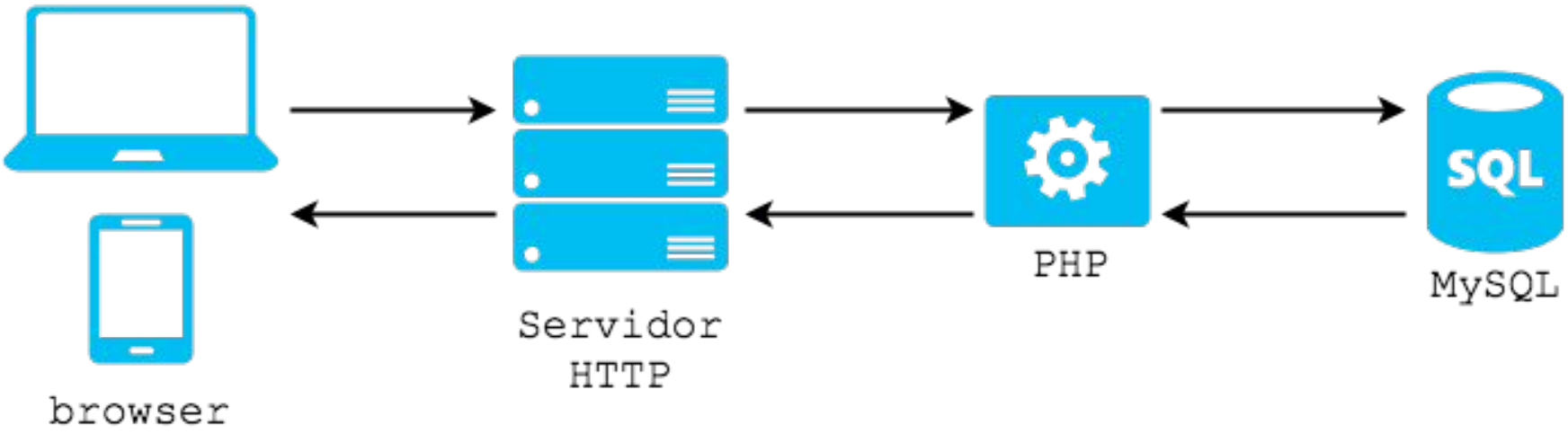
PHP

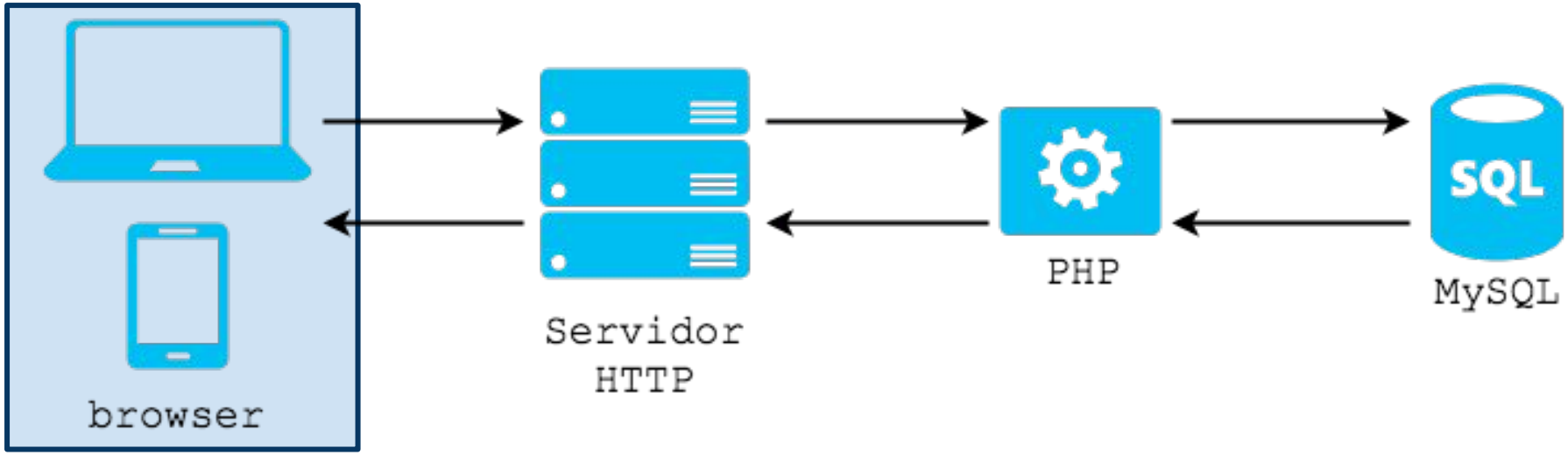
Aula 3

Regras de Negócio



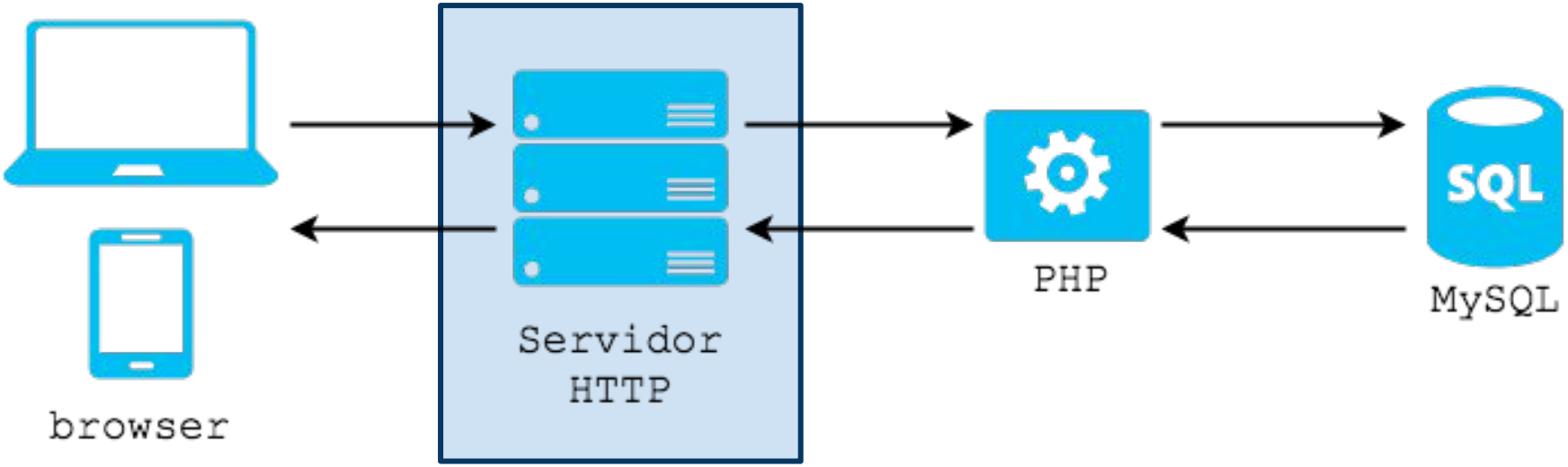
Em qual camada devemos colocar as regras de negócio?

- Browser (front-end)
- Servidor HTTP (servidor estático HTTP)
- PHP (servidor de aplicação)
- MySQL (banco de dados)



