

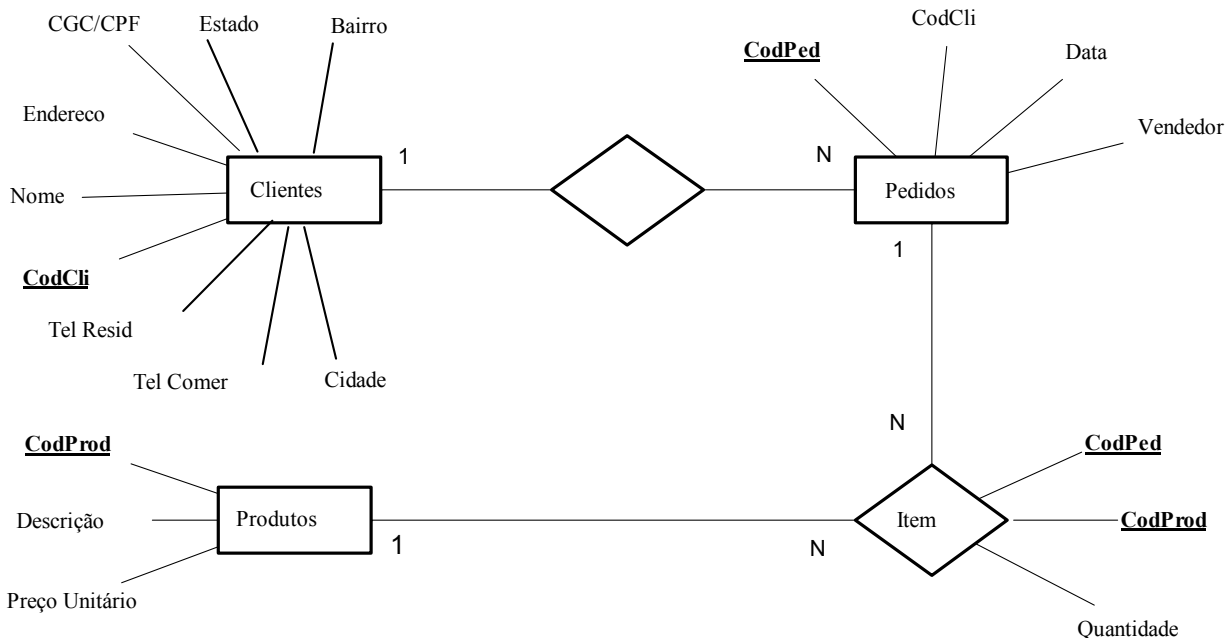
# Bancos de Dados

## Conceitos Importantes

O gerenciamento de bancos de dados é essencial para o desenvolvimento comercial, e para criar um banco de dados eficiente é necessário o conhecimento prévio de modelagem de bancos de dados relacionais. Conceitos como banco de dados, tabelas, campos, registros, índices, chaves, relacionamentos, normalização, dentre outros são pré-requisitos básicos para o desenvolvimento desse conteúdo.

## Modelo de Dados

É essencial planejar o banco de dados antes de implementar. Um dos métodos que você pode utilizar é o DER, como no exemplo não normalizado mostrado logo abaixo.



## Borland Database Engine

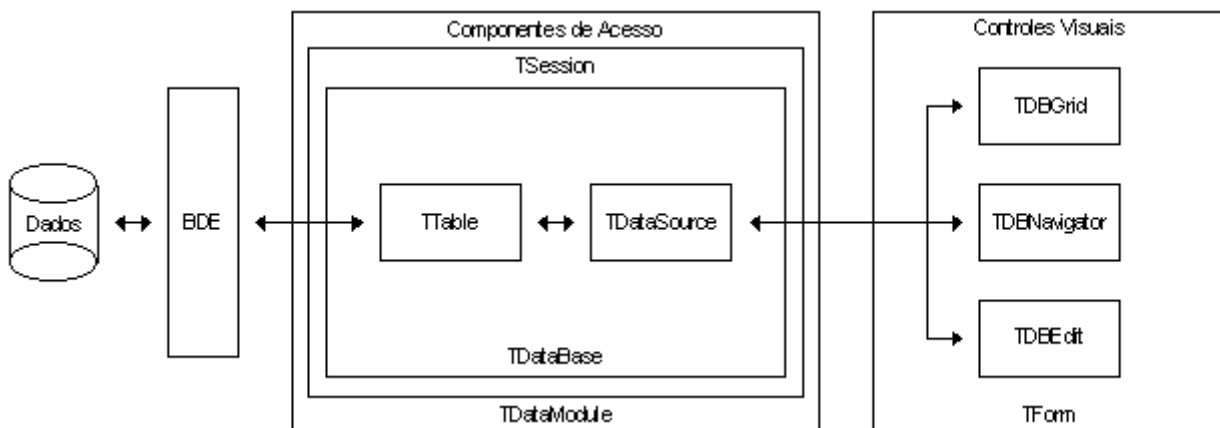
A BDE fornece a capacidade de acesso padronizado a banco de dados para Delphi, C++ Builder e outros ambientes de programação da Borland, oferecendo um grande conjunto de funções para auxiliar no desenvolvimento de aplicações Desktop e Cliente/Servidor.

Os controladores da BDE podem ser usados para acessar bases de dados dBase, Paradox, Access, FoxPro, Interbase, Oracle, Sybase e MS-SQL Server, DB2, Informix, além de um controlador de acesso a arquivos texto. Você também pode utilizar fontes de dados ODBC, podendo acessar qualquer base de dados compatível.

As funções que compõem uma API da BDE são usadas internamente pelos componentes de acesso a dados do Delphi e muito raramente você teria que usá-las diretamente, mas isso é totalmente possível. A referência completa das funções da BDE, com exemplos em Delphi, está no BDE API Help na pasta do Delphi no Menu Iniciar.

