Unisenac-Pelotas