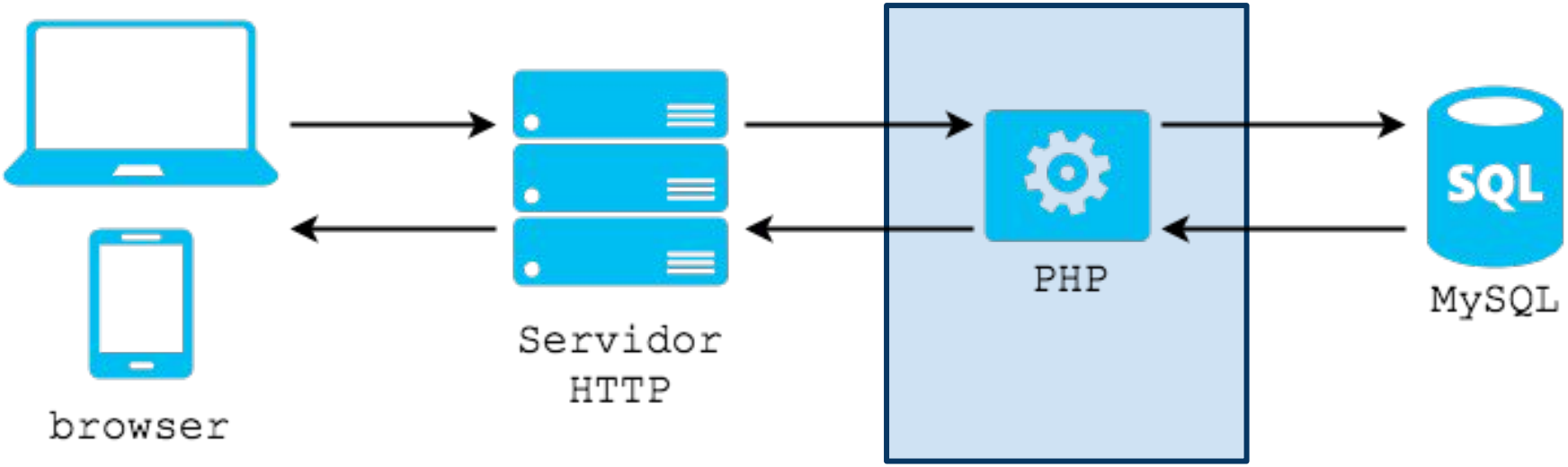
Manter regras de negócio no **browser** (com JavaScript) é uma boa prática porque caso o dado esteja inválido, não é necessário enviar uma requisição ao servidor.

NUNCA mantenha a regra de negócio **APENAS** no browser. Os usuários podem quebrá-las facilmente.



Não é possível manter regras de negócio no servidor estático HTTP. É possível, porém, manter requisitos não funcionais de redirecionamento de páginas (Not Found) ou de portas sem criptografia (porta 80) para portas com SSL/TLS (porta 443).

Observação: os scripts PHP rodam dentro do servidor Apache (ou IIS), mas isso nem sempre é verdadeiro em outras linguagens.



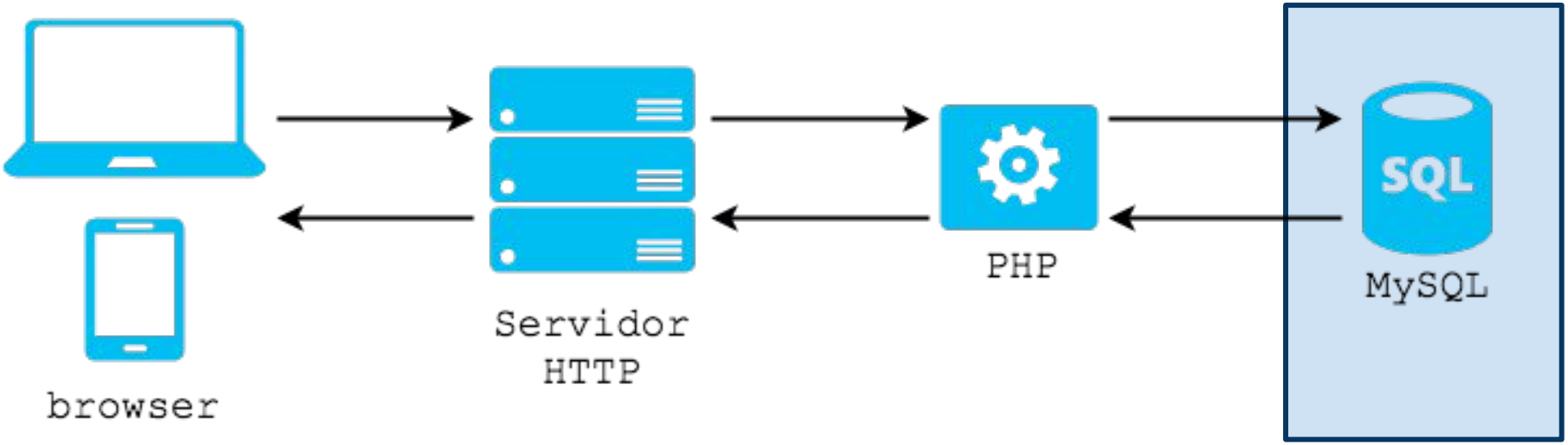
SEMPRE mantenha as regras de negócio no PHP, mesmo que seja necessário mantê-las também no front-end e no banco de dados.

