COMPONENTES DA BIBLIOTECA ZEOSLIB

Paleta “Zeos Access” no Lazarus

TZConnection 

É o componente que estabelece a conexão com o banco de dados e possui a capacidade de manipular ou controlar transações. Todo acesso ao banco é sempre feito sob a execução de uma transação. Ou seja, qualquer que seja a conexão estabelecida, o acesso ao banco de dados é automaticamente realizado dentro de um contexto de uma transação. O então chamado modo “AutoCommit” está sempre ativo, setado como “True”. Como o modo AutoCommit está sempre ativo então toda alteração efetuada por um comando SQL será confirmada no banco de dados pelo COMMIT, em caso de sucesso. Se este comportamento for desabilitado, uma transação deve ser iniciada explicitamente, através do método StartTransaction. Dentro desta transação explícita é possível executar um conjunto de comandos SQL que efetuem, em seqüência, modificações no banco de dados. Este grupo de comandos podem ser confirmados por um COMMIT. Se uma transação explícita é ativada o modo AutoCommit é sempre passado para o estado desativado (“False”). A chamada ao método Commit faz com que todas as modificações efetuadas dentro deste contexto explícito de transação sejam confirmadas. Ao contrário, a chamada ao método Rollback libera (cancela) estas modificações. Em ambos os casos o modo AutoCommit será setado para True, quando o método chamado (Commit ou Rollback) for concluído e a transação explícita ser então finalizada.

Retaining

Após confirnar as modificações feitas em uma transação pelo COMMIT ou liberá-las (cancelá-las) pelo ROLLBACK a transação normalmente é concluída e um conjunto de dados resultado da execução de uma consulta (query) ou de um procedimento armazenado (stored procedure) será descartado. Estes COMMITs e ROLLBACKs são chamados de “hard commit” ou “hard rollback”. Ao se usar a biblioteca ZEOS se observará uma pequena diferença, pois ZEOS não descarta o conjunto de dados, mas os mantém. Isto é obtido porque o ZEOS finaliza a transação com “soft commits” ou “soft rollbacks”. Tudo isto é realizado pelo objeto TZConnection. Este método é chamado de “retaining”. Comandos COMMIT e ROLLBACK são executados com a adição do RETAINING. O método Retaining provoca o encerramento da transação atual e imediatamente inicia uma nova transação com todos os dados e recursos da transação anterior.

O método Retaining se torna um problema se usado em grandes tabelas. No caso do Firebird, limita o mecanismo garbage collection (coleta de lixo para fins de liberação e aproveitamento da memória) e a manutenção de um log de grande quantidade destes dados, que muitas vezes não serão mais necessários, degrada a performance. Apenas através de um hard COMMIT ou de um hard ROLLBACK os dados mantidos seriam descartados com a conseqüente melhoria da performance. A Biblioteca de ZEOS só executa estes “hard” comandos quando do fechamento da conexão com o banco de dados. Não é possível enviar estes tipos de comandos enquanto uma conexão com o banco de dados encontra-se ativa. Assim sendo, a conexão com o banco de dados deveria, freqüentemente, ser encerrada e imediatamente aberta, a fim de se obter uma melhoria no desempenho.

Níveis de Isolamento de Transação do TZConnection

O componente de TZConnection provê quatro tipos predefinidos de Níveis de Isolamento de Transação (TIL):

tiRepeatableRead

Corresponde ao TIL "SNAPSHOT” que é o padrão de servidores Firebird. É uma combinação dos parâmetros de transação “concurrency” e “nowait”. Um “snapshot” (instantâneo) do banco de dados é feito. Usuários sofrerão influência se duas transações trabalharem simultaneamente sobre um mesmo registro. Se conflitos surgem quando dados estão sendo acessados, uma mensagem de erro é retornada. Atualizações dentro de outras transações não serão nofificadas. Este TIL atende amplamente aos requisitos do SQL padrão (SERIALIZABLE).

tiReadCommitted

Corresponde ao TIL “READ COMMITTED”. É uma combinação dos parâmetros de transação “read_committed”, “rec_version” e “nowait”. Este TIL reconhece todas as atualizações efetuadas por outra transação e confirmadas através do COMMIT. O parâmetro “rec_version” é responsável pelo comportamento de que os valores mais atuais que foram “commitados” por outros usuários serão considerados. O parâmetro “nowait” é responsável pelo comportamento de que não há espera pela liberação de um registro bloqueado. Neste nível o servidor é mais sobrecarregado que no TIL tiRepeatableRead, porque tem que fazer todos os “refreshs” para adquirir estes valores novamente e iterativamente.