## Arquitetura de Acesso

O acesso e manipulação de um banco de dados por um programa Delphi é realizado como mostrado abaixo, note que a aplicação não acessa os dados diretamente, mas usa sempre a BDE.



Assim, para uma aplicação de bancos de dados funcionar, é preciso que a BDE esteja instalada na máquina, não bastando apenas o arquivo executável.

## Criação do Banco de Dados

Para criar um banco de dados novo, normalmente, é necessário dispor de alguma ferramenta do próprio banco de dados, como o Access, mas se a base de dados for Paradox, ou dBase, você pode usar o Database Desktop, um utilitário que vem com o Delphi e permite a criação desses tipos de bancos de dados.

### Database Desktop

Fornecer uma interface simples e completa para configuração, definição e manipulação de tabelas de bancos de dados Paradox e dBase. Além disso na Opção *Tools/Alias Manager* você pode configurar seu banco de dados, como será lembrado logo adiante.

### Tabelas Paradox

Para criar tabelas Paradox, siga os passos abaixo. Você deve salvar as tabelas de um mesmo banco de dados na mesma pasta, pois o Paradox trata a pasta onde estão as tabelas como sendo o banco de dados.

- Clique em *File/New/Table*
- Escolha o tipo da nova tabela, *Paradox 7*
- Aparece uma janela para que você defina a estrutura de campos, índices e demais opções necessárias na criação da tabela
- Em *Field Name*, você escolhe o nome do campo, com até 25 caracteres
- Em *Type*, o Tipo do campo, com a barra de espaço ou o botão direito do mouse você pode escolher o tipo a partir de uma lista
- *Size* é o tamanho do campo, usado somente em alguns tipos de campos
- *Key* especifica os campos que farão parte da chave primária, que não pode se repetir e deve ser composta pelos primeiros campos da tabela

### Table Properties

Em Table Properties você define os vários aspectos de configuração da tabela. Muitas dessas opções podem ser implementadas no Delphi e vários programadores preferem não usá-las no Database Desktop.

Opção	Descrição
Validity Checks	Validações para os campos, como obrigatoriedade, valor mínimo e máximo
Table Lookup	Indica que o valor atribuído a um determinado campo tem que estar gravado em outra tabela
Secondary Indexes	Cria índices secundários
Referential Integrity	Cria integridade referencial, geralmente utilizada em relacionamentos de 1 para N.
Password Security	Permite a criação de senhas, protegendo a tabela de acesso não autorizado
Table Language	Especificar o driver de língua utilizado pela tabela, geralmente é o <i>Pdox ANSI Intl850</i>
Dependent Tables	Mostra todas as tabelas dependentes através da integridade referencial

### Tipos de Campos

Os principais tipos de campos são mostrados abaixo, mas existem outros além desses. Os tamanhos marcados com asterisco indicam que o campo pode guardar tamanhos maiores que os informados, o que ultrapassar o tamanho será guardado em um arquivo externo com a extensão MB.

Tipo	Descrição	Faixa	Tamanho
A	Alfanumérico		1-255
N	Numérico	$\pm 10^{308}$	
\$	Monetário		
S	Short Integer	$\pm 32767$	
I	Long Integer	$\pm 2147483648$	
D	Data		
T	Hora		
@	Data e Hora de modificação		
M	Memo		1-240*
G	Gráfico		1-240*
L	Lógico	True/False	
+	Autoincremental	1-2147483648	

### Configuração

Para configurar o acesso a um banco de dados, você tem várias opções, criar um Alias, usar o componente TDatabase ou os dois juntos.

### Aliases

Um Alias é um nome lógico, um atalho para um banco de dados. Todo o trabalho do Delphi com um banco de dados pode ser feito baseado no Alias, de forma que para mudar de banco de dados, só é necessário mudar o Alias. Para criar um Alias você pode usar *Database Explorer*, o *BDE Administrator* ou o próprio *Database Desktop*.

### Database Explorer

Pode aparecer com os nomes *Database Explorer* ou *SQL Explorer*. Nele você pode manipular os Aliases, navegar pelas estruturas dos bancos de dados, alterar os dados das tabelas e executar comandos SQL.

Para criar um Alias selecione o item *Databases*, clique em *Object/New*, escolha o tipo do banco de dados, ou *Standard* para dBase, Paradox e arquivos texto, depois digite um nome do Alias, esse nome será usado pelo Delphi quando você quiser acessar o banco de dados, finalmente defina as propriedades do banco de dados na seção *Definition*, cada banco de dados terá suas próprias definições.

### BDE Administrator

Com o BDE Administrator você pode alterar a configuração da BDE, por exemplo em *Configuration/System/Init* você tem a propriedade *Local Share* que deve ser setada para True, quando você quiser que a base de dados seja compartilhada em uma rede. Além disso, você pode criar Aliases, como no *Database Explorer*.

### TDatabase

Esse componente permite a manipulação de um banco de dados, através de um Alias da BDE ou a criação de um Alias local, somente visível dentro da aplicação, esse componente também permite o gerenciamento de transações, garantindo uma integridade maior no projeto. Por essas e outras razões o uso do componente Database é altamente recomendado como opção para criação de Aliases.