No PHP, qual o melhor lugar para colocar as regras de negócio?

index.php
Apresentação

business.php
Camada de Negócios

usuario.php
Modelo de Domínio



Regras de relacionamento (chaves estrangeiras), checks, campos não nulos e únicos **DEVEM** sempre existir no banco.

Também é possível manter regras no banco com *triggers* e *procedures*. Não chega a ser uma má prática, porém tenha em mente que isso "pode" prejudicar a manutenibilidade. Às vezes por questões de consistência ou performance, é necessário utilizar essa opção.

Tabela `usuarios` (script.sql)

Veja um exemplo de regras que normalmente mantemos no banco de dados.

```
create table usuarios (
    codigo bigint auto_increment,
    email varchar(100) unique not null,
    nome varchar(100) not null,
    senha varchar(64) not null,
    codigoPerfil bigint not null,
    primary key (codigo),
    foreign key (codigoPerfil) references perfis(codigo)
);
```

Tabela `usuarios` (script.sql)

Exemplo de inserção com o campo **nome** nulo.

```
mysql> insert into usuarios (email,nome,senha,codigoPerfil) values ('pedro@site.com',
null,'df19ec5a55cfeb9597d0bd235f37e4e15b7552989f43384017852cd5b1646a95',2);
ERROR 1048 (23000): Column 'nome' cannot be null
mysql>
```

Exemplo de inserção com o valor de **email** que já existe no banco de dados.

```
mysql> insert into usuarios (email,nome,senha,codigoPerfil) values ('maria@site.com',
'Maria','df19ec5a55cfeb9597d0bd235f37e4e15b7552989f43384017852cd5b1646a95',2);
ERROR 1062 (23000): Duplicate entry 'maria@site.com' for key 'email'
mysql>
```

Regras de negócio no PHP (exe1/modelo.php)

Optando e manter as regras de negócio nas classes do modelo, podemos realizar as validações em métodos como *setEmail()* ou *setNome()*.

```php
public function setEmail($email) {
    if ( empty($email) ) {
        throw new Exception('E-mail inválido!');
    }
    if ( existeEmail($email) ) {
        throw new Exception("O e-mail <i>{$email}</i> já existe!");
    }

    $this->email = $email;
}
```

Formulário de cadastro

Recebendo dados de um `<select>` (exe2/cadastro.php)

PASSO 1: O primeiro passo é listar corretamente os perfis na tela de cadastro.

Uma boa prática é manter um método estático na própria classe de perfis. Este método deve invocar uma função do repositório.

```php
public static function obter($codigoPerfil) {
    return obterPerfil($codigoPerfil);
}
```

Métodos estáticos (ou de classe) são aqueles que podem ser invocados sem a necessidade de instanciar o objeto.