tiSerializable

Corresponde ao TIL “SNAPSHOT TABLE STABILITY”. É usado para obter um acesso exclusivo ao conjunto de dados retornado. Realizado pelo parâmetro de transação “consistency” que previne que uma outra transação possa ter acesso aos dados gravados. Só a transação que está

editando os dados pode ter acesso aos mesmos. Isto também previne, num ambiente multi-usuário, que outro usuário tenha acesso aos dados que estão sendo editados. Como este TIL é muito restritivo no que se refere ao acesso dr dados que estão sendo editados, deve ser aplicado com cautela.

tiNone

Nenhum TIL é usado para isolar a transação.

tiReadUncommitted

O TIL tiReadUncommitted não é suportado pelo Firebird. Se este TIL for usado, um erro será disparado e a transação não será isolada (de uso semelhante ao TIL tiNone).

Recomendação

É aconselhável isolar transações com o nível de isolamento de transação tiRepeatableRead (o padrão de Firebird). Este TIL atende amplamente aos requisitos do SQL padrão (SERIALIZABLE). Previne todos os problemas relativos a consistência que pode surgir usando transações. Uma segunda escolha seria o tiReadCommitted, mas isto depende da aplicação e da necessidade se o conjunto de dados sempre tem que ser o mais atual.

OBS.: Um TIL poder personalizado em tempo de execução. Exemplo:

```
:
ZConnection.TransactIsolationLevel := tiNone;
ZConnection.Properties.Add('isc_tpb_concurrency');
ZConnection.Properties.Add('isc_tpb_wait');
ZConnection.Connect;
:
```

Protocol (Protocolo)

Esta propriedade define qual o banco de dados a ser acessado.

Read-Only Connection

A conexão com o banco de dados mantida pelo TZConnection é setada, por padrão, como somente leitura (ReadOnly = True). Isto significa que a gravação de dados não é permitida. Para se poder gravar dados, você deve setar esta propriedade para False (ReadOnly = False).

Codepages (código de página)

No TZConnection esta propriedade é setada através dos parâmetros “lc_ctype” ou “Codepage”. Estes parâmetros podem ser adicionados na pela propriedade “Properties”. Exemplo:

```
ZConection.Properties.Add ('lc_ctype=ISO8859_1');

ou

ZConnection.Properties.Add ('Codepage=ISO8859_1');
```

HostName

Normalmente o nome do servidor ou o endereço IP é atribuído a esta propriedade. Se o banco se encontrar na máquina local atribua-lhe o valor “localhost”.

Connected

Esta propriedade ao ser setada como True, estabelece a conexão com o banco de dados, se tudo ocorrer bem. Evite ativar esta propriedade em tempo de projeto, principalmente antes de compilação. Procure estabelecer a conexão através de linhas de código, em tempo de execução, como por exemplo, no evento OnCreatedo

formulário principal da aplicação e fechá-la quando do evento OnDestroy d formulário principal.

Exemplo de Uso do TZConnection:
(Este exemplo demonstra também a criação de um banco em tempo de execução)

```
:
ZConnection1.Database := 'd:\db1.fdb';
ZConnection1.Protocol := 'firebird-1.5';
ZConnection1.Properties.Add
('CreateNewDatabase=CREATE DATABASE ' +
QuotedStr ('d:\db1.fdb') + ' USER ' +
QuotedStr ('sysdba') + ' PASSWORD ' + QuotedStr
('masterkey') +
' PAGE_SIZE 4096 DEFAULT CHARACTER SET ISO8859_1');
ZConnection1.Connect;
:
```

Rolename (papéis)

A esta propriedade deve ser atribuído o papel que define os direitos e privilégios de um usuário ou grupo de usuários estabelecidos no banco de dados. O Firebird não suporta este recurso.

TZQuery 

É recomendado o uso do TZQuery no modo RequestLive associado com o componente TZUpdateSQL.

Se um conjunto de dados deve ser atualizável, então RequestLive deve ser setado como True e usado com o componente TZUpdateSQL, a fim de se usar os comandos SQL de ação (update, insert, delete).

Para retornar um conjunto de dados você deve chamar o método Open. Caso queira executar uma query de ação (que não retorna um recordset) chame o método ExecSQL, para os comandos update, delete, insert.

