Capítulo 8: Cached Updates

O que veremos neste capítulo:

- Como atualizar registros cached updates.
- Os métodos ApplyUpdates e Commit
- Os manipuladores de evento OnUpdateError e OnUpdateRecord
- Transações fora de componentes.

Visão Geral

Quando estiver trabalhando em ambientes cliente/servidor, velocidade se torna crucial. Uma vez que muitos usuários tentam acessar o mesmo registro em um banco de dados ao mesmo tempo, a manipulação direta dos dados faria com que o servidor de banco de dados viesse a falir. Uma forma de resolver este dilema é trabalhar com cached updates. Cached updates armazenam os dados em um cache local. Isto significa que o servidor tem que utilizar somente uma transação inicial para fornecer dados à máquina do usuário. Uma vez que o usuário tiver terminado todas as modificações, é necessária somente uma única transação para atualizar o banco de dados. Isto pode fazer com que o tempo de resposta seja mais eficiente neste tipo de ambiente.

No capítulo Utilizando Sessões, aprendemos que a propriedade PrivateDir contém o diretório onde o cache local é armazenado. Quando cache updates são utilizados, quaisquer alterações que o usuário fizer nos dados serão armazenadas neste diretório privativo local. O usuário pode então modificar, inserir ou apagar diversos registro localmente e, uma vez terminado, os dados podem ser gravados de volta no banco em uma única transação.

Outra propriedade discutida no capítulo Utilizando Sessões foi NetFileDir. Esta propriedade específica onde os arquivos .net e .lck serão armazenados. Para usuários do Paradox, este é o diretório onde o arquivo PDOXUSRS.NET é armazenado. Lembre-se de que este arquivo é quem controla o compartilhamento de tabelas Paradox em uma rede local. Podemos configurar tanto a propriedade PrivateDir como NetFileDir em nossa aplicação quando estivermos utilizando cached updates.

Discutiremos a significância do componente UpdateSQL na próxima seção.

(figura 8.4 - componente TQuery utilizando um UpdateObject)

Iremos agora criar um speedbuttom no form Supplier. Este speedbuttom utilizará o método ApplyUpdates para atualizar o banco de dados. Ele também gravará estas atualizações no banco de dados.

Gravando o Cache

No capítulo sobre performance em ambientes Cliente/Servidor discutimos a utilização de dados Reais vs. Não-Reais. Naquele capítulo, utilizamos o SQL Monitor e descobrimos que existe muita sobrecarga quando utilizamos dados reais. Esta se…ão focará em como modificar, inserir e apagar registros utilizando dados não reais. Para isto, utilizaremos Cached Updates.

Para começar, vejamos as propriedades de um componente TQuery:

(figura 8.1 - propriedades exibidas pelo TQuery)

As propriedades nas quais nos atentaremos são CachedUpdates, RequestLive e UpdateObject. Começaremos com CachedUpdates.

A propriedade CachedUpdates é do tipo Boolean. Quando CachedUpdates estiver configurado como true, atualizações em um dataset são armazenadas em um cache interno na máquina cliente, ao invés de serem gravadas diretamente tabelas do banco de dados. Quando as alterações estiverem completas, a aplicação poderá gravar todas as alterações do cache no banco de dados, no contexto de uma única transação. Isto oferece diversas vantagens em um ambiente cliente/servidor. Por exemplo, uma vez que as atualizações são feitas em uma única transação, a duração da transação é reduzida e o tráfego na rede local minimizado. Entretanto, existem umas poucas desvantagens. Uma vez que os usuários não estarão utilizando dados reais, um usuário pode gravar alterações em um dos dados no servidor enquanto outro ainda estiver editando o mesmo dado no cache local. Isto poderá resultar em conflitos de atualização quando as atualizações forem feitas no bando de dados. Discutiremos conflitos e exceções posteriormente neste capítulo. Por enquanto, vamos examinar a atualização dos dados utilizando o método ApplyUpdates.

UpplyUpdates

Existem dois componente que possuem o método ApplyUpdates. São os componentes TDataSet e TDatabase. Começaremos definindo o método ApplyUpdates dos DataSets.

Aplicar a atualização no nível do DataSet grava as atualizações pendentes no banco de dados. Isto não efetua um commit. Para realmente gravar as atualizações, o método Commit deve ser chamado. O exemplo a seguir demonstra como utilizar o método ApplyUpdates do Dataset. Observe que utilizamos o método Commit para gravar todas as alterações do bando de dados.