Recebendo dados de um <select> (exe2/cadastro.php)

PASSO 2: Em seguida deve-se exibir a <*select*> na tela com os dados obtidos do banco de dados.

Note que o código do objeto deve preencher o *value* do <*option*>, enquanto que a descrição deve preencher o valor exibido na tela.

```
<select id="perfil" name="perfil">
    <?php
        $perfis = Perfil::listar();
        foreach($perfis as $perfil):
    ?>
        <option value="<?= $perfil->getCodigo() ?>"><?= $perfil->getDescricao() ?></option>
    <?php endforeach; ?>
</select>
```

Recebendo dados de um `<select>` (exe2/cadastro.php)

PASSO 3: Receba o código do perfil via *$_POST*, obtenha os perfis do banco de dados e, caso o código recebido seja válido, atribua ao objeto usuário.

Atente-se que *$_POST['perfil']* é o atributo *value* do *<option>* (que neste caso, é o código do perfil).

```php
$perfil = Perfil::obter($_POST['perfil']);
$usuario->setPerfil($perfil);
```

No PHP a chamada de métodos estáticos é realizada pela sintaxe: `Classe::metodo()`

Para validação, resta-nos verificar se o perfil existe ou não.

```php
public function setPerfil($perfil) {
    if (!$perfil) {
        throw new Exception("Perfil não existe!");
    }
    $this->perfil = $perfil;
}
```

Recebendo dados de *radios* (exe3/cadastro.php)

O recebimento no back-end é igual ao do <*select*>. A diferença está na página (cadastro.php), que exige uma lógica para identificar o *radio* que está *checked*.

```php
<?php
    $perfis = Perfil::listar();
    foreach($perfis as $index => $perfil):
        $idPerfil = "perfil-" . $perfil->getCodigo();
        $indexChecked = 0;
        if ($usuario && $usuario->getPerfil()->getCodigo() == $perfil->getCodigo()) {
            $indexChecked = $index;
        }
        $checked = $indexChecked == $index ? "checked" : "";
?>
    <input type="radio" id="<?= $idPerfil ?>" name="perfil"
        value="<?= $perfil->getCodigo() ?>" <?= $checked ?> />
    <label for="<?= $idPerfil ?>"><?= $perfil->getDescricao() ?></label>
<?php endforeach; ?>
```

Validando senhas (exe4/cadastro.php)

Campos do tipo *password* são tratados como texto simples. Em formulários, é comum criarmos 2 campos desse tipos: um para a senha e o outro para a confirmação.

```php
public function setSenha($senha,$confirmarSenha) {
    if ( empty($senha) || strlen($senha) < 5 ) {
        throw new Exception("A senha deve ter no mínimo 5 letras!");
    }
    if ($senha != $confirmarSenha) {
        throw new Exception("Confirmação de senha não confere!");
    }
    $this->senha = $senha;
}
```

Caso você queira criar uma validação mais sofisticada, veja: http://www.cafewebmaster.com/check-password-strength-safety-php-and-regex

Recebendo dados de *checkbox* (exe5/cadastro.php)