TZReadOnlyQuery

Este componente é similar ao TZQuery, porém com a diferença de que recordset retornado é somente leitura. Conseqüentemente não é possível o uso associado do TZUpdateSQL.

TZUpdateSQL 

O TZUpdateSQL fornece comandos SQL de atualização (update, insert, delete) para um conjunto de dados, recuperados por um TZQuery. Exemplo de uso:

ZQuery1.Sql:

```
SELECT * FROM names
```

UpdateSQL1.InsertSql:

```
INSERT INTO names (recno, name)
VALUES      (:recno, :name)
```

UpdateSQL1.ModifySql:

```
UPDATE names
SET    recno = :RecNo,
         name  = :name
WHERE  recno = :old_recno
```

UpdateSQL1.DeleteSql:

```
DELETE FROM names
WHERE      recno = :old_recno
```

O Prefixo de Parâmetro "OLD_"

O prefixo "old_" de um TFieldName possibilita o acesso ao valor do referido campo antes da sua modificação. Isto é útil quando se necessita comparar valores de campos na montagem de uma cláusula WHERE.

Consultas com Conjunto de Dados Somente Leitura

Geralmente um TZUpdateSQL está associado a um TZQuery que retorna um recordset somente leitura, porque as consultas, em sua maioria, retornam dados advindos da junção de várias tabelas. Entretanto, como visto anteriormente, é possível usar um TZUpdateSQL com um conjunto de dados atualizável (RequestLive).

Múltiplos Comandos com TZQuery e TZUpdateSQL

Os componentes TZQuery and TZUpdateSQL fornecem a possibilidade de executar internamente múltiplos comandos. Na propriedade SQL, eles podem ser definidos com parâmetros e devem ser separados por ponto-e-vírgula. Exemplo:

```
:
With Query do
    Begin
       Sql.Clear;
       Sql.Add('DELETE FROM table1;');
       Sql.Add('INSERT INTO table1 VALUES (:Val1, :Val2);');
       Sql.Add('INSERT INTO table2 VALUES (:Val3, :Val2);');
       Sql.Add('UPDATE      table3 SET field1 = :Val4;');
       Params.ParamByName('Val1').AsInteger := 123;
       :
       ExecSql;
    End;
:
```

Os comandos serão executados obedecendo à ordem de seqüência. Também é possível agrupar internamente múltiplos comandos em um TZUpdateSql, a fim de se atualizar múltiplas tabelas.

TZTable 

O componente TZTable deve ser utilizado apenas em aplicações cliente-servidor que manipulam pequenas tabelas, uma vez que todos os registros da tabela serão transferidos para a memória da máquina cliente. É equivalente ao comando SQL “SELECT * FROM ANYTABLE”.

TZStoredProc

O componente TZStoredProc fornece a possibilidade de executar um procedimento armazenado (stored procedure) de um banco de dados. Existem dois tipos de procedimentos armazenados. O que retorna um conjunto de dados e o outro que não retorna. Não é necessário chamar o método Prepare antes de executar o método ExecProc.

Stored Procedures com Retorno de um Conjunto de Dados

Se um procedimento armazenado retorna um conjunto de dados, ele deve ser chamado através do método Open, semelhante a uma Query. Quando existirem parâmetros de passagem, seus valores devem ser atribuídos antes. O conjunto de dados retornado pode ser manipulado como o recordset de um TZQuery.

```
:
With spSumByName do
    Begin
      Close;
      ParamByName ('Name').Value := 'DontKnowHow';
      Open;
    End;
:
```

Stored Procedures sem Retorno de um Conjunto de Dados

Se um procedimento armazenado não retorna um conjunto de dados, ele deve ser chamado através do método ExecProc. Quando existirem parâmetros de passagem, seus valores devem ser atribuídos antes.
Exemplo (com conConnection.AutoCommit = True):

```
:
With spDeleteByName do
    Begin
      ParamByName ('Name').Value := 'DontKnowHow';
      conConnection.StartTransaction;
      Try
```

```
            // execute StoredProc
            ExecProc;
        Except
            conConnection.Rollback;
        End;
        conConnection.Commit;
    End;
:
```

Stored Procedures Usando TZQuery

Executar um procedimento armazenado no TZQuery é de maneira similar a do componente TZStoredProc. Na propriedade SQL monte a string.
Exemplo (com conConnection.AutoCommit = True):

