UNIP UNIVERSIDADE PAULISTA Instituto de Ciências Exatas e Tecnologia *Campus São José do Rio Preto*	**Curso:** Análise e Desenvolvimento de Sistemas **Disciplina:** Programação Orientada a Objetos II **Docente:** Prof. Me. William Tenório

LISTA DE EXERCÍCIOS II

1. Explique com suas palavras o papel do DataGridView em uma aplicação Windows Forms. Ele pertence a qual camada da arquitetura MVC?
2. O que significa Data Binding? Cite um exemplo prático em que o uso do binding evita código repetitivo.
3. Compare o uso de List<T> e BindingList<T> no contexto do DataGridView. Qual é a principal diferença entre as duas coleções?
4. O que acontece no DataGridView quando um item é adicionado a uma BindingList ligada a ele por um BindingSource?
5. Explique a função do BindingSource e sua importância na comunicação entre a camada de controle e a View.
6. Qual é a boa prática ao inserir um novo registro no DataGridView:
   a) Recarregar todos os dados com um novo SELECT, ou
   b) Inserir apenas o novo registro na lista vinculada?
   Justifique sua resposta.
7. Em quais cenários faz sentido recarregar toda a lista do banco de dados após uma inserção ou atualização?
8. Em uma aplicação com DataGridView, explique o conceito de interação orientada a eventos e cite exemplos de eventos comuns utilizados.
9. Descreva o fluxo completo entre as camadas quando o usuário altera um valor no DataGridView e o sistema precisa atualizar o banco de dados.
10. Analise o trecho de código abaixo:

```
var lista = clienteController.ObterTodos();
dataGridView1.DataSource = lista;
```

Após a execução, o DataGridView exibe corretamente todos os clientes. No entanto, quando um novo cliente é inserido no banco, o DataGridView não atualiza automaticamente.

a) Explique por que isso acontece.

b) Reescreva o trecho de código utilizando uma coleção que permita atualização automática dos dados na tela.

11. Considere o código a seguir:

```
BindingSource binding = new BindingSource();
binding.DataSource = clienteController.ObterTodos();

dataGridView1.DataSource = binding;

private void btnExcluir_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count > 0)
    {
        var cliente = (Cliente)dataGridView1.CurrentRow.DataBoundItem;
        clienteController.Excluir(cliente.Id);
        binding.Remove(cliente);
    }
}
```

a) Explique o papel do BindingSource neste código.
b) O que aconteceria se, ao invés de binding.Remove(cliente), fosse executado dataGridView1.Refresh()

após a exclusão?
c) Por que é boa prática **não recarregar toda a lista** do banco nesse tipo de operação?

12. Defina o que é uma **transação** no contexto de banco de dados e explique o princípio da **atomicidade**.
13. Diferencie **transações implícitas** e **explícitas** no C#. Dê um exemplo de cada uma.
14. Explique, com suas palavras, o que significam os conceitos de **commit** e **rollback**.
15. Por que é importante utilizar transações quando múltiplos DAOs (ex.: ClienteDAO e EnderecoDAO) são executados em uma mesma operação?
16. Qual é o papel da camada **Service** no controle de transações?
17. O que aconteceria se, dentro de um mesmo Service, cada DAO abrisse sua própria conexão e transação?
18. Por que é necessário passar o mesmo objeto de **conexão e transação** para todos os DAOs envolvidos?
19. Cite uma boa prática no uso de transações explícitas para evitar bloqueios e lentidão no sistema.

Considere o problema abaixo para resolver as questões 20 a 26.

Uma loja virtual precisa registrar um pedido de compra no banco de dados. O processo de cadastro de pedido envolve duas operações principais:

- Inserir o registro do pedido na tabela Pedidos (contendo data, cliente e valor total).
- Inserir vários itens desse pedido na tabela ItensPedido (cada um com produto, quantidade e valor unitário).

