. Explique a diferença entre mysql2 e mysql2/promise

mysql2 é a versão padrão do pacote e usa **callbacks** para lidar com operações assíncronas, como consultas ao banco de dados.

```
const mysql = require('mysql2');

const connection = mysql.createConnection({...});


connection.query('SELECT * FROM users', (err, results) => {
  if (err) throw err;
  console.log(results);
});
```

mysql2/promise é a versão **baseada em Promises**, permitindo o uso de async/await, o que torna o código mais limpo e moderno.

```
const mysql = require('mysql2/promise');

const connection = await mysql.createConnection({...});


const [rows] = await connection.execute('SELECT * FROM users');

console.log(rows);
```

2. Quais as vantagens de usar variáveis de ambiente na conexão com o banco de dados?

Usar variáveis de ambiente traz várias vantagens importantes:

1. Segurança

Evita deixar **dados sensíveis expostos no código**, como senhas, usuários e URLs.

2. Facilidade de configuração

Permite **alterar credenciais e configurações sem modificar o código-fonte**. Basta mudar os valores no .env.

3. Portabilidade

Facilita o uso do mesmo código em **ambientes diferentes** (desenvolvimento, testes, produção), mudando apenas as variáveis.

4. Manutenção mais simples

Torna o código mais **organizado e limpo**, centralizando configurações em um único lugar.

3. O que é um prepared statement e como ele ajuda na segurança da aplicação?

Um **prepared statement** é uma forma de **executar consultas SQL com parâmetros**, em que o comando SQL é **preparado primeiro** e os dados são enviados depois, separados da instrução.

Como ajuda na segurança:

Ele **evita ataques de SQL Injection**, pois os dados inseridos pelo usuário **não são interpretados como código SQL**, e sim tratados como valores seguros.

4. Por que é recomendável usar conexões em pool em aplicações Node.js?

O uso de **pool de conexões** (connection pool) é recomendado porque ele **reaproveita conexões** com o banco de dados, em vez de abrir uma nova a cada requisição.

Vantagens principais:

1. **Desempenho melhor**
   Evita o custo de abrir e fechar conexões repetidamente, o que economiza tempo e recursos.
2. **Maior escalabilidade**
   Permite que várias requisições sejam atendidas de forma eficiente, sem sobrecarregar o banco.
3. **Controle de uso**
   Você pode limitar o número máximo de conexões simultâneas, evitando que o banco de dados seja sobrecarregado.

5. Descreva o papel do módulo dotenv em projetos Node.js

O módulo dotenv serve para **carregar variáveis de ambiente** definidas em um arquivo .env para dentro do process.env no Node.js.

Função principal:

Permitir que você **mantenha dados sensíveis e configurações** (como senhas, URLs e chaves de API) **fora do código-fonte**, deixando o projeto mais seguro e fácil de configurar.

6. Qual é a estrutura básica para realizar uma consulta SELECT com o pacote mysql2?

```
const mysql = require('mysql2');

// Cria a conexão
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'sua_senha',
```

```
  database: 'nome_do_banco'
});

// Faz a consulta SELECT
connection.query('SELECT * FROM usuarios', (err, results) => {
  if (err) {
    console.error('Erro na consulta:', err);
    return;
  }
  console.log(results); // resultados da consulta
});

// Fecha a conexão
connection.end();
```

7. Três boas práticas ao conectar o Node.js com o MySQL:

1. **Usar variáveis de ambiente**
   Nunca exponha senhas ou dados sensíveis no código. Use dotenv e um arquivo .env para guardar credenciais.

2. **Utilizar pool de conexões**
   Use createPool() em vez de createConnection() para **reaproveitar conexões**, melhorar a performance e evitar sobrecarga no banco.

3. **Sempre usar prepared statements**
   Evita **SQL Injection** e garante que os dados do usuário sejam tratados com segurança.

8. Explique o conceito de transações no MySQL e como ele pode ser implementado em Node.js

O que é uma transação no MySQL?

Uma **transação** é um conjunto de operações SQL que são executadas de forma **atômica**: ou **todas são concluídas com sucesso**, ou **nenhuma é aplicada** (em caso de erro, tudo é revertido). Isso garante **consistência** e **integridade** dos dados.

9. Quais erros podem ocorrer ao conectar ao MySQL, e como tratá-los no Node.js?

Erros comuns ao conectar:

1. **Credenciais inválidas**
    - Usuário ou senha incorretos.
    - Mensagem: ER_ACCESS_DENIED_ERROR
2. **Banco de dados inexistente**
    - Nome do banco errado ou não criado.
    - Mensagem: ER_BAD_DB_ERROR
3. **Servidor MySQL indisponível**
    - MySQL não está rodando ou endereço incorreto.
    - Mensagem: ECONNREFUSED
4. **Limite de conexões excedido**
    - Muitas conexões abertas simultaneamente.
    - Mensagem: PROTOCOL_CONNECTION_LOST ou Too many connections

Como tratar os erros no Node.js:

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
 host: 'localhost',
 user: 'root',
 password: 'senha123',
 database: 'meu_banco'
});

connection.connect(err => {
 if (err) {
  console.error('Erro na conexão:', err.code); // ou err.message
  return;
 }
 console.log('Conexão bem-sucedida!');
});
```

10. Por que é importante organizar o código em módulos ao trabalhar com bancos de dados?

Organizar o código em módulos separados traz vários benefícios importantes:

1. Clareza e manutenção

Separa responsabilidades: conexão, consultas e lógica de negócio ficam em arquivos diferentes, facilitando a leitura e manutenção.

2. Reutilização de código

Você pode reaproveitar funções de acesso ao banco em vários lugares do projeto sem repetir código.

3. Facilidade para testar

Com módulos bem definidos, fica mais fácil testar partes específicas do sistema, como funções que consultam ou atualizam dados.

4. Escalabilidade

Projetos grandes exigem organização. Módulos bem estruturados tornam mais fácil adicionar novas funcionalidades sem bagunçar o código.