Debug - Usage Guide Content

🚀 Roteiro de Uso - Parameters ORM v1.0.2

📋 O que é o Parameters ORM?

O **Parameters ORM v1.0.2** é um sistema unificado de gerenciamento de parâmetros de configuração que permite armazenar e recuperar configurações de múltiplas fontes de dados com fallback automático.

Units Públicas:

- `Parameters.pas` - Factory class (TParameters) para criar instâncias
- `Parameters.Interfaces.pas` - Todas as interfaces públicas

✅ Funcionalidades:

- ✅ Múltiplas fontes: Database, INI Files, JSON Objects
- ✅ Fallback automático em cascata
- ✅ Suporte a 7 tipos de banco (PostgreSQL, MySQL, SQL Server, SQLite, FireBird, Access, ODBC)
- ✅ Suporte a 3 engines (UniDAC, FireDAC, Zeos)
- ✅ Thread-safe (proteção com TCriticalSection)
- ✅ Hierarquia completa: ContratoID + ProdutoID + Title + Name
- ✅ Importação/Exportação bidirecional entre fontes

1. Começando - Primeiro Uso (Sem Attributes)

💡 Sem Attributes vs Com Attributes - Qual Usar?

✅ SEM Attributes (Abordagem Direta)

O que é: Código puro, sem decorators. Você chama os métodos diretamente das interfaces.

✅ Benefícios:

- **Simplicidade:** Mais fácil de entender e debugar
- **Performance:** Zero overhead de reflexão (RTTI)
- **Controle Total:** Você decide exatamente o que fazer em cada linha
- **Compatibilidade:** Funciona em qualquer versão do Delphi/FPC
- **Curva de Aprendizado:** Rápida - ideal para iniciantes

❌ Desvantagens:

- Mais código repetitivo (boilerplate)
- Mapeamento manual classe ↔ tabela
- Sem validação automática em tempo de compilação

⚡ COM Attributes (Abordagem Declarativa)

O que é: Usa decorators como `[Table]`, `[Column]`, `[Required]` para mapear e validar automaticamente.

✅ Benefícios:

- **Código Limpo:** Menos linhas, mais declarativo
- **Auto-Documentado:** Attributes servem como documentação
- **Validação Automática:** `[Required]`, `[Email]`, `[Range]` validam antes de salvar
- **Mapeamento Automático:** Classe ↔ Tabela mapeado via reflexão
- **Integração ORM:** Perfeito para sistemas complexos com muitas entidades

❌ Desvantagens:

- **Performance:** Overhead de RTTI (reflexão em runtime)
- **Complexidade:** Curva de aprendizado maior
- **Debug:** Mais difícil de rastrear erros (código gerado dinamicamente)
- **Compatibilidade:** Requer RTTI habilitado (`{$M+}`)

👉 Recomendação:

- **Iniciantes:** Comece **SEM Attributes** (Seções 1-6)
- **Projetos Simples:** Use **SEM Attributes** (mais rápido e direto)
- **Projetos Grandes/ORM:** Use **COM Attributes** (Seção 7) para reduzir boilerplate

1.1. Usar Database (Sem Attributes)

Este é o exemplo mais simples e comum: conectar ao banco, inserir e buscar um parâmetro.

```
uses
  Parameters;  // Apenas esta unit!

var
  DB: IParametersDatabase;
  Param: TParameter;
begin
  // 1️⃣ Criar e conectar ao banco
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Port(5432)
    .Database('mydb')
    .Username('postgres')
    .Password('senha')
    .TableName('config')
    .Schema('public')
    .AutoCreateTable(True)  // ✨ Cria tabela automaticamente!
```

```
    .Connect;

  // 2️⃣ Inserir um parâmetro
  Param := TParameter.Create;
  Param.ContratoID := 1;
  Param.ProdutoID := 1;
  Param.Titulo := 'ERP';
  Param.Name := 'servidor_api';
  Param.Value := 'https://api.exemplo.com';
  Param.ValueType := pvtString;
  Param.Description := 'URL do servidor de API';

  DB.Setter(Param);  // Insere ou atualiza automaticamente!
  Param.Free;

  // 3️⃣ Buscar o parâmetro
  DB.ContratoID(1).ProdutoID(1).Title('ERP');
  Param := DB.Getter('servidor_api');
  try
    if Assigned(Param) then
      ShowMessage('Servidor: ' + Param.Value);  // Mostra: https://api.exemplo.com
  finally
    if Assigned(Param) then
      Param.Free;
  end;
end;
```