Em nosso exemplo, utilizaremos o projeto SupplierInventory, criado no capítulo processamento de Transação Client/Server. Entretanto, neste exemplo, trocaremos o componente TTable por um componente TQuery. Nosso DataModule Supplier deve se parecer com o seguinte:

(figura 8.2 - dtmdlSupplier com componentes TQuery)

Configuraremos CachedUpdates de ambos os TQuery para True.

Configuraremos também a propriedade RequestLive para False uma vez que estaremos utilizando dados armazenados em nosso cache local:

(figura 8.3 - componente TQuery utilizando CachedUpdates)

Se a aplicação fosse executada neste ponto, o usuário não poderia fazer quaisquer alterações nos dados. Como RequestLive está configurado para False, devemos

fornecer ao Delphi uma forma de atualizar o banco de dados. Isto é feito com a propriedade UpdateObject. A propriedade UpdateObject para datasets especifica qual objeto será utilizado para atualizar um conjunto de dados de resultado de somente-leitura. O tipo de objeto que utilizaremos é o componente UpdateSQL. Coloque dois componentes UpdateSQL no datamodule. Renomeie-os para updtsqlSupplier e updtsqlInventory. Selecione a propriedade UpdateObject e selecione o componente UpdateSQL para cada dataset.

O form deve estar parecido com isto:

(figura 8.5 - Form Supplier com Speedbutton Commit)

Daremos o nome de spdbtnCommit ao speedbutton. No manipulador de evento OnClick iniciaremos uma nova transação, aplicaremos as atualizações e finalmente gravaremos estas alterações no banco de dados. O código ficará:

```
procedure TfrmSupplier.spdbtnCimmitClick(Sender: TObject);
begin
  with dtmdlSupplier.grySupplier do
  begin
     dtmdlOrderEntry.dtbsOrderEntry.StartTrasaction
     try
        ApplyUpdates;
        dtmdlOrderEntry.dtbsOrderEntry.Commit;
     except
        dtmdlOrderEntry.dtbsOrderEntry.Rollback;
     raise; {gera a exceção para prevenir uma chamada
             a CommitUpdates!}
     end;//  try..except
     CommitUpdates; {limpa o cache}
  end;//  whit..do
end;
```

A primeira coisa que fizemos foi inicializar uma transação de banco de dados. Depois, em uma instrução `try...except`, aplicamos as atualizações e gravamos estas alterações. Caso ocorresse uma exceção, a integridade do banco de dados estaria protegida através de um rollback nos dados e a geração de uma exceção. Finalmente, limpamos o cache antigo, uma vez que o banco de dados foi alterado.

Caso executássemos a aplicação e fizéssemos quaisquer alterações na tabela Supplier, o que aconteceria? Vamos executar a aplicação e alterar o campo REGIONKEY. Uma vez que as alterações tenham sido feitas, dê um clique em spdbtnCommit:

(figura 8.6 - exceção gerada)

Uma exceção ocorreu informando-nos de que nenhuma instrução SQL estava disponível. Isto simplesmente significa que dentro de nosso objeto de atualização (em nosso caso, estamos utilizando componentes UpdatesSQL devemos informar ao Delphi os passos envolvidos nessa atualização. A seção a seguir sobre o componente UpdateSQL explica este processo.

Antes de continuarmos com o componente UpdateSQL, devemos discutir o método ApplyUpdates do Databases. Muito embora o método ApplyUpdates do dataset e database sejam muito similares, o componente Database possui um recurso que o Dataset não possui. O método ApplyUpdates do componente Database grava automaticamente qrava automaticamente quaisquer alterações no banco de dados. Vamos alterar um pouco o código do botão Commit para atualizar o método ApplyUpdates do Database.

O código será parecido com o seguinte:

```
procedure TfrmSupplier.spdbtnCommitClick(Sender; TObject);
begin
   with dtmdlOrderEntry.dtbsOrderEntry do
   begin
     try
        ApplyUpdates([ dtmdlSupplier.gryInventory,
            dtmdlSupplier.grySupplier ]);
     except
         Rollback; {em caso de falha, desfaz as alterações};
         raise; {gera a exceção para prevenir uma chamada
                 a CommitUpdates!}
     end;//    try..except
end;
```

Observe que não precisamos mais utilizar Commit para gravar as alterações. O método ApplyUpdates do Database aplicou as atualizações e gravou-as para nós. Tivemos somente que passar ao método o TDBDataSets que queríamos gravado. Ainda devemos utilizar o bloco `try...except` no caso de ocorrer alguma exceção. Entretanto, não tivemos que utilizar o método CommitUpdates para limpar o cache, uma vez que ApplyUpdates do Database limpou o cache por nós.