A estrutura simplificada é a seguinte:

```
CREATE TABLE Pedidos (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    ClienteId INTEGER,
    DataPedido TEXT,
    ValorTotal REAL
);

CREATE TABLE ItensPedido (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    PedidoId INTEGER,
    ProdutoId INTEGER,
    Quantidade INTEGER,
    ValorUnitario REAL
);
```

No código C#, o método responsável por cadastrar o pedido é o seguinte:

```
public void CadastrarPedido(Pedido pedido)
{
    using (var conn = Conexao.ObterConexao())
    {
        conn.Open();
        using (var tx = conn.BeginTransaction())
        {
            try
            {
                var pedidoId = pedidoDAO.Inserir(pedido, conn, tx);

                foreach (var item in pedido.Itens)
                {
                    item.PedidoId = pedidoId;
                    itemDAO.Inserir(item, conn, tx);
                }

                tx.Commit();
            }
            catch
            {
                tx.Rollback();
                throw;
            }
        }
```

```
    }
}
```

20. Quais **operações** fazem parte desta transação?
21. O que aconteceria se um dos ItensPedido gerasse erro durante a execução?
22. Considerando a existência de dois DAOs, pedidoDAO e itemDAO, como a criação de transações implícitas é evitada neste código e por que ambos utilizam uma mesma transação explícita?
23. Por que é importante que tanto o pedidoDAO quanto o itemDAO recebam a **mesma conexão e transação**? Isso basta para evitar o uso de transações implícitas?
24. O que ocorreria se o programador esquecesse de chamar tx.Commit() ao final?
25. Cite outro exemplo real de sistema que também deveria usar transação, explicando brevemente o motivo.
26. O método abaixo, no pedidoDAO, é responsável por inserir o pedido no banco de dados. Contudo, ainda que o servisse tenha enviado a transação e a conexão, este método continua a abrir uma nova transação implícita. Por que isso ocorre? Como corrigir?

```
public int Inserir(Pedido p, SqliteConnection conn, SqliteTransaction tx)
{
      using (var cmd = conn.CreateCommand())
      {
            cmd.CommandText = @"
                              INSERT INTO Pedidos (ClienteId, DataPedido, ValorTotal) VALUES
                              (@cliente, @data, @valor);
                              SELECT last_insert_rowid();";

            cmd.Parameters.AddWithValue("@cliente", p.Cliente.Id);
            cmd.Parameters.AddWithValue("@data",  p.dataPedido);
            cmd.Parameters.AddWithValue("@valor",  p.ValorTotal);


            var id = Convert.ToInt32(cmd.ExecuteScalar());
            return id;
      }
}
```

Considere o projeto prático sobre transações, visto em laboratório, para responder as questões de 27 a 33. Link para download do projeto: https://tinyurl.com/5x4kza27

27. No método `ClienteService.Cadastrar()`, observe as linhas:

    ```
    var id = clienteDAO.Inserir(cliente, tx);

    endereco.ClienteId = id;

    enderecoDAO.Inserir(endereco, tx);
    ```

    Explique por que é necessário passar o mesmo objeto `tx` para ambos os DAOs.

28. No `ClienteDAO`, há um trecho semelhante a:

    ```
    cmd.Transaction = tx;

    cmd.ExecuteNonQuery();
    ```

    O que aconteceria se o programador esquecesse de atribuir `cmd.Transaction = tx;`?

    O comando seria executado normalmente?

29. Onde é feita a abertura da conexão com o banco de dados? Explique por que isso não é feito nos métodos do DAO.
30. Tanto `ClienteDAO`, quanto `EnderecoDAO` possuem métodos similares `InserirTransacao`. Explique como ambos conseguem participar da mesma transação, mesmo estando em classes separadas.
31. Por que no método de inserção do `ClienteDAO` é utilizado `ExecuteScalar()` e no `EnderecoDAO` é utilizado `ExecuteNonQuery()`?

32. A tebela `Enderecos` possui uma chave estrangeira `ClienteId`, que referencia a tabela `Clientes`. O que acontece se o cliente não for realmente inserido? O banco aceitaria o endereço? Como a transação evite esse cenário?
33. Por que o método de inserção do `ClienteDAO` retorna um `int` e o método de inserção do `EnderecoDAO` é `void`? No `Service`, o retorno da chamada no método `ClienteDAO` é usado para quê?
34. O que é o **Microsoft Report Builder** e qual é seu principal objetivo?
35. Diferencie um **relatório paginado** (como os do Report Builder) de um **dashboard interativo** (como o Power BI).
36. Explique como é possível integrar um relatório criado no Report Builder a uma aplicação .NET.
37. Cite e descreva os principais componentes de um relatório RDL (Fontes de Dados, Conjuntos de Dados, Regiões de Dados, etc.).
38. O que é o **Windows Presentation Foundation (WPF)** e qual problema ele veio resolver em relação ao WinForms?
39. Qual é a principal vantagem da separação entre **XAML** (layout) e **.cs** (lógica) no WPF?
40. Como o WPF contribui para a aplicação de boas práticas de arquitetura, como o padrão MVC?