1.2. Usar Arquivo INI (Sem Attributes)

Perfeito para aplicações desktop que não querem depender de banco de dados:

```
uses
  Parameters;

var
  Ini: IParametersInifiles;
  Param: TParameter;
begin
  // 1️⃣ Criar arquivo INI
  Ini := TParameters.NewInifiles
    .FilePath('C:\Config\app.ini')
    .Section('ERP')
    .AutoCreateFile(True)  // ✨ Cria arquivo se não existir!
    .ContratoID(1)
    .ProdutoID(1);

  // 2️⃣ Inserir parâmetro
  Param := TParameter.Create;
  Param.ContratoID := 1;
  Param.ProdutoID := 1;
  Param.Titulo := 'ERP';
  Param.Name := 'servidor_api';
  Param.Value := 'https://api.exemplo.com';

  Ini.Setter(Param);
  Param.Free;

  // 3️⃣ Buscar parâmetro
  Param := Ini.Getter('servidor_api');
  try
    if Assigned(Param) then
      ShowMessage('Servidor: ' + Param.Value);
  finally
    if Assigned(Param) then
      Param.Free;
```

```
  end;

  // O arquivo app.ini foi criado com:
  // [ERP]
  // servidor_api=https://api.exemplo.com
end;
```

1.3. Usar JSON (Sem Attributes)

Ideal para integração com APIs REST e aplicações modernas:

```
uses
  Parameters;

var
  Json: IParametersJsonObject;
  Param: TParameter;
begin
  // 1️⃣ Criar JSON
  Json := TParameters.NewJsonObject
    .FilePath('C:\Config\app.json')
    .ObjectName('ERP')
    .AutoCreateFile(True)
    .ContratoID(1)
    .ProdutoID(1);

  // 2️⃣ Inserir parâmetro
  Param := TParameter.Create;
  Param.ContratoID := 1;
  Param.ProdutoID := 1;
  Param.Titulo := 'ERP';
  Param.Name := 'servidor_api';
  Param.Value := 'https://api.exemplo.com';

  Json.Setter(Param);
  Param.Free;

  // 3️⃣ Buscar parâmetro
  Param := Json.Getter('servidor_api');
  try
    if Assigned(Param) then
      ShowMessage('Servidor: ' + Param.Value);
  finally
    if Assigned(Param) then
      Param.Free;
  end;

  // O arquivo app.json foi criado com:
  // {
  //   "ERP": {
  //     "servidor_api": "https://api.exemplo.com"
  //   }
  // }
end;
```

2. Convergência - Múltiplas Fontes com Fallback

2.1. Fallback Automático (Database → INI → JSON)

O sistema tenta buscar em cada fonte automaticamente até encontrar o parâmetro:

```
uses
  Parameters;

var
  P: IParameters;
  Param: TParameter;
begin
  // 1️⃣ Criar com múltiplas fontes
  P := TParameters.New([pcfDataBase, pcfInifile, pcfJsonObject]);

  // 2️⃣ Configurar Database (prioridade 1)
  P.Database
    .Host('localhost')
    .Database('mydb')
    .TableName('config')
    .Connect;

  // 3️⃣ Configurar INI (fallback se Database falhar)
  P.Inifiles
    .FilePath('C:\Config\app.ini')
    .Section('ERP');

  // 4️⃣ Configurar JSON (último fallback)
  P.JsonObject
    .FilePath('C:\Config\app.json')
    .ObjectName('ERP');

  // 5️⃣ Buscar com fallback automático
  P.ContratoID(1).ProdutoID(1);
  P.Database.Title('ERP');
  P.Inifiles.Title('ERP');
  P.JsonObject.Title('ERP');

  Param := P.Getter('servidor_api');
  // ✨ Busca AUTOMATICAMENTE:
  // 1º Database → se não encontrar...
  // 2º INI → se não encontrar...
  // 3º JSON → se não encontrar... retorna nil

  try
    if Assigned(Param) then
      ShowMessage('Encontrado: ' + Param.Value);
  finally
    if Assigned(Param) then
      Param.Free;
  end;
end;
```