Uma nota importante: você deve ter observado que o dataset gryInventory foi colocado em primeiro lugar no conjunto, sendo passado ao método ApplyUpdates. Isto foi

necessário porque grySuppliers e gryInventory possuíam um relacionamento Master/Detail. Se tivéssemos colocado grySuppliers em primeiro no conjunto, ApplyUpdates teria gravado as alterações para grySupplier e no processo teria atualizado qualquer filho associado a ele. Isto significa que o dataset gryInventory teria sido atualizado, apagando quaisquer alterações que o usuário tivesse feito. Assim passamos gryInventory primeiro, de forma que as alterações feitas neste dataset fossem gravadas.

Componente UpdateSQL

O componente UpdateSQL é utilizado para atualizar, inserir e deletar registros em um dataset. Para isto, o componente UpdateSQL encapsula o funcionamento de três componentes TQuery. Cada um destes componentes query executa uma única tarefa. Para completar esta tarefa, instruções SQL são colocadas nas propriedades DeleteSQL, InsertSQL e ModifySQL. Vamos dar uma olhada nestas e no restante das propriedades associadas com o componente UpdatesSQL:

(figura 8.7 - propriedades do componente TUpdateSQL)

Observe que as propriedades principais do componente UpdateSQL são DeleteSQL, InsertSQL e ModifySQL. Estas três propriedades são do tipo Tstrings. É onde as instruções SQL são armazenadas para executar a ação desejada.

Ao darmos um triplo clique em uma destas propriedades, a caixa de diálogo a seguir é exibida:

(figura 8.8 - caixa de diálogo String List Editor)

Este String list editor é onde as instruções são colocadas para manipular a alteração, inserção ou deleção de registros; entretanto, o Delphi fornece uma forma mais fácil de completar a mesma tarefa.

Dê um duplo clique sobre o componente Update SQL e a seguinte caixa de diálogo é exibida:

(figura 8.9 - caixa de diálogo UpdateSQL)

A primeira coisa a ser notada são as abas no topo do form. Discutiremos a aba SQL mais a frente; por enquanto vamos nos concentrar na aba Options. Dentro de uma aba Options, uma lista drop-down é utilizada para exibir todas as tabelas disponíveis para o componente UpdateSQL. Como não temos nenhum dataset conectado ao componente UpdateSQL, as tabelas não podem ser conectadas. Para conectar um dataset a um componente UpdateSQL, devemos configurar a propriedade UpdateObject do dataset que quiser utilizar em conjunto com o componente UpdateSQL.

Anteriormente, configuramos a propriedade UpdateObject no dataset grySuppliers para o componente UpdateSQL. Uma vez que o componente UpdateSQL possui uma conexão com o dataset grySuppliers, as informações sobre este dataset devem ser exibidas na caixa de diálogo UpdateSQL.

Dê um duplo clique no componente updtsqlSupplier e a caixa de diálogo a seguir deve ser exibida:

(figura 8.10 - caixa de diálogo UpdateSQL utilizando a tabela Supplier)

A lista Table Name agora exibe a tabela Supplier como uma opção. Quatro botões também estão disponíveis. O primeiro botão, Get Table Fields, simplesmente preenche Key Fields e Update Fields com os campos da tabela selecionada na lista Table Name.

Como Supplier é o único dataset conectado com o componente UpdateSQL, Key Fields e Update Fields também estão populados.

O segundo botão, Dataset Defaults, é utilizado para restaurar os valores default do dataset associado. Isto fará com que todos os campos da lista Key Fields e Update Fields sejam selecionados e o nome da tabela seja restaurado.

O terceiro botão, Select Primary Keys, distingue qual campo é a chave primária e a seleciona na lista Key Fields. Para nosso exemplo, utilizaremos somente nossa chave primária para localizar os registros a serem atualizados.

Selecione este botão e o seguinte deve ocorrer:

(figura 8.11 - selecionando a chave primária)

O botão final, Generate SQL, faz o trabalho de caixa de diálogo UpdateSQL. Quando este botão é clicado, o código SQL é gerado para a atualização de registros. Este código SQL é criado utilizando as listas Key Fields e Update Fields.

Existe também um checkbox rotulado Quote Field Names. Esta opção simplesmente coloca aspas nos nomes de campos quando o código SQL é gerado. Alguns bancos de dados requerem aspas nos nomes de campos quando utilizar SQL. Uma vez que estamos utilizando um banco de dados InterBase, aspas nos nomes não são necessários.