Propriedades	Descrição
AliasName	Nome do Alias do banco de dados, usado quando você criar um Alias da BDE
Connected	Define se a conexão com o banco de dados está ativa
DatabaseName	Nome do Alias local a ser usado pelos outros componentes do Delphi
DataSetCount	Número de DataSets (Tabelas) abertos no banco de dados
DataSets	Lista com os DataSets abertos
DriverName	Driver usado para criar um Alias local, automaticamente cancela a propriedade AliasName
InTransaction	Define se o Database está em transação
KeepConnection	Define se a conexão com o banco de dados será mantida, mesmo sem DataSets abertos
LoginPrompt	Define se será mostrado o quadro de login padrão da BDE
Params	Parâmetros do banco de dados, com itens semelhantes à seção Definition do Database Explorer
TransIsolation	Nível de isolamento da transação, define como uma transação irá enxergar outra
Métodos	Descrição
Close	Encerra a conexão com o banco de dados, todos os DataSets serão fechados
CloseDataSets	Fecha todos os DataSets abertos, mas a conexão não é encerrada
Commit	Grava alterações feitas durante a transação
Open	Abre a conexão com o banco de dados
Rollback	Anula todas as alterações feitas durante a transação
StartTransaction	Inicia uma transação
Eventos	Descrição
OnLogin	Evento usado quando você quiser escrever seu próprio método de conexão com o banco de dados

Para acessar uma base de dados Access, você poderia usar os valores mostrados na descrição textual a seguir.

```
AliasName = 'Northwind'
DatabaseName = 'Dados'
LoginPrompt = False
KeepConnection = True
Params.Strings = (
  'DATABASE NAME=C:\Meus Documentos\NorthWind.mdb'
  'USER NAME=paulo'
  'OPEN MODE=READ/WRITE'
  'LANGDRIVER=intl850'
  'PASSWORD=elvis')
```

Para ajudar a preencher os parâmetros de um Database, clique duas vezes sobre o componente e clique em Defaults, todos os parâmetros defaults serão apresentados.

Para acessar uma base Paradox, use as propriedades abaixo, note que para o Paradox, a única informação realmente significativa é o Path, a pasta onde estão as tabelas.

```
AliasName = 'DBDEMOS'
DatabaseName = 'Dados'
LoginPrompt = False
KeepConnection = True
Params.Strings = (
  'PATH=d:\Borland\Delphi 3\Demos\Data'
  'ENABLE BCD=FALSE'
  'DEFAULT DRIVER=PARADOX')
```

Após a criação do Alias da BDE ou do Alias local, usando o componente TDatabase, o banco de dados está configurado e pronto para ser usado.

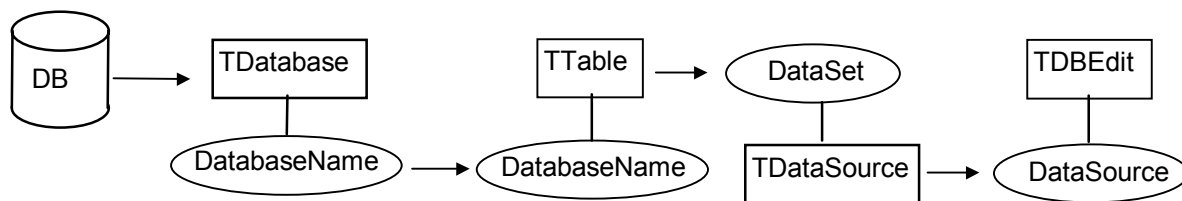
### Database Form Wizard

Após a configuração do banco de dados, a maneira mais rápida, de se fazer uma janela de manutenção de dados é através do *Form Wizard* no menu *Database*. Ao chegar no Wizard são feitas uma série de perguntas que podem resultar em uma janela simples ou Mestre/Detalhe. O acesso ao banco de dados pode ser feito através de componentes TTable ou através de SQL, com o componente TQuery, usaremos o componente TTable. Todos os campos selecionados aparecem na janela permitindo entrada de dados através de componentes do tipo TDBEdit. Cada DBEdit recebe um Label baseado no nome do campo na tabela selecionada. Na Janela é incluído também

um componente para permitir a navegação e a manutenção dos dados, um DBNavigator. O componente utilizado para fazer a ligação entre os componentes visuais e o TTable é um TDataSource. Geralmente os componentes TTable e TDataSource são inseridos em DataModules, que são a base para a criação de classes de dados. Sempre Após usar o Wizard, lembre-se de mudar os nomes dos componentes, para que fiquem mais claros.

### Form Passo a Passo

O diagrama abaixo mostra como o Wizard fez a ligação entre os componentes, onde os quadrados são componentes e as elipses, propriedades.



Para concluir, acompanhe abaixo os passos realizados pelo Wizard e tente você mesmo criar seu próprio Form.

- Inclua um novo DataModule
- Adicione ao DataModule um Table e um DataSource
- No Table Coloque em DatabaseName o nome do Alias criado pela propriedade DatabaseName do Database e em TableName, o nome da tabela
- No evento OnCreate do DataModule, chame o método Open do componente Table
- No DataSource coloque em DataSet o nome do componente TTable
- No Form, para definir a interface com o usuário, use os componentes de controle de dados que estão na página DataControls, basicamente DBEdit e DBNavigator
- Para poder acessar os dados, coloque a Unit onde está o DataModule no uses da Unit do Form
- Em todos os componentes DataControls, escolha na propriedade DataSource, o componente DataSource criado no DataModule
- Em alguns controles, como no DBEdit, deve ser especificado também o campo da tabela, na propriedade DataField

Seguindo esses passos, o Form estará pronto para usar. Mais adiante, veremos uma forma mais rápida de se criar um Form de manutenção, mas o mais importante é compreender os passos mostrados acima, com todos os componentes e propriedades envolvidas. Vamos detalhar agora cada um dos componentes envolvidos nesse processo, para compreendermos melhor o que está acontecendo.

### TDataModule

Um DataModule é como se fosse um Form invisível, onde iremos inserir os componentes de acesso a dados, como o Table e o Datasource. Por serem também classes, os DataModules permitem a fácil implementação de modelos de objetos, permitindo herança, criação de métodos, dentre outros aspectos. Para inserir um DataModule em um projeto, escolha New DataModule do menu File. Os DataModules não gastam recursos do sistema, servem apenas para conter os componentes de acesso a dados e criar, assim, uma classe persistente.