2.2. Listar de Todas as Fontes (Merge)

Combina parâmetros de todas as fontes, removendo duplicatas:

```
uses
  Parameters;

var
  P: IParameters;
  ParamList: TParameterList;
  I: Integer;
begin
  P := TParameters.New([pcfDataBase, pcfInifile, pcfJsonObject]);

  // Configurar fontes...
  P.Database.Host('localhost').Database('mydb').Connect;
```

```
  P.Inifiles.FilePath('C:\Config\app.ini');
  P.JsonObject.FilePath('C:\Config\app.json');

  // Listar TUDO (merge de todas as fontes)
  ParamList := P.List;
  // ✨ Remove duplicatas automaticamente!
  // Se mesmo parâmetro existe em Database e INI, mostra apenas 1 vez

  try
    for I := 0 to ParamList.Count - 1 do
      ShowMessage(ParamList[I].Name + ' = ' + ParamList[I].Value);
  finally
    ParamList.Free;
  end;
end;
```

3. Hierarquia Completa - Organizando Parâmetros

3.1. Entendendo a Hierarquia (ContratoID + ProdutoID + Title + Name)

A hierarquia permite organizar parâmetros por contrato, produto e seção:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Param: TParameter;
begin
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .Connect;

  // Inserir parâmetro para CONTRATO 1, PRODUTO 1, seção ERP
  Param := TParameter.Create;
  Param.ContratoID := 1;        // Empresa A
  Param.ProdutoID := 1;         // Sistema ERP
  Param.Titulo := 'ERP';        // Seção/Módulo
  Param.Name := 'servidor_api'; // Chave
  Param.Value := 'https://empresa-a-erp.com';
  DB.Setter(Param);
  Param.Free;

  // Inserir MESMO parâmetro para CONTRATO 2 (outra empresa)
  Param := TParameter.Create;
  Param.ContratoID := 2;        // Empresa B
  Param.ProdutoID := 1;         // Sistema ERP
  Param.Titulo := 'ERP';
  Param.Name := 'servidor_api'; // Mesma chave!
  Param.Value := 'https://empresa-b-erp.com';  // Valor diferente!
  DB.Setter(Param);
  Param.Free;

  // Buscar para Empresa A
  DB.ContratoID(1).ProdutoID(1).Title('ERP');
  Param := DB.Getter('servidor_api');
  ShowMessage(Param.Value);  // https://empresa-a-erp.com
  Param.Free;

  // Buscar para Empresa B
  DB.ContratoID(2).ProdutoID(1).Title('ERP');
  Param := DB.Getter('servidor_api');
  ShowMessage(Param.Value);  // https://empresa-b-erp.com
```

```
  Param.Free;

  // ✨ Mesma chave, valores diferentes por hierarquia!
end;
```

3.2. Múltiplas Seções (Títulos) no Mesmo Sistema

Organizar parâmetros por módulos/seções:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Param: TParameter;
begin
  DB := TParameters.NewDatabase.Host('localhost').Database('mydb').Connect;
  DB.ContratoID(1).ProdutoID(1);

  // Parâmetro do módulo ERP
  DB.Title('ERP');
  Param := DB.Getter('servidor_api');
  ShowMessage('ERP: ' + Param.Value);  // https://erp.exemplo.com
  Param.Free;

  // Parâmetro do módulo CRM (MESMA chave, seção diferente!)
  DB.Title('CRM');
  Param := DB.Getter('servidor_api');
  ShowMessage('CRM: ' + Param.Value);  // https://crm.exemplo.com
  Param.Free;

  // Parâmetro do módulo Financeiro
  DB.Title('Financeiro');
  Param := DB.Getter('servidor_api');
  ShowMessage('Financeiro: ' + Param.Value);  // https://financeiro.exemplo.com
  Param.Free;

  // ✨ Mesma chave em seções diferentes = valores diferentes!
end;
```

4. Importação e Exportação entre Fontes

4.1. Exportar Database → INI (Backup)

Fazer backup dos parâmetros do banco para arquivo INI:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Ini: IParametersInifiles;
begin
  // Fonte: Database
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .Connect;

  // Destino: INI
```

```
  Ini := TParameters.NewInifiles
    .FilePath('C:\Backup\config_backup.ini')
    .AutoCreateFile(True);

  // Exportar Database → INI
  Ini.ImportFromDatabase(DB);
  // ✨ Todos os parâmetros do banco foram salvos no INI!