Para nosso exemplo, utilizaremos o botão Generate SQL. Antes de clicarmos o botão, vejamos a aba SQL e vejamos qual o código SQL escrito para atualizar nossos registros:

(figura 8.12 - aba SQL antes do SQL gerado)

Observe as três propriedades SQL listadas no group box Statement Type. Se olharmos dentro de cada tipo de instrução, nenhum código SQL foi escrito para atualizar registros. Foi isto que causou nossa exceção anteriormente. Quando o Delphi tentou

atualizar o banco de dados, ele procurou pelo Update Object para instruções sobre como atualizar os registros. Uma vez que não escrevemos qualquer código SQL para isto, uma exceção foi gerada. Neste ponto, o código SQL pode ser escrito manualmente para cada tipo de instrução; entretanto, o Delphi fornece uma forma mais fácil de completar esta tarefa.

Se voltarmos à aba Options e dermos um clique no botão Generate SQL, o Delphi gera automaticamente o código SQL necessário para cada tipo de instrução.

A figura a seguir mostra a string ModifySQL após o Delphi ter gerado o código SQL:

(figura 8.13 - código SQL gerado)

Se olharmos o código gerado mais de perto, o registro será atualizado onde `Supplier_Key=:Old_Supplier_Key.` Como selecionamos somente Supplier_Key como Key Field, este é o único critério necessário para atualizar um registro. Caso precisássemos de mais segurança, os registro corretos poderiam ser atualizados; poderíamos simplesmente selecionar os campos necessários para verificar os registro corretos na aba Options na caixa Key Fields.

Por exemplo, poderíamos ter selecionado os campos SUPPLIER_KEY e NAME como key fields:

(figura 8.14 - selecionando mais de um Key Field)

Se dermos um clique em Generate SQL e olharmos o código gerado, veremos que o Delphi configurou o critério de atualização para que inclua os campos SUPPLIER_KEY e NAME:

(figura 8.15 - código gerado com dois Key Fields)

Agora é necessário gerar o código SQL para a tabela Inventory. Dê um duplo clique no componente updtsqlInvwentory colocado no datamodule. Dentro da caixa de diálogo UpdateSQL, dê um clique no botão Select Primary Key, seguido pelo botão GenerateSQL. Agora altere o manipulador de evento OnClick de spdbtnCommit para que fique semelhante ao seguinte:

```
procedure TfrmSupplier.spdbtnCimmitClick(Sender: TObject);
begin
  with dtmdlSupplier.grySupplier,
      dtmdlSupplier.gryInventory do
  begin
     dtmdlOrderEntry.dtbsOrderEntry.StartTransaction;
     try
        ApplyUpdates; {tenta gravar as atualizações no
                       Banco de dados};
        dtmdlOrderEntry.dtbsOrderEntry.Commit; {se OK,
                       efetua o commit};
     except
        dtmdlOrderEntry.dtbsOrderEntry.Rollback; {caso contrário
                       desfaz alterações};
     raise; {gera a exceção para prevenir uma chamada
                       a CommitUpdates!}
     end;//  try..except
     CommitUpdates; {se OK, limpa o cache}
  end;//  with..do
end;
```

Nossa aplicação agora deve manipular a modificação, inserção e deleção dos registros dentro das tabelas Supplier e Inventory.

OnUpdateError

Vamos nos focar agora em como manipular erros associados a caches updates. O evento OnUpdateError do dataset é disparado caso uma exceção seja gerada quando as atualizações são aplicadas a um banco de dados. Caso não haja nenhum código dentro do manipulador de evento quando este for disparado, o Delphi automaticamente exibe uma exceção default. Entretanto, o evento OnUpdateError pode ser utilizado para resolver diversos problemas associados à utilização de cached updates.

Para começar, vamos discutir alguns dos parâmetros passados a OnUpdateError. O trecho de código a seguir mostra o evento OnUpdateError para o dataset grySupplier:

```
procedure TdtmdlSupplier.grySupplierUpdateError(DataSet:
  TDataSet; E: EDatabaseError; UpdateKind: TUpdateKind;
  var UpdateAction: TUpdateAction);
begin

end;
```

O primeiro parâmetro é DataSet. Este parâmetro simplesmente define que DataSet está tentando aplicar as atualizações. Observe que DataSet é do tipo TDataSet. Se virmos a Visual Component Hierarchy, veremos que TDataSet encapsula TClientDataSet, TBDEDataset, TQuery, TStoredProc e TTable.