### TTable

Componente usado para acessar uma tabela em um banco de dados. Esse componente é o mais importante quando acessamos bases de dados Desktop. Muitas dos itens mostrados abaixo estão definidos na classe TDataSet, ancestral do TTable.

Propriedades	Descrição
Active	Define se a tabela esta aberta ou fechada
BOF	Informa se está no início da tabela
CanModify	Define se a aplicação pode inserir, deletar ou alterar registros
DatabaseName	Nome do banco de dados onde está a tabela, deve ser escolhido um Alias, que pode ser local
EOF	Informa se está no fim da tabela
Exclusive	Define se a tabela pode ser compartilhada por outro usuário
FieldCount	Número de campos da tabela
FieldDefs	Lista com a Definição dos campos da tabela
Fields	Lista de objetos do tipo TField, que representam os campos da tabela
Filter	String com uma condição de filtragem
Filtered	Define se a tabela é filtrada
IndexFieldNames	Nome dos campo de índice, usados para ordenar os registros da tabela
IndexName	Nome do índice atual, vazia quando o índice for a chave primária
IndexDefs	Lista com a definição dos índices
MasterFields	Campos usados no relacionamento com a tabela mestre
MasterSource	DataSource da tabela mestre em uma relação Mestre/Detalle
Modified	Define se o registro atual foi modificado
ReadOnly	Define se a tabela é somente para leitura

RecNo	Número do registro atual
RecordCount	Número de registros
State	Estado da tabela
TableName	Nome da tabela
TableType	Tipo da tabela
<b>Método</b>	<b>Descrição</b>
AddIndex	Cria um novo índice, a tabela deve ser exclusiva
Append	Entra em modo de inserção e, ao gravar, o registro será colocado no fim do arquivo
AppendRecord	Insere um registro no final do arquivo através de código
Cancel	Cancela as alterações feitas no registro atual
Close	Fecha a tabela
CreateTable	Cria uma tabela, depende de FieldDefs e IndexDefs
Delete	Exclui o registro corrente
DeleteIndex	Exclui um índice
DeleteTable	Exclui a tabela
DisableControls	Desabilita a atualização dos controles visuais
Edit	Permite a alteração dos campos do registro atual
EmptyTable	Apaga todos os registro da tabela, para isso a tabela não pode esta sendo compartilhada
EnableControls	Habilita os controles visuais
FieldByName	Acessa um campo, do tipo TField, pelo nome
FindKey	Procura o registro com os valores exatos aos dos parâmetros nos campos do índice atual
FindNearest	Procura o registro com os valores mais aproximados aos dos parâmetros nos índices
First	Move para o primeiro registro
Insert	Entra em modo de inserção de um novo registro na posição atual
InsertRecord	Adiciona um novo registro, já com os dados, na posição atual
IsEmpty	Define se a tabela está vazia
Last	Move para o último registro
Locate	Procura um registro, usando ou não índices, de acordo com a disponibilidade
LockTable	Trava a tabela
Lookup	Procura um registro e retorna valores dos campos deste
MoveBy	Move um número específico de registros
Next	Move para o próximo registro
Open	Abre a tabela
Post	Grava as alterações no registro atual
Prior	Move para o primeiro registro
Refresh	Atualiza a tabela com os dados já gravados
RenameTable	Renomeia a tabela
UnlockTable	Destrava a tabela
<b>Evento</b>	<b>Descrição</b>
AfterCancel	Após do método Cancel
AfterClose	Após o fechamento da tabela
AfterDelete	Após do método Delete
AfterEdit	Após do método Edit
AfterInsert	Após do método Insert
AfterOpen	Após do método Open
AfterPost	Após do método Post
AfterScroll	Após mudar de registro
BeforeCancel	Antes do método Cancel
BeforeClose	Antes do fechamento da tabela
BeforeDelete	Antes do método Delete
BeforeEdit	Antes do método Edit
BeforeInsert	Antes do método Insert
BeforeOpen	Antes do método Open
BeforePost	Antes do método Post
BeforeScroll	Antes de mudar o registro
OnCalcFields	Evento usado para calcular os valores dos campos calculados
OnDeleteError	Quando ocorre um erro ao chamar o método Delete
OnEditError	Quando ocorre um erro ao chamar o método Edit
OnFilterRecord	Evento usado com filtragem variável
OnNewRecord	Quando a tabela entra em modo de inserção, não deixa Modified igual a True
OnPostError	Quando ocorre um erro ao chamar o método Post

### Filtros

Usando o *Filter*, você pode filtrar os registro de uma tabela usando uma expressão lógica, como nos exemplos abaixo. Para tornar um filtro ativo, basta colocar *Filtered* igual a True.

```
Data = '20/04/1998'
(Data = '20/04/1998') AND (Vendedor = 'Gilherme Augusto da Fonseca')
(Nome > 'A') AND (Nome < 'B')
```

Contudo, se a condição de filtragem for muito variável, é preferível usar um código como o mostrado abaixo no evento OnFilterRecord da Table, para fazer uma filtragem dinâmica, com a propriedade Filter vazia e Filtered igual a True.

```
Accept := TblData.Value = Date;
```

Ao filtrar uma tabela, a propriedade RecordCount da Table, só mostra o número de registros que satisfazem ao filtro, como se os outros registros não existissem.