  ShowMessage('Backup criado em: C:\Backup\config_backup.ini');
end;
```

4.2. Importar INI → Database (Restaurar)

Restaurar parâmetros do arquivo INI para o banco:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Ini: IParametersInifiles;
begin
  // Fonte: INI
  Ini := TParameters.NewInifiles
    .FilePath('C:\Backup\config_backup.ini');

  // Destino: Database
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .Connect;

  // Importar INI → Database
  DB.ImportFromInifiles(Ini);
  // ✨ Todos os parâmetros do INI foram salvos no banco!

  ShowMessage('Parâmetros restaurados no banco de dados!');
end;
```

4.3. Migrar Database → JSON

Migrar parâmetros do banco para JSON:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Json: IParametersJsonObject;
begin
  // Fonte: Database
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .Connect;

  // Destino: JSON
  Json := TParameters.NewJsonObject
    .FilePath('C:\Config\params.json')
    .AutoCreateFile(True);

  // Migrar Database → JSON
  Json.ImportFromDatabase(DB);
```

```
  // ✨ Todos os parâmetros agora estão em JSON!

  ShowMessage('Migração concluída!');
  ShowMessage(Json.ToJSONString);  // Ver JSON formatado
end;
```

5. Operações Avançadas

5.1. Listar Tabelas e Bancos Disponíveis

Descobrir quais bancos e tabelas estão disponíveis:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Databases, Tables: TStringList;
  I: Integer;
begin
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Username('postgres')
    .Password('senha')
    .Connect;

  // Listar bancos disponíveis
  Databases := DB.ListAvailableDatabases;
  try
    for I := 0 to Databases.Count - 1 do
      ShowMessage('Banco: ' + Databases[I]);
  finally
    Databases.Free;
  end;

  // Listar tabelas do banco atual
  Tables := DB.ListAvailableTables;
  try
    for I := 0 to Tables.Count - 1 do
      ShowMessage('Tabela: ' + Tables[I]);
  finally
    Tables.Free;
  end;
end;
```

5.2. Criar e Dropar Tabelas Manualmente

Gerenciar tabelas de parâmetros:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
begin
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .TableName('config_teste')
    .Connect;
```

```
  // Verificar se tabela existe
  if not DB.TableExists then
  begin
    // Criar tabela
    DB.CreateTable;
    ShowMessage('Tabela criada com estrutura padrão!');
  end;

  // Usar a tabela...

  // Remover tabela (CUIDADO!)
  if MessageDlg('Deseja remover a tabela?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
  begin
    DB.DropTable;
    ShowMessage('Tabela removida!');
  end;
end;
```

5.3. Contar e Verificar Existência

Operações úteis de contagem e verificação:

```
uses
  Parameters;

var
  DB: IParametersDatabase;
  Total: Integer;
  Existe: Boolean;
begin
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .ContratoID(1)
    .ProdutoID(1)
    .Title('ERP')
    .Connect;

  // Contar parâmetros
  Total := DB.Count;
  ShowMessage('Total de parâmetros: ' + IntToStr(Total));

  // Verificar se parâmetro existe
  Existe := DB.Exists('servidor_api');
  if Existe then
    ShowMessage('Parâmetro servidor_api existe!')
  else
    ShowMessage('Parâmetro servidor_api NÃO existe!');
end;
```

> ## ⚠️ Regras Importantes
>
> ### Hierarquia Completa (UNIQUE Constraint)
>
> A combinação `(ContratoID, ProdutoID, Title, Name)` é ÚNICA no banco. Não pode haver duplicatas.
>
> ### Métodos Getter vs Get (Deprecated)
>
> - ✅ Use `Getter()` - Recomendado

- ❌ Evite `Get()` - Deprecated (será removido)

Métodos Setter vs Update (Deprecated)

- ✅ Use `Setter()` - Insere se não existir, atualiza se existir
- ❌ Evite `Update()` - Deprecated (será removido)

Liberar Memória

Sempre libere objetos `TParameter` e `TParameterList` após uso:

```
Param := DB.Getter('chave');
try
  // Usar Param...
finally
  if Assigned(Param) then
    Param.Free;
end;
```

6. Exemplos Práticos Completos

6.1. Sistema Multi-Empresa com Fallback

Sistema que busca configuração no banco e, se falhar, usa arquivo INI local:

```
uses
  Parameters;

function GetConfiguracao(AContratoID, AProdutoID: Integer; AChave: string): string;
var
  P: IParameters;
  Param: TParameter;
begin
  Result := '';

  // Criar com fallback Database → INI
  P := TParameters.New([pcfDataBase, pcfInifile]);

  try
    // Tentar conectar ao banco (pode falhar)
    P.Database
      .Host('servidor-remoto.com')
      .Database('config_global')
      .Username('user')
      .Password('pass')
      .TableName('parametros')
      .Connect;
  except
    // Se banco falhar, INI será usado automaticamente
  end;

  // Configurar INI (fallback local)
  P.Inifiles
    .FilePath(ExtractFilePath(ParamStr(0)) + 'config.ini')
    .Section('Sistema');

  // Buscar com hierarquia
  P.ContratoID(AContratoID).ProdutoID(AProdutoID);
  P.Database.Title('Sistema');
```

```
  P.Inifiles.Title('Sistema');

  Param := P.Getter(AChave);
  try
    if Assigned(Param) then
      Result := Param.Value;
  finally
    if Assigned(Param) then
      Param.Free;
  end;
end;

// Usar:
var
  ServidorAPI: string;
begin
  ServidorAPI := GetConfiguracao(1, 1, 'servidor_api');
  ShowMessage('API: ' + ServidorAPI);
end;
```

6.2. Configuração Distribuída (Database + JSON Local)

Configurações globais no banco + configurações locais em JSON:

```
uses
  Parameters;

var
  P: IParameters;
  Param: TParameter;
begin
  P := TParameters.New([pcfDataBase, pcfJsonObject]);

  // Configurações GLOBAIS (banco remoto)
  P.Database
    .Host('config-server.com')
    .Database('global_config')
    .TableName('parametros')
    .Connect;

  // Configurações LOCAIS (JSON no computador)
  P.JsonObject
    .FilePath(ExtractFilePath(ParamStr(0)) + 'local_config.json')
    .ObjectName('Local');

  // Configurar hierarquia
  P.ContratoID(1).ProdutoID(1);
  P.Database.Title('Global');
  P.JsonObject.Title('Local');

  // Buscar GLOBAL primeiro, se não achar, busca LOCAL
  Param := P.Getter('timeout_api');
  try
    if Assigned(Param) then
      ShowMessage('Timeout: ' + Param.Value);
  finally
    if Assigned(Param) then
      Param.Free;
  end;
end;
```

💡 Dicas e Boas Práticas

- **Use AutoCreateTable(True)** para desenvolvimento - cria a estrutura automaticamente
- **Sempre configure hierarquia completa** - ContratoID, ProdutoID e Title antes de buscar
- **Use Setter() em vez de Insert()** - ele insere ou atualiza automaticamente
- **Libere memória** - TParameter e TParameterList precisam ser liberados manualmente
- **Use fallback para contingência** - Database → INI garante que sempre terá configuração
- **Organize por Title** - Use títulos como "ERP", "CRM", "Financeiro" para modularizar

7. Uso Avançado COM Attributes (Mapeamento Declarativo)

O que são Attributes?

Attributes (ou decorators) permitem mapear classes Pascal para estruturas de dados usando anotações declarativas como `[Table]`, `[Column]`, etc.

Vantagens:

- Código mais limpo e declarativo
- Mapeamento automático classe ↔ tabela
- Validação em tempo de compilação
- Integração com ORM e reflexão (RTTI)

Units Attributes disponíveis:

- `Parameters.Attributes.pas` - Attributes principais
- `Parameters.Attributes.Interfaces.pas` - Interfaces de Attributes
- `Parameters.Attributes.Types.pas` - Tipos de Attributes
- `Parameters.Attributes.Consts.pas` - Constantes de Attributes
- `Parameters.Attributes.Exceptions.pas` - Exceções de Attributes

7.1. Classe Mapeada com [Table] Attribute

Usar Attributes para mapear uma classe Pascal para uma tabela de parâmetros:

```
uses
  Parameters,
  Parameters.Attributes;  // ✨ Unit de Attributes!

{$M+}  // Habilitar RTTI
type
  [Table('config')]         // ✨ Mapeia para tabela 'config'
  [Schema('public')]        // ✨ Schema do banco
  TConfiguracao = class
  private
    FID: Integer;
    FName: string;
    FValue: string;
    FDescription: string;
  published
```

```
    [PrimaryKey]            // ✨ Chave primária
    [AutoIncrement]         // ✨ Auto incremento
    property ID: Integer read FID write FID;

    [Column('name')]        // ✨ Nome da coluna no banco
    [Required]              // ✨ Campo obrigatório
    [MaxLength(100)]        // ✨ Validação de tamanho
    property Name: string read FName write FName;

    [Column('value')]
    property Value: string read FValue write FValue;

    [Column('description')]
    property Description: string read FDescription write FDescription;
  end;

var
  DB: IParametersDatabase;
  Config: TConfiguracao;
  Param: TParameter;
begin
  // 1️⃣ Conectar ao banco
  DB := TParameters.NewDatabase
    .Host('localhost')
    .Database('mydb')
    .Connect;

  // 2️⃣ Criar instância da classe mapeada
  Config := TConfiguracao.Create;
  try
    Config.Name := 'servidor_api';
    Config.Value := 'https://api.exemplo.com';
    Config.Description := 'URL do servidor de API';

    // 3️⃣ Converter classe para TParameter usando RTTI
    Param := TParameter.FromClass(Config);
    try
      // 4️⃣ Inserir usando o parâmetro
      DB.Setter(Param);
      ShowMessage('Configuração inserida com Attributes!');
    finally
      Param.Free;
    end;
  finally
    Config.Free;
  end;
end;
```

7.2. Attributes de Validação

Usar Attributes para validar dados antes de inserir:

```
uses
  Parameters,
  Parameters.Attributes;

{$M+}
type
  [Table('config')]
  TConfiguracao = class
  private
    FEmail: string;
    FPorta: Integer;
    FAtivo: Boolean;
  published
```

```
    [Column('email')]
    [Required]              // ✨ Não pode ser vazio
    [Email]                 // ✨ Valida formato de email
    property Email: string read FEmail write FEmail;

    [Column('porta')]
    [Required]
    [Range(1, 65535)]       // ✨ Porta entre 1 e 65535
    property Porta: Integer read FPorta write FPorta;

    [Column('ativo')]
    property Ativo: Boolean read FAtivo write FAtivo;
  end;

var
  Config: TConfiguracao;
  Param: TParameter;
begin
  Config := TConfiguracao.Create;
  try
    Config.Email := 'admin@exemplo.com';  // ✅ Email válido
    Config.Porta := 8080;                 // ✅ Porta válida
    Config.Ativo := True;

    // Converter e validar automaticamente
    Param := TParameter.FromClass(Config);
    try
      // ✨ Se validação falhar, lança exceção!
      ShowMessage('Validação passou!');
    finally
      Param.Free;
    end;
  finally
    Config.Free;
  end;
end;
```

7.3. Attributes de Comportamento

Controlar comportamento de campos com Attributes:

```
uses
  Parameters,
  Parameters.Attributes;

{$M+}
type
  [Table('config')]
  TConfiguracao = class
  private
    FID: Integer;
    FSenha: string;
    FDataCriacao: TDateTime;
    FVersao: string;
  published
    [PrimaryKey]
    [AutoIncrement]
    property ID: Integer read FID write FID;