O parâmetro seguinte, E, define a classe de exceção para erros de bancos de dados. Isto simplesmente significa que a exceção ocorrida será passada ao evento OnUpdateError. Podemos utilizar este parâmetro para testar e ver qual a exceção ocorrida e aplicar a ação apropriada neste EDatabaseError.

UpdateKind é o parâmetro seguinte e é do tipo TUpdateKind. Este parâmetro descreve o tipo da atualização que gerou o erro. Existem três valores associados a UpdateKind. A tabela a seguir descreve cada um em detalhe:

Valor	Descrição
ukModify	A edição de um registro existente causou o erro
ukInsert	A inserção ou adição de um novo registro causou o erro
ukDelete	A deleção de um registro existente causou o erro.

O parâmetro final passado ao manipulador de evento OnUpdateError é UpdateAction. O parâmetro UpdateAction é do tipo TUpdateAction e indica a ação a ser tomada quando o manipulador de evento OnUpdateError for terminado.

A tabela a seguir lista os valores possíveis para UpdateAction e o que indicam:

Valor	Descrição
uaAbort	Aborta a operação de atualização sem exibir mensagem de erro
uaApplied	Não utilizado em rotinas de manipulação de erro
uaFail	Aborta a operação de atualização e exibe mensagem de erro
uaSkip	Pula a atualização do registo que causou a condição de erro e mantém as alterações não aplicadas no cache
uaRetry	Repete a operação de atualização que originalmente causou a condição de erro.

Transações não Baseadas em Componentes

Transações não baseadas em componentes são utilizadas para configurar parâmetros de transações que não podem ser configuradas com o componente TDatabase. Muito embora este tipo de transação esteja alem do escopo deste curso, veremos brevemente sua utilidade e como é utilizada. Por exemplo, podemos iniciar uma transação através da utilização do componente TQuery. Dentro da instrução SQL do componente TQuery, podemos configurar os valores default de uma transação InterBase.

```
SET TRANSACTION
ISOLATION LEVEL SNAPSHOT TABLE STABILITY
RESERVING
      CUSTOMER, ORDERS FOR PROTECTED WRITE;
```

Nesta instrução, uma transação é iniciada com um isolation level de SNAPSHOT TABLE STABILITY. Note também que duas tabelas são reservadas para minimizar a possibilidade de deadlocks.

Uma nota importante, quando utilizar transações não baseadas em componentes: o modo SQLPASSTHRU deve estar configurado para NOTSHARED. Isto significa que você deve utilizar componentes Database separados para componentes TQuery que passem instruções SQL de transação ao servidor dos outros.

Resumo

O que vimos neste capítulo:

- Vimos como atualizar registros cached updates.
- Examinamos como utilizar os métodos ApplyUpdatesSQL.
- Vimos como gerar código SQL para o componente TUpdateSQL.
- Vimos a utilidade dos manipuladores de eventos TUpdateError e OnUpdateRecord.
- Vimos brevemente como utilizar transações não baseadas em componentes.

Capítulo 9: Manipulando BLObs

O que veremos neste capítulo:

- Subtipos de BLOb no InterBase.
- Editando BLObs com Live Result Sets.
- Editando BLObs com componentes UpdateSQL.

Editando BLObs com Live Result Sets

Um BLOb é um **B**inary **L**arge **O**bject que pode conter dados binários. Ele simplesmente pode representar qualquer coisa que não possa ser facilmente armazenada em um dos tipos de dados padrão. Por exemplo, Um BLOb pode ser utilizado para armazenar:

- Imagens
- Arquivos de Som
- Vídeo
- Planilhas
- Texto

Em alguns servidores de banco de dados, o dado BLOb não é armazenado diretamente na tabela. Neste caso, ele seria um ponteiro que referencia um arquivo separado que contém o valor para aquele campo. Entretanto, no InterBase, dados BLOb são

armazenados diretamente no banco. O ponteiro na Tabela aponta para o local do BLOb no banco de dados. Os dados BLOb são mantidos completamente dentro do banco de dados, melhorando o acesso e gerenciamento dos dados.

No Delphi, campos BLOb são gerenciados pelo objeto TBlobField. Como é um descendente de TField, ele inclui diversas propriedades, métodos e eventos que são utilizados para gerenciar os dados armazenados no campo BLOb. TBlobField também é o ancestral dos objetos TMemoField e TGraphicField. TMemoField requer que o tipo de dado do campo BLOb seja texto (ftMemo) e, como você pode imaginar, TGraphicField difere de TBlobField porque requer que o campo BLOb contenha dados do tipo ftGraphic. Existem diversos outros tipos de dados para os quais TBlobField pode ser definido, mas não serão listados aqui.