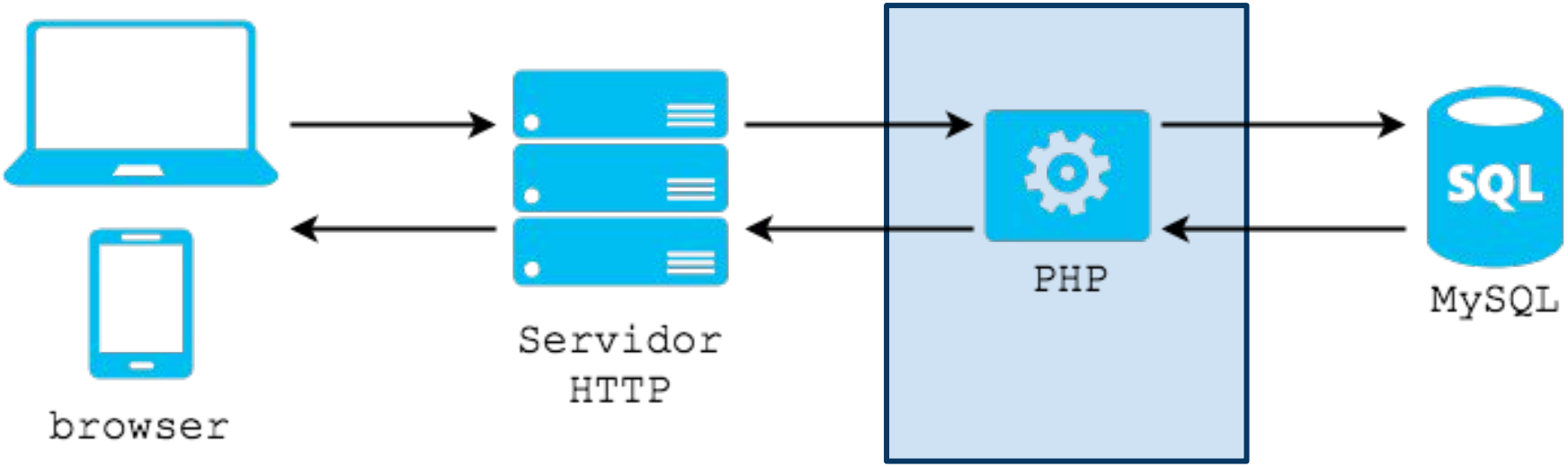
A única observação que existe em relação ao recebimento de dados do *checkbox* é que caso você não defina o atributo *value*, o PHP vai receber uma string com o valor "on", se o *checkbox* estiver marcado, ou nulo caso contrário.

```
<input type="checkbox" id="termos" name="termos" value="true" />
<label for="termos">Aceita termos?</label>
```

Como só há duas possibilidades, a validação do preenchimento do *checkbox* exige uma simples verificação.

```
if ( !isset($_POST['termos']) ) {
    throw new Exception("É necessário aceitar os termos!");
}
```

Inserindo dados após a validação (exe5/cadastro.php)



Com todas as regras validadas no PHP, a inserção não corre o risco de apresentar problemas (até porque existem as *constraints* definidas na tabela).

É uma boa prática implementar o método de salvar na classe *Usuario*.

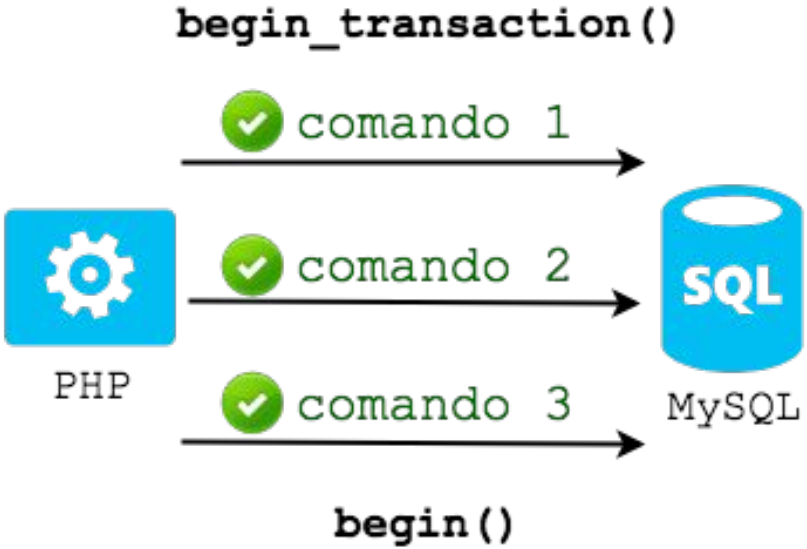
```php
public function salvar() {
    inserirUsuario($this);
}
```