### Alterando Registros

Para alterar registros em código, colocamos a tabela em modo de edição, alteramos o valor dos campos e gravamos as alterações, se for necessário.

```
with DtmPedidos do;
begin
  Tbl.Edit;
  TblData.Value := Date;
  TblHora.Value := Time;
  Tbl.Post;
end;
```

### Inserindo Registros

Para inserir registros em código você pode usar os métodos AppendRecord e InsertRecord, caso você não precise de algum campo, mesmo assim ele deve ser informado com o valor *Null*.

```
DtmProd.Tbl.AppendRecord([Null, EdtDescricao.Text, EdtPreco.Text]);
```

### Localizando Registros

Para localizar registros você pode usar vários métodos, mas o melhor deles é o Locate, no exemplo abaixo é feita uma pesquisa exata.

```
if not DtmCli.Tbl.Locate('CodCli', Edt.Text, []) then
  ShowMessage('Cliente não encontrado.');
```

Você também pode fazer uma pesquisa parcial e/ou sem sensibilidade de caso usando o terceiro parâmetro, que é um conjunto de opções.

```
DtmCli.Tbl.Locate('Nome', Edt.Text, [loPartialKey, loCaseInsensitive]);
```

Se você precisar fazer uma pesquisa por mais de um campo, separe os nomes dos campos por ponto e vírgula e use a função *VarArrayOf* para criar um array com os valores que você quer procurar.

```
if not DtmPed.Tbl.Locate('Vendedor;Data', VarArrayOf([EdtVendedor.Text, EdtData.Text]),
  [loCaseInsensitive]) then
  ShowMessage('O vendedor não realizou nenhuma venda nessa data');
```

Caso os campos pesquisados sejam indexados, a pesquisa será muito mais eficiente, senão será criado um filtro temporário da BDE para localizar os registros

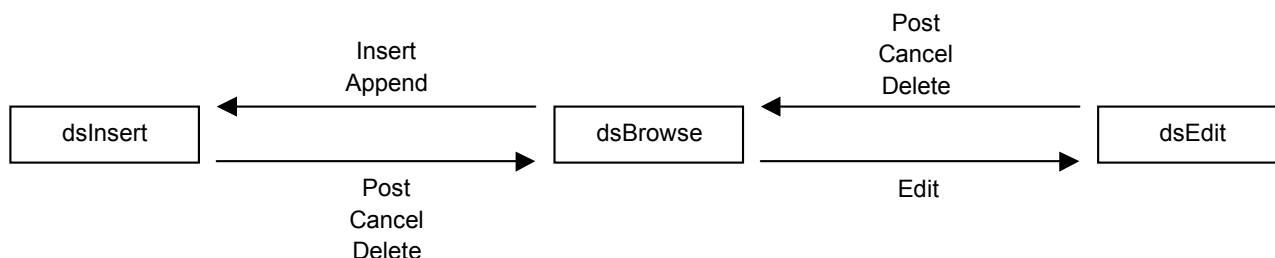
### Indexação

A indexação é usada para ordenar os registros da tabela, para isso você deve escolher os campos pelos quais você quer ordenar na propriedade IndexFieldNames, inclusive em código, como mostrado abaixo, todos campos devem ser indexados e separados por ponto e vírgula.

```
DtmCli.Tbl.IndexFieldNames := 'NomCli';
DtmPed.Tbl.IndexFieldNames := 'Data, Vendedor';
```

### Estados da Tabela

A propriedade State determina o estado das tabelas, os principais estados são demonstrados abaixo, veja como os métodos mudam o estado.



### Verificando Alterações

Onde for necessário a verificação de alterações feitas em uma Tabela, por exemplo no evento OnClose de um Form de manutenção, você pode usar a propriedade Modified, como mostrado no exemplo abaixo.

```
if DtmCli.Tbl.Modified then
  if Application.MessageBox('Gravar alterações?', 'Dados Alterados', MB_ICONQUESTION
    + MB_YESNO) = IDYES then
    DtmCli.Tbl.Post
  else
    DtmCli.Tbl.Cancel;
```

### Valores Default

Caso você queira especificar valores Default para os campos de uma tabela, use o evento OnNewRecord, pois nesse evento o registro não é marcado como modificado.

```
TblData.Value := Date;
```

## Percorrendo uma Tabela

Utilize um código semelhante ao mostrado abaixo para percorrer uma tabela do início ao fim.

```
Tbl.DisableControls;  
Total := 0;  
Tbl.First;  
while not Tbl.EOF do  
begin  
    Total := Total + TblValor.Value;  
    Tbl.Next;  
end;  
Tbl.EnableControls;
```

## Forms Modais de Inclusão/Alteração

Para mostrar Forms Modais de inclusão ou alteração de registros utilize comandos como os mostrados abaixo.

```
TblCli.Insert;  
if FormInsCli.ShowModal = mrOk then  
    TblCli.Post  
else  
    TblCli.Cancel;
```

## Mestre/Detalhe

Nos relacionamentos de 1 para N, uma tabela pode estar ligada a outra em uma relação Mestre/Detalhe, nesse tipo de relação os registros da tabela de ordem N são filtrados pelo campo de relacionamento com a tabela de ordem 1. Por exemplo, se o relacionamento de Clientes com Pedidos for mestre/detalhe, só serão acessados em pedidos, os registros cujo campo CodCli seja igual ao CodCli da tabela de Clientes.

Para fazer esse tipo de relacionamento, siga os passos abaixo.

- No uses da Unit detalhe, Pedidos, inclua a Unit da tabela mestre, Clientes
- Na Table detalhe, Pedidos, Coloque em MasterSource o DataSource da tabela mestre, Clientes
- Em MasterFields, chame o Fields Links Designer e escolha os campos de ligação das tabelas, no caso, CodCli para as duas tabelas

## Fields Editor

Para criar objetos para os campos de uma tabela clique duas vezes no componente TTable ou escolha *Fields Editor* no seu menu de contexto, na janela do Fields Editor, clique com o botão direito do mouse e escolha Add, na janela Add Fields, escolha os campos que você vai querer usar e clique em Ok.

No Fields Editor podemos também remover os campos criados, alterar sua ordem de apresentação e usar suas propriedades e eventos no Object Inspector. Para cada campo é criado um objeto de um tipo descendente de TField, como TStringField, TIntegerField, TFloatField. As principais propriedades dos objetos TField estão listadas na tabela abaixo.