Zquery1.SQL:

```
EXECUTE PROCEDURE DeleteByName (:Name)

:
// using a TZQuery object
With qryDeleteByName do
    Begin
        ParamByName ('Name').Value := 'DontKnowHow';
        conConnection.StartTransaction;
        Try
            ExecSQL; // execute SQL statement in TZQuery
        Except
            conConnection.Rollback;
        End;
        conConnection.Commit;
    End;
:
```

Consultas Master/Detalhe (Master/Detail) com ZEOS

Você pode usar coomponentes TZQuery ou TZReadOnlyQuery para estabelecer uma conexão do tipo master/detalhe.

Master SQL
Zquery1.SQL:

```
SELECT master_id, feld1, feld2, feld3
FROM   MasterTable
```

Detalhe SQL
Zquery1.SQL:

```
SELECT feld1, feld2, id
FROM   detail
WHERE id = : master_id
```

Exemplo de inserção de um registro na Tabela Detalhe:

```
Procedure dmMasterDetail.qryDetailNewRecord (DataSet: TDataSet);
Begin
      qryDetailMASTER_ID.Value := qryMasterID.Value;
End;
```

Exemplo de consulta de um registro na Tabela Detalhe, a partir da navegação na Tabela Master:

```
Procedure dmMasterDetail.dsMasterDataChange (Sender: TObject; Field: TField);
Begin
      With qryDetail do
          Begin
             Close;
             ParamByName('id').Value := qryMasterID.Value;
             Open;
          End;
End;
```

Cached Updates

Você tem que setar esta propriedade para True para os componentes TZTable, TZQuery ou TZStoredProc, nas situações de operação de atualização. Desta forma todas as atualizações realizadas em cache poderão ser facilmente “comitadas” através da execução dos métodos ApplyUpdates e CommitUpdates. O método CancelUpdates cancela as alterações efetuadas em cache.
Exemplo (com AutoCommit = True em TZConnection):

```
:
With DataSet do
    Begin
      bEverythingOK := True;
      CachedUpdates := True;
      DataSet.First;

      While (not DataSet.EOF) and (bEverythingOK) do
        Begin
          DataSet.Edit;
          :
          // process record
          :
          DataSet.Post;
          DataSet.Next;
          :
          bEverythingOK := AFunctionForValidation;
        End;

      If bEverythingOK Then
        Begin
          ApplyUpdates;
          CommitUpdates;
        End
      Else
          CancelUpdates;

      CachedUpdates := False;
End;
:
```

Campos BLOB

A biblioteca ZEOS é capaz de manipular campos Blob. A seguir um exemplo de como inserir, em uma tabela, um novo registro com um campo Blob. O campo Blob é preenchido com um bitmap. Para tanto, devemos usar um Stream. Stream é útil para manipular e transferir dados binários em memória.

```
:
Var TheStream : TMemoryStream;
    Begin
      TheStream := TMemoryStream.Create;
      Try
```

```
            Image1.Picture.Bitmap.Savetostream(TheStrea
      m);
            With qryBlobInsert do
              Begin
                Sql.Text := 'INSERT INTO EVENTS
            (EventNo,EVENT_PHOTO) ' +
                            'VALUES (100,:ThePicture)';
                Params.Clear;
                Params.CreateParam(ftBlob,'ThePicture',
            ptInput);
                ParamByName('ThePicture').LoadfromStre
            am(TheStream,ftBlob);
                ExecSQL;
              End;
      Finally
            TheStream.Free;
      End;
    End;
:
```

OBSERVAÇÕES:

Meu link:
http://www.pauloamaral.com.br/cefetse.pa/zeoslib_tutorial.htm

1) Este tutorial é uma tradução e adaptação do seguinte tutorial:
The ZeosLib DBOs 6.1.5 - With Delphi 7 and Firebird 1.5
Disponível em: http://forum.zeoslib.net.ms/ (knowledge Base)
Description: An introduction to the basic usage of ZeosLib DBOs and Firebird basics
Author: Michael
Date: 20.09.2005, 12:42
Type: Tutorials Category ZeosLib Database Objects

2) Como o tutorial de Michael se baseou na aplicação do Zeos para o Firebird, talvez alguns recursos da biblioteca aqui descritos possam não ser suportados por outros bancos de dados (PostgreSQL, MySQL, etc.).

Em: 17/04/2006
Cohako Namara