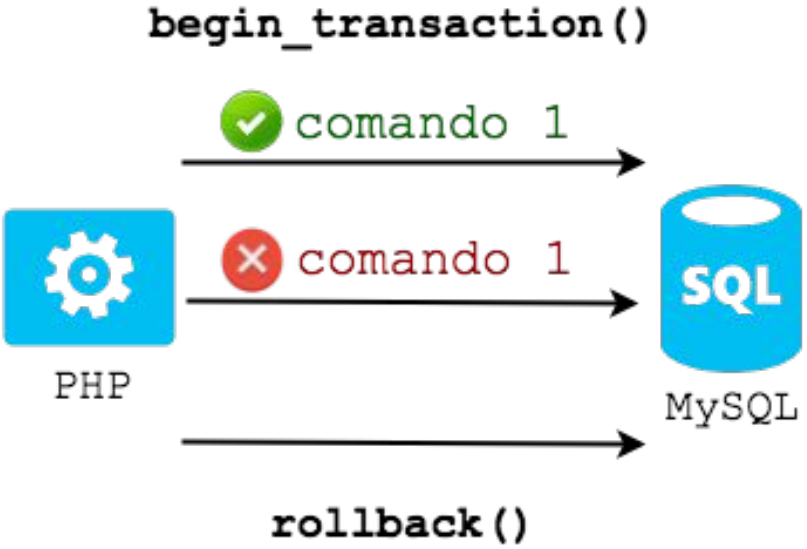
Transações

O que é uma transação no banco de dados?

Uma transação é um recurso do banco de dados pelo qual o sistema **garante** que alguns comandos sejam executados em conjunto.

Neste caso, se qualquer um deles falhar, todos os outros serão descartados.





Transações no MySQL

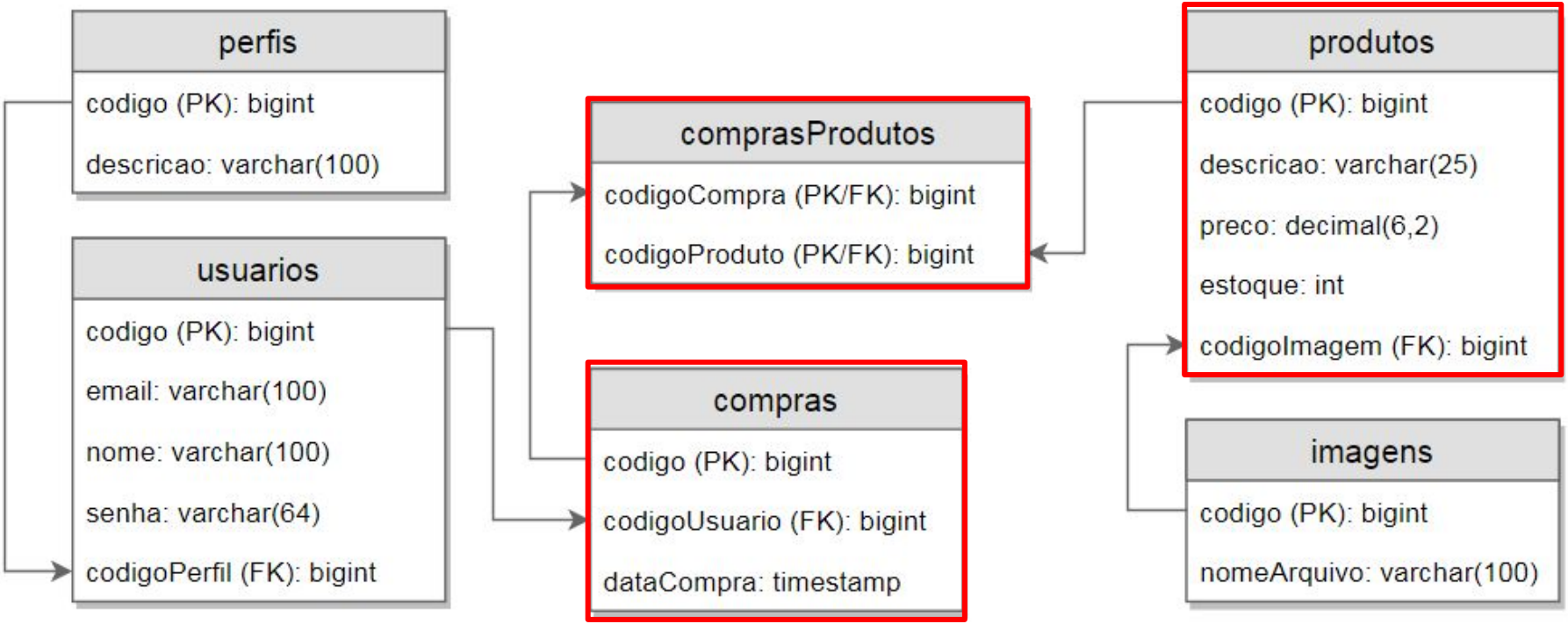
Algumas *engines* do banco de dados MySQL suportam transações. Uma *engine* é um mecanismo de armazenamento suportado pelo MySQL. Existem várias *engines*, sendo que cada uma delas oferece vantagens específicas, tais como velocidade de leitura, de escrita, transações, integridade, entre outras.