    [Column('senha')]
    [Encrypted]             // ✨ Campo será criptografado
    property Senha: string read FSenha write FSenha;

    [Column('data_criacao')]
    [Timestamp]             // ✨ Timestamp automático
```

```
    [Default('NOW()')]      // ✨ Valor padrão no banco
    property DataCriacao: TDateTime read FDataCriacao write FDataCriacao;

    [Ignore]                // ✨ NÃO será salvo no banco
    property Versao: string read FVersao write FVersao;
  end;
```

7.4. Attributes de Auditoria

Rastreamento automático de criação/modificação:

```
uses
  Parameters,
  Parameters.Attributes;

{$M+}
type
  [Table('config')]
  TConfiguracao = class
  private
    FID: Integer;
    FName: string;
    FCreatedAt: TDateTime;
    FUpdatedAt: TDateTime;
    FCreatedBy: Integer;
    FUpdatedBy: Integer;
    FDeletedAt: TDateTime;
  published
    [PrimaryKey]
    [AutoIncrement]
    property ID: Integer read FID write FID;

    [Column('name')]
    property Name: string read FName write FName;

    [Column('created_at')]
    [Timestamp]             // ✨ Preenchido ao criar
    property CreatedAt: TDateTime read FCreatedAt write FCreatedAt;

    [Column('updated_at')]
    [Timestamp]             // ✨ Atualizado a cada modificação
    property UpdatedAt: TDateTime read FUpdatedAt write FUpdatedAt;

    [Column('created_by')]
    [UserStamp]             // ✨ ID do usuário que criou
    property CreatedBy: Integer read FCreatedBy write FCreatedBy;

    [Column('updated_by')]
    [UserStamp]             // ✨ ID do usuário que atualizou
    property UpdatedBy: Integer read FUpdatedBy write FUpdatedBy;

    [Column('deleted_at')]
    [SoftDelete]            // ✨ Soft delete (não remove, marca como deletado)
    property DeletedAt: TDateTime read FDeletedAt write FDeletedAt;
  end;
```

7.5. Ler Attributes em Runtime (RTTI)

Acessar Attributes de uma classe em tempo de execução:

```
uses
  Parameters,
```

```
  Parameters.Attributes,
  TypInfo, Rtti;

var
  Context: TRttiContext;
  RttiType: TRttiType;
  RttiProp: TRttiProperty;
  Attr: TCustomAttribute;
  TableAttr: TableAttribute;
  ColumnAttr: ColumnAttribute;
begin
  Context := TRttiContext.Create;
  try
    // Obter informações RTTI da classe
    RttiType := Context.GetType(TConfiguracao);

    // Ler [Table] attribute da classe
    for Attr in RttiType.GetAttributes do
    begin
      if Attr is TableAttribute then
      begin
        TableAttr := TableAttribute(Attr);
        ShowMessage('Tabela: ' + TableAttr.TableName);
        ShowMessage('Schema: ' + TableAttr.SchemaName);
      end;
    end;

    // Ler [Column] attributes das propriedades
    for RttiProp in RttiType.GetProperties do
    begin
      for Attr in RttiProp.GetAttributes do
      begin
        if Attr is ColumnAttribute then
        begin
          ColumnAttr := ColumnAttribute(Attr);
          ShowMessage('Propriedade: ' + RttiProp.Name +
                      ' → Coluna: ' + ColumnAttr.ColumnName);
        end;
      end;
    end;
  finally
    Context.Free;
  end;
end;
```

⚠️ Quando Usar Attributes?

✅ Use Attributes Quando:

- Você quer mapeamento declarativo classe ↔ tabela
- Precisa de validação em tempo de compilação
- Está construindo um ORM ou sistema baseado em reflexão
- Quer código mais limpo e auto-documentado

❌ NÃO Use Attributes Quando:

- Você está apenas lendo/escrevendo parâmetros simples
- Performance é crítica (RTTI tem overhead)
- Você prefere controle explícito do código
- Está começando a aprender o sistema (comece sem Attributes!)

> 💡 **Dica:**
>
> O Parameters ORM funciona **perfeitamente sem Attributes**! Attributes são um recurso **opcional** para casos avançados.