Trabalhando com Transações

Introdução

Resumo: Quase tudo que acontece num banco de dados interbase, acontece num contexto de uma transação. Transações são o coração de qualquer aplicação cliente/servidor bem implementada. Com IBO você tem 100% de visibilidade e controla tudo o que acontece com transações nas suas aplicações. Com isso vem a responsabilidade de entender como usar transações.

Se você começa a usar Interbase e antes usava banco de dados desktop como Paradox, dBase e Access, então o uso de transações pode ser um pouco confuso para você.

Sistemas Desktop usam a estratégia de travar registros para prevenir que usuários atualizem registros que estão sendo usados por outros usuários. Lidar com travamentos pode ser bastante complicado e confuso em qualquer coisa nos banco de dados mais simples. Isto é especialmente verdadeiro se você quer fazer muitas alterações e ou você grava todas ou não grava nenhuma. Isto é uma das coisas que as transações fazem facilmente.

Básico sobre Transação

A forma mais fácil de entender o conceito de transações é quebrá-lo em três partes distintas ou aspectos: Explicita, física e lógica. Será muito importante que você conheça estes aspectos porque eles serão freqüentemente mencionados no restante deste documento.

Transações Explicitas.

O aspecto em que a maioria das pessoas é familiar é a transação Explicita. Que é quando o desenvolvedor explicitamente toma controle da transação. Isto é feito chamando os métodos: StartTransaction, Commit e Rollback. Isto permite ao desenvolvedor ter o controle explicito das suas transações mas também o impede de saber tudo o que esta acontecendo para poder acompanhá-lo. A propriedade InTransaction diz se uma transação explicita foi inicializada ou não.

No IBO transações explicitas funcionam exatamente como no BDE, então não haverá problemas sérios se você quiser continuar trabalhando como fazia (desde que você estava fazendo as coisas certas).

Transações Físicas.

Transações físicas ficaram escondidas no BDE, mas o IBO as expôs totalmente. IBO automaticamente inicia uma transação física quando necessário. A transação física representa a existência atual de uma transação no servidor e a presença de um handle p/ a transação no cliente. A propriedade Started mostra se a transação foi iniciada ou não.

A partir do momento que uma transação física é iniciada, interbase ativa um contexto que “protege” esta transação de outras alterações processadas no banco de dados. E também providencia um contexto para gerenciamento de conflitos se dois usuários tentarem atualizar a mesma informação.

A proteção de uma transação significa que o Interbase é prevenido de executar garbage collection alem da mais antiga transação (OAT – Oldest active transaction). Em troca, qualquer transação com alterações de dados não confirmadas (commit), previne outras transações de executar outras alterações nestes registros. Por esses dois motivos, transações físicas não devem ser mantidas abertas desnecessariamente por longos períodos de tempo.

IBO tem diversos mecanismos para termos certeza que uma transação física somente será mantida aberta pelo tempo absolutamente necessário. Na maioria dos casos, IBO toma total controle das transações físicas p/ você. Mais adiante discutiremos mais sobre OAT (Oldest active transaction).

Transações Lógicas

As transações lógicas tem por aspecto fazer com que instruções com alterações de dados ou metadata (alterações, inclusões, etc.) sejam enviados ao servidor. Isto é que define se uma unidade de trabalho será confirmada ou descartada como um todo. Assim que o dataset entre no estado de edição, ou uma instrução SQL envolvida em uma operação de DML (Data Manipulation Language) ou DDL (Data Definition Language) é executada com sucesso, uma transação lógica está presente.

A propriedade TransactionIsActive mostra se uma transação lógicas esta presente ou não. Uma transação lógica termina quando um lote de trabalho é ou confirmada (commit) ou descartada (rollback) usando qualquer um dos métodos que o acompanham.

O aspecto lógico é distinguido dos outros porque é possível para uma transação explicita ser iniciada chamando o método StartTransaction sem uma transação lógica existir. Se o usuário não fez qualquer alteração, não existe uma transação lógica apesar de uma explicita estar presente. É muito útil saber quando o usuário fez alguma alteração, ou estar p/ fazer, apesar da presença de uma transação explicita.

Se nenhuma transação lógica esta presente, não faz diferença se Commit ou RollBack é chamado para terminar uma transação física, desde que nenhuma alteração esta pendente para se preocupar. Em muitos casos, a transação física termina quando a física termina, mas não é sempre.

Uma Transação está em Progresso?

A propriedade Started mostra se existe ou não uma transação física alocada. Uma transação física deve ser iniciada apenas para mostrar os dados pela query. Ela será iniciada automaticamente p/ você pelo IBO sempre que necessário. Sob certas circunstancias IBO também finaliza transações físicas.

A propriedade TransactionIsActive mostra houve ou não uma alteração confirmada (post) ou se uma alteração está pendente para o componente de transação. Em termos lógicos uma transação é começada no momento em que o dataset entra em um estado onde uma condição de alteração existe. Se aquela mudança é cancelada e nenhuma outra mudança foi confirmada então a transação lógica não é mais ativa ou "iniciada ". Uma vez uma alteração é confirmada a transação então é ativa até aquela mudança ser ou confirmada (commit) ou descartada (roll back).

A propriedade InTransaction mostra se você chamou ou não o método StartTransaction para começar uma transação explícita. Começando a transação explícita suspenderá o AutoCommit se ele tiver sido configurado. O AutoCommit retoma depois da transação explícita é terminado. Isto acontece quando Cometa, CommitRetaining, Rollback, RollbackRetaining, Refrescam (), Feche, etc. foi chamado causando um cometa ou rollback para acontecer. SavePoint não reajusta o comportamento de AutoCommit como não é considerado um fim para a transação embora faça todas as mudanças àquele ponto permanente.