Por default, o MySQL trabalha com a *engine* InnoDB, que oferece suporte a transações.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
```

Exemplo de uso de transações (site Roupas Legais)

Imagine uma ação de compra, que além do registro na tabela ***compras***, deve identificar os produtos em ***comprasProdutos*** e reduzir o estoque na tabela ***produtos***. Este é um típico caso para trabalharmos com transações.



Trabalhando com transações (exe7.php)

PASSO 1
É uma boa prática encapsular os comandos realizados na transação dentro de uma estrutura *try/catch*. Para que isso funcione corretamente, configure o *MySQL Improved* para disparar erros.

```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

PASSO 2
Desligue o comportamento padrão de *auto commit*. Isso evita que os comandos sejam efetuados de imediato e aguardem a invocação manual do *commit()*

```
$conexao->autocommit(false);
```

Trabalhando com transações (exe7.php)

PASSO 3
Inicie a transação

```php
$conexao->begin_transaction();
```

PASSO 4
Se for necessário realizar validações de negócio, lembre-se de disparar exceções caso as regras não sejam atendidas. Neste exemplo, deve existir estoque para registrar a compra.

```php
$produto = obterProdutos($conexao);
$estoque = $produto['estoque'];
if ( $estoque == 0 ) {
    throw new Exception("Produto sem estoque!");
}
```

Trabalhando com transações (exe7.php)

PASSO 5

Realize todos os comandos de manipulação de dados (*insert, update, delete*). Note que em caso de *insert*, podemos recuperar o código inserido através da propriedade *insert_id* disponível na conexão do *MySQL Improved*.

```
$conexao->query('insert into compras (codigoUsuario) values (1)');
$codigoCompra = $conexao->insert_id;
$conexao->query('insert into comprasProdutos (codigoCompra,codigoProduto) ' .
    'values (' . $codigoCompra . ',1)');
$conexao->query('update produtos set estoque = ' . ($estoque-1) . ' where codigo = 1');
```

PASSO 6

Por fim, execute *commit()* em caso de sucesso ou *rollback()* em caso de falha. O *rollback()* preferencialmente deve ser realizado no bloco de *catch*.

```
$conexao->commit();
```

OU

```
$conexao->rollback();
```

Exercício em sala (exe7.php)

Provoque erros intencionais na lógica de manipulação dos dados no exercício 7 e veja que o *rollback()* descartará todas as alterações realizadas.

Dica: Faça consultas no MySQL para confirmar esse exercício.

Obrigado!