No exemplo deste capítulo, criaremos um form utilizando o Form Wizard que exiba informações de clientes.

O dataset que utilizaremos é um objeto query. O form deve se parecer com o seguinte:

(figura 9.1 - form de clientes)

A tabela Customer no banco de dados OrderEntry possui um campo BLOb Comment. Ele possui um sub-tipo 1, o que significa que é do tipo texto ASCII. No InterBase, os tipos de dados dos BLObs armazenados no banco de dados são determinados por sub-tipos. Estes sub-tipos são números que o administrador do banco de dados especifica quando cria o banco de dados. Não discutiremos a forma como o InterBase armazena seus BLObs, mas é importante conhecer dois sub-tipos que o InterBase reserva para si. Se o campo BLOb for do tipo 1, ele é um campo de texto ASCII. Se for do sub-tipo 0, é um campo binário. Qualquer sub-tipo negativo que você utilize é um sub-tipo definido pelo usuário, determinado pelo DBA. A maioria dos servidores de banco de dados permite que o administrador do banco de dados defina rotinas personalizadas (tipicamente chamadas de *filters*) para permitir que o banco de dados leia e grave tipos BLOb definidos pelo usuário facilmente. Entretanto, cada servidor de banco de dados implementa esta funcionalidade de sua própria forma.

O campo Comment é definido como sub-tipo 1e, por causa disto, o Form Wizard sabe como criar um TMemoField para este valor BLOb. Desta forma, ele pode ser exibido em um componente TDBMemo. O componente TDBMemo é o componente default utilizado pelos campos BLOb de texto. Ele permite que o usuário visualize os dados armazenado neste campo.

Como utilizamos um componente TQuery como nosso dataset, devemos configurar a propriedade RequestLive para True para podermos editar o conteúdo do campo BLOb diretamente. Não há mais nada a fazer para podermos editar campos BLOb de texto. Caso não configurássemos esta propriedade para True, precisaríamos utilizar um componente UpdateSQL para editar os dados. Um exemplo disto é mostrado a seguir:

(figura 9.2 - executando o Form de Clientes/Editando o BLOb)

Editando BLObs com Componentes UpdateSQL

Nesta seção, utilizaremos o mesmo form de clientes visto na seção anterior. Entretanto, configuraremos a propriedade RequestLive da query para False. Neste exemplo, demonstraremos a utilização do componente UpdateSQL para aplicar as alterações dos dados em nosso banco de dados. Devemos também configurar a propriedade CachedUpdates da query para True para utilizarmos o componente UpdateSQL. Você pode notar que ter um campo BLOb no banco de dados não afeta este processo. Primeiro, adicionaremos um componente UpdateSQL ao DataModule:

(figura 9.3 - DataModule com o componente UpdateSQL)

Como aprendemos anteriormente, podemos dar um duplo clique neste componente para configurarmos os campos Table, Primary Key e Updateable Fields. Podemos então gerar as instruções SQL necessárias (Insert, Update, Delete):

(figura 9.9 - editor UpdateSQL)

Em seguida, precisamos configurar a propriedade UpdateObject da query para o nome do componente UpdateSQL:

(figura 9.5 - configurando a propriedade UpdateObject)

Agora, tudo o que precisamos é um local para aplicar as alterações dos dados no form. Conseguimos isto adicionando um speedbutton ao form. Este botão será utilizado para aplicar as atualizações:

(figura 9.6 - form de clientes com o botão Apply Updates)

O evento OnClick do speedbutton utiliza o código a seguir para aplicar as alterações no banco de dados:

```
procedure TfrmBLOBEdits.spdbtnApplyUpdatesClick(Sender:
TObject);
begin
  dtmdlCustomer.gryCustomer.ApplyUpdates;
end;
```

Observe que não foi preciso programação especial para se trabalhar com texto BLObs. Uma vez que o campo BLOb em nosso banco de dados InterBase foi declarado um sub-tipo 1, o Delphi foi capaz de ler e gravar neste tipo especial de dado.

Resumo

O que vimos neste capítulo:

- Discutimos o que são BLObs e como os sub-tipos BLOb são utilizados.
- Aprendemos como editar BLObs com live result sets e com componentes UpdateSQL.