Se você não criar nenhum objeto TField, todos os campos da tabela estarão disponíveis, mas caso você crie algum, somente os campos que você criar estarão disponíveis.

Se você selecionar os campos no Fields Editor e arrastar para o Form, serão criados os controles visuais para esses campos, Label, DBEdit e outros, mas antes coloque a descrição dos campos na propriedade DisplayLabel.

## TField

A classe TField é usada como ancestral para todos as classes dos campos. Geralmente iremos usar objetos de classes descendentes de TField, mas em todos eles podemos encontrar os itens mostrados abaixo.

Propriedades	Descrição
Alignment	Alinhamento do texto do campo nos controles visuais
AsBoolean	Valor do campo convertido para Boolean
AsCurrency	Valor do campo convertido para Currency
AsDateTime	Valor do campo convertido para DateTime
AsFloat	Valor do campo convertido para Double
AsInteger	Valor do campo convertido para Integer
AsString	Valor do campo convertido para string
AsVariant	Valor do campo convertido para Variant
Calculated	Indica se o campo é calculado em tempo de execução
CanModify	Indica se um campo pode ser modificado
ConstraintErrorMessage	Mensagem de erro se a condição de CustomConstraint não for satisfeita
CustomConstraint	Condição de validação do campo
DataSet	DataSet onde está o campo
DataSetSize	Tamanho do campo, em Bytes
DataType	Propriedade do tipo TFieldType, que indica o tipo do campo
DefaultExpression	Expressão com valor Default do campo para novos registros
DisplayLabel	Título a ser exibido para o campo
DisplayText	Texto exibido nos controles visuais associados ao campo
DisplayWidth	Número de caracteres que deve ser usado para mostrar o campo no controles visuais

EditMask	Máscara de edição do campo
FieldKind	Propriedade do tipo TFieldKind que indica o tipo do campo, como Calculado ou Lookup
FieldName	Nome do campo na tabela
FieldNo	Posição física do campo na tabela
Index	Posição do campo nos controles visuais
IsIndexField	Indica se um campo é válido para ser usado como índice
IsNull	Indica se o campo está vazio
KeyFields	Campo chave da tabela no relacionamento com LookupDataSet, usado em campos Lookup
Lookup	Indica se um campo é Lookup
LookupCache	Define se será usado cache para campos Lookup
LookupDataSet	DataSet onde está definido o valor do campo Lookup
LookupKeyFields	Campo chave do relacionamento em LookupDataSet
LookupResultField	Valor do campo, que será mostrado nos controles visuais
ReadOnly	Define se um campo é somente para leitura
Required	Define se o campo é obrigatório
Size	Tamanho físico do campo
Text	Texto de edição do campo
Value	Acesso direto ao valor do campo
Visible	Define se um campo é visível
<b>Eventos</b>	<b>Descrição</b>
OnChange	Chamado quando o valor do campo é mudado
OnSetText	Chamado pelos controles visuais para atribuir o texto digitado pelo usuário ao campo
OnGetText	Chamado para formatar o texto de exibição do campo
OnValidate	Validação do valor atribuído ao campo, caso o valor não seja válido, gere uma exceção
<b>Método</b>	<b>Descrição</b>
Assign	Atribui um valor de um campo a outro, inclusive nulo
FocusControl	Seta o foco para o controle visual ligado ao campo nos Forms
Clear	Limpa o conteúdo do campo

Estão listadas abaixo algumas classes que realmente iremos manipular no tratamento dos campos de uma tabela, são classes descendentes de TField.

TStringField	TBlobField	TTimeField
TSmallIntField	TIntegerField	TBytesField
TFloatField	TWordField	TVarBytesField
TCurrencyField	TAutoIncField	TGraphicField
TBooleanField	TBCDField	TMemoField
TDateField	TDateTimeField	

Em alguns desses campos você pode encontrar as propriedades mostradas abaixo, que não estão presentes em TField.

<b>Propriedades</b>	<b>Descrição</b>
MaxValue	Valor máximo para o campo
MinValue	Valor mínimo para campo
DisplayFormat	Formato de apresentação do campo, como ,0.00" %" ou ,0.##" Km"
EditFormat	Formato de edição do campo
Currency	Define se um campo é monetário
DisplayValues	Usado com campos Boolean, define o texto para True e False, como Sim;Não
<b>Métodos</b>	<b>Descrição</b>
LoadFromFile	Carrega o conteúdo do campo de um arquivo
SaveToFile	Salva o conteúdo do campo para um arquivo

Para acessar os campo de uma tabela, existem várias abordagens, como mostrado abaixo..

- Usando o objeto TField ligado ao campo.

```
TblDescricao.Value := TblVendedor.Value + ' em ' + TblData.AsString;
```

- Usando a notação de colchetes. Se você não especificar nenhuma propriedade, é assumida a propriedade Value por padrão.

```
Tbl['Descricao'] := Tbl['Vendedor'] + ' em ' + Tbl['Data'].AsString;
```

- Através do método FieldByName

```
Tbl.FieldByName('Descricao').Value := Tbl.FieldByName('Vendedor').Value + ' em ' + Tbl.FieldByName('Data').AsString;
```

- Usando a lista Fields do TTable

```
Tbl.Fields[5].Value := Tbl.Fields[3].Value + ' em ' + Tbl.Fields[4].AsString;
```

### Conversão de Tipos

A conversão de tipo de um campo pode ser feita através as propriedades tipo As..., como AsString.

```
DtmPed.TblData.AsString := EdtData.Text;
```

### Validação

Para validar os valores de um campo, você pode usar a propriedade CustomConstraint, por exemplo para garantir que a quantidade de um item seja maior que zero, use em CustomConstraint *Quantidade > 0*, e em

CustomConstraint coloque a mensagem para o usuário caso a condição seja falsa. Outra forma, mais flexível, é usando o evento OnValidate, com um código como abaixo, onde é gerada uma exceção para cancelar a atribuição do valor ao campo.

```
if TblQuantidade.Value <= 0 then
    raise Exception.Create('Quantidade deve ser maior que zero.');
```

### Formatação Personalizada

Caso queira fazer uma formatação personalizada do campo, pode usar os eventos OnGetText e OnSetText. Por exemplo, se tiver um campo *Estado*, e quiser que quando o valor do campo for *C* fosse mostrado *Casado* e *S*, *Solteiro*, no evento OnGetText use um código como o abaixo.

```
if TblEstado.Value = 'C' then
    Text := 'Casado'
else if TblEstado.Value = 'S' then
    Text := 'Solteiro';
```

Como controle visual para o usuário escolher o valor do campo, você poderia usar o DBComboBox, com *Solteiro* e *Casado* na propriedade *Items*, e no evento OnGetText do campo o código mostrado abaixo.

```
if Text = 'Casado' then
    TblEstado.Value := 'C'
else if Text := 'Solteiro' then
    TblEstado.Value = 'S';
```

### Campos Calculados

Para criar campos calculados, clique com o direito no Fields Editor e escolha *New Field*, no quadro *NewField*, digite o nome do campo, o nome do objeto será automaticamente informado, o tipo do campo, seu tamanho e escolha *Calculated* em *Field type*.

Para colocar um valor nesse campo usaremos o evento OnCalcFields do componente TTable, em nenhuma outra parte os valores desses campos podem ser alterados.

O código do evento OnCalcFields deve ser enxuto, pois este é chamado várias vezes durante a edição de um registro e um procedimento pesado pode comprometer a performance do sistema.

```
procedure TDtmAluno.TblCalcFields(DataSet: TDataSet);
begin
    if TblFaltas.Value > DtmTurma.TblMaxFaltas.Value then
        TblSituacao.Value := 'Evadido'
    else if TblNota.Value >= 7 then
        TblSituacao.Value := 'Aprovado'
    else
        TblSituacao.Value := 'Retido'
end;
```

### Campos Lookup

Para fazer um relacionamento, às vezes precisamos criar um campo de descrição, por exemplo em uma biblioteca, na tabela de empréstimos, temos o código do Livro, mas gostaríamos de mostrar o Título, esses campos são chamados de campos Lookup.

Para criar um campo Lookup, siga os passos abaixo, tomando como exemplo o caso do livro no empréstimo.

- Abra o Fields Editor do Table desejado, Empréstimos
- Clique com o direito e escolha *New Field*
- No quadro *New Field*, escolha as propriedades do campo como descrito em campos calculados, mas em *Field type*, escolha *Lookup*
- Em *Key Fields* escolha o campo da tabela que faz parte do relacionamento, *CodLivro*
- *DataSet* é a tabela onde está a descrição, *Livros*
- Em *Lookup Keys*, escolha o campo de *DataSet* que faz parte do relacionamento, *CodLivro*
- Finalmente, escolha em *Result field* o campo de *DataSet* que vai ser mostrado para o usuário, *Título*

Essas opções correspondem a algumas propriedades do objeto TField gerado, que podem ser alteradas no Object Inspector, *KeyFields*, *LookupDataSet*, *LookupKeyFields*, *LookupDataSet* e *LookupResultField*.

Quando esses campo são exibidos em um DBGrid, por padrão é criado um botão de lookup que mostrará os valores da outra tabela uma lista. Para colocar esses campos em um Form, devemos usar o DBLookupComboBox, apenas com as propriedades padrão, *DataSource* e *DataField*, onde deve ser escolhido o campo Lookup, quando você arrastar o campo para o Form isso será feito automaticamente.

### TDataSource

Componente usado para fazer a ligação entre um DataSet e os componentes visuais.

Propriedade	Descrição
AutoEdit	Define se a tabela entrará em modo de edição assim que o usuário digitar novos valores nos controles
DataSet	DataSet ao qual o TDataSource faz referência
Evento	Descrição
OnDataChange	Ocorre quando o DataSet é alterado, ao mudar de registro ou mudar os valores dos campos
OnStateChange	Ocorre quando o estado do DataSet é alterado
OnUpdateData	Ocorre antes de uma atualização

## Botões de Navegação Personalizados

O DBNavigator tem os principais botões necessários para a navegação por uma tabela, contudo se você quiser criar seus próprios botões de navegação, o que não é recomendado, no evento OnClick desses botões deve ser chamados os métodos de navegação, como indicado abaixo.

```
DtmCli.Tbl.Next;
```

Para controlar a habilitação dos botões de navegação use o evento onDataChange do DataSource correspondente como indicado abaixo.

```
BtnProx.Enabled := not DtmCli.Tbl.EOF;
```

Para criar botões de controle, como inclusão e exclusão, use o evento OnStateChange do DataSource como indicado abaixo para controlar a habilitação.

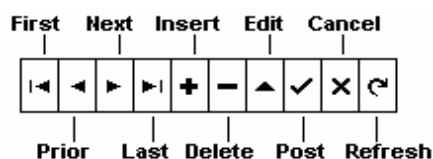
```
BtnAlterar.Enabled := DtmCli.Tbl.State = dsBrowse;
```

## Data Controls

Controles usados na interface com o usuário. Todos esses componentes tem uma propriedade DataSource, que deve ter o DataSource do Table ao qual estão ligados.

### TDBNavigator

O DBNavigator permite que o usuário realize operações padrão de controle de dados. Cada um dos botões do DBNavigator chama um método do Componente Table ao qual está ligado.



Podemos personalizar o DBNavigator usando as suas propriedades e eventos, mas se quisermos mudar a figura dos botões teremos que editar diretamente o arquivo LIBDBCTRLS.RES, na pasta do Delphi.

Propriedades	Descrição
VisibleButtons	Define os botões que serão visíveis
Hints	Hints exibidos pelos botões
ConfirmDelete	Define se será solicitado uma confirmação antes da exclusão
Eventos	Descrição
BeforeAction	Quando um botão do Navigator é pressionado, antes da ação ser executada
OnClick	Quando um botão do Navigator é pressionado, depois da ação ser executada

### TDBGrid

Mostra os registros de uma tabela em forma de grade, cada coluna é um campo e cada registro, uma linha.

Propriedades	Descrição
Columns	Lista do tipo TDBGridColumn, com as colunas da Grid, cada item da lista é do tipo TColumn
Fields	Lista de objetos TField mostrados nas colunas
Options	Set com as opções da Grid, como ConfirmDelete, MultiSelect, ColumnResize
SelectedField	Campo da coluna selecionada
SelectedIndex	Índice da coluna selecionada
SelectedRows	Lista do tipo TBookmarkList, com os registros selecionados em uma Grid com MultiSelect
TitleFont	Fonte do título das colunas
FixedColor	Cor Fixa, usada nas colunas e indicadores
Eventos	Descrição
OnCellClick	Ao clicar em uma célula da Grid
OnColEnter	Quando uma célula de alguma coluna da Grid recebe o foco
OnColExit	Quando uma célula de alguma coluna da Grid perde o foco
OnColumnMoved	Quando o usuário mover uma coluna
OnDrawDataCell	Evento usado para personalizar a forma de desenhar os dados que são apresentados na Grid
OnEditButtonClick	Ao clicar no botão de edição de uma célula, mostrado pela propriedade ButtonStyle da coluna
OnTitleClick	Ao clicar no título das colunas

### TColumn

Item de uma lista *TDBGridColumn*, usada na propriedade *Columns* da Grid, objetos desse tipo representam uma coluna da Grid. Às vezes as propriedades definidas para o campo sobrepõem as propriedades

Propriedades	Descrição
ButtonStyle	Botão mostrado ao editar as células da coluna
Field	Objeto TField ligado à coluna
FieldName	Nome do campo ligado à coluna
PickList	TStrings com os itens da lista DropDown usada nas células da coluna
Title	Propriedade do tipo TColumnTitle com as opções do título da coluna

### TDBText, TDBEdit, TDBMemo, TDBListBox, TDBComboBox, TDBImage, TDBRichEdit

Controles genéricos ligados a um campo de uma tabela.

Propriedades	Descrição
DataField	Campo ao qual o controle está ligado

### **TDBCheckBox**

Usado em campos que podem receber apenas dois valores, como campos lógicos.

Propriedades	Descrição
ValueChecked	Valor a ser armazenado quando está selecionado
ValueUnchecked	Valor a ser armazenado quando não está selecionado

### **TDBRadioGroup**

Mostra algumas opções para o preenchimento de um campo.

Propriedades	Descrição
Values	Valor a ser armazenado para cada botão de rádio

### **TDBLookupListBox, TDBLookupComboBox**

Preenche um campo com dados contidos em outra tabela. Se o campo mostrado nesses componentes for um campo Lookup, você não precisa especificar nenhuma das propriedades abaixo, apenas DataSource e DataField.

Propriedades	Descrição
ListSource	DataSource que contém os valores a serem exibidos na lista
ListField	Campo de ListSource que será exibido
KeyField	Campo de ListSource usado no relacionamento

### **Exercícios**

1. Crie uma aplicação que cadastre os Clientes de uma empresa e as Compras feitas por estes Clientes, permita inclusão, alteração, exclusão e consulta aos dados cadastrados. Na janela principal fica o cadastro de Clientes, com a grade de visualização de suas Compras, crie também uma Janela para localizar Clientes por Nome.

A tabela de clientes deve ter Nome, Endereço, Bairro, Cidade, Estado, CEP e Telefone, defina também índices para melhorar a localização de clientes por Nome. Na tabela de Compras, deseja-se saber a Data, Produtos e Valor, assuma que cada compra tem um Produto apenas. Como foi mencionado, as compras serão cadastradas pelo cliente atual, crie a relação Mestre/Detalhe entre Clientes e Compras.

O Form de localização de Clientes deve permitir pesquisa Nome, da mesma forma da questão anterior.

2. Uma academia de ginástica deseja manter um controle maior sobre seus Alunos, para isso ela organizou os clientes em turmas. Os dados de uma Turma são Número de alunos, Horário da aula, Duração da aula, Data inicial, Data final e Instrutor. Deve ser feita também uma tabela de instrutores para evitar a digitação repetitiva do Nome. Os dados dos Alunos são Matrícula, Data de Matrícula, Nome, Endereço, Bairro, Cidade, Estado, Telefone, Data de nascimento, Altura e Peso. Crie um banco de dados, normalizado, para guardar essas informações.

No cadastro de Turmas, o Horário de aulas deve ser entre 7:00 e 18:00, a Duração não pode ser maior que 2 horas e a Data Final tem que ser, no mínimo 5 dias após a Inicial. Esse cadastro deve ser ordenado primeiro pela Data Final, em ordem decrescente e depois pelo Horário, em ordem crescente. As turmas já encerradas não devem ser mostradas no cadastro, mas crie um arquivo morto com as turmas já encerradas, onde os dados não possam ser alterados. Deve ser possível também procurar o Instrutor pelo Nome, usando um ComboBox, com os registros da tabela de Instrutores.

No cadastro de Alunos, a matrícula é Auto-incremental, a Data de Matrícula deve ser, obrigatoriamente, a Data do sistema e deve ser criado um campo calculado com o peso ideal do cliente, altura menos 1,15.