O Inicio de uma transação explícita suspenderá o comportamento AutoCommit se isso foi fixado como true. Comportamento AutoCommit retomará depois que a transação explícita for terminada. Isto acontece quando Commit, CommitRetaining, Rollback, RollbackRetaining, Refresh(), Close, etc. foram chamados causando um commit ou rollback. SavePoint não reseta o comportamento AutoCommit como não é considerado um fim da transação embora faça todas as mudanças àquele ponto permanente.

Assim, para responder esta pergunta, depende de que aspecto da transação que você quer saber. Se você precisa saber que se há uma transação física em andamento use a propriedade Started, para verificar uma transação lógica use a propriedade TransactionIsActive e para verificar uma transação explícita use a propriedade InTransaction.

Fechando uma Transação

Com o primeiro Prepare de uma query associada a uma transação, uma transação física é automaticamente iniciada. Debaixo da maioria das circunstâncias IBO garantirá que a transação física não ficará aberta muito tempo. Se você sentir a necessidade de garantir que uma transação não ficará aberta muito tempo, aqui está alguns modos que a transação física pode ser fechada.

Os seguintes métodos dos componentes TIB_Transaction ou TIBODatabase fecharão a transação física.

Close;
Commit;
Rollback;
Refresh(CommitChanges: boolean);
ChangeIsolation(NewIsolation: TIB_Isolation; CommitChanges: boolean);
Pause/Resume;

Quando Close é chamado, ele tentará dar um Commit se AutoCommit for true; caso contrário, Rollback será executado. Se uma exceção acontece durante o Commit ele prosseguirá e chamará Rollback ao invés. Uma chamada a Close também fecha todos os datasets associados com a transação. Se um objeto de Transação ativa for destruído, Close será chamado no destructor.

Chamando Commit ou Rollback causa a o termino da transação física e cada dataset executa de acordo com sua própria setagem para CommitAction. Para datasets nativos do IBO a ação padrão é fechar. Para os componentes TDataset a ação padrão é ficar aberto mas invalidar o cursor se um ainda estiver aberto (i.e., registros unfetched permanecem no servidor). Se havia um cursor aberto, o dataset dará refresh depois que a transação terminar para estabelecer um cursor novo.

Refresh é essencialmente o mesmo que chamar Commit ou Rollback, Exceto que força todo o datasets dar refresh junto com o fim da transação física, Apesar do que cada propriedade CommitAction de cada dataset seja setada.

ChangeIsolation é quase igual a Refresh, exceto que ele permite mudar o isolamento usado pela transação. Você não pode mudar a propriedade de Isolamento diretamente enquanto uma transação está ativa. Esta função lhe permite dar refresh na transação e mudar seu Isolamento durante o curto período que a transação esta inativa.

Os métodos Pause/Resume permitem ao desenvolvedor parar uma transação sem fechar todo os datasets associados e essencialmente congelar a aplicação até que o usuário esteja pronto para voltar a trabalhar. Pause efetua o congelamento e Resume reativa a transação. Isto é útil em casos onde um grande dataset exigiria para o sistema constantemente dar refresh para manter o dataset aberto e não manter uma transação física aberta muito tempo.

CommitRetaining, RollbackRetaining e SavePoint não dão reset na transação física.

AutoCommit e StartTransaction

Se Autocommit é true então uma vez que o datase grava(post) suas alterações estas são fisicamente commitadas no servidor através da chamada interna ao método SavePoint.

A propriedade AutoCommit não deveria ser mal interpretada como um modo do IBO de automaticamente assegurar que commits estão ocorrendo, como para prevenir que uma transação física fique aberta muito tempo. Enquanto AutoCommit ajuda manter transações curtas, ele próprio não é uma garantia. O mecanismo de administração de OAT do IBO independentemente executa commits automáticos para ter certeza uma transação física não é mantida desnecessariamente aberta mais que o necessário.

Chamando o método StartTransaction temporariamente suspende o comportamento AutoCommit até um "fisico" commit ou rollback for chamado. Quando Autocommit é suspenso, mudanças gravadas(post) permanecerão em um estado não confirmadas (uncommitted) e capaz ser desfeito(rollback) se necessário.

Resumindo esta situação, embora AutoCommit seja true, se você está em uma transação explícita pelo chamado a StartTransaction, uma vez que as alterações sejam gravadas (post) então uma transação lógica permanecerá ativa até que Commit, CommitRetaining, Rollback ou RollbackRetaining, etc. for chamado. Então, neste momento a transação lógica e explícita terminará.

Se chamar Commit ou Rollback então a transação física também terminará. Tenha cuidado desde que isto também pode fechar automaticamente qualquer datasets aberto (dependendo do CommitAction fixado do dataset).

Isolamento da transação

Este tópico discute isolamento de transação. Cada seção neste tópico lista uma série de passos para demonstrar como diferentes configurações do isolamento de transação impactam o usuário dos dados associado com uma transação particular. Para entender cada seção você deve se referir ao tutorial de TransactionPausing distribuído com IBO - que tem dois datasets, cada associado com uma transação diferente mas cada que se referindo à mesma tabela InterBase.

Um dataset (query “Controle”) te permite especificar o isolamento da transação.
Outro dataset (query “AUTO”) esta associada com a transaçãp padrão (default), o que significa que o isolamento da transação esta sempre setada p/ Ticommited.

Cada dataset tem seu próprio grid p/ permitir tentar alterar os dados.

Somente Alterações confirmadas (commit) são visíveis para outras transações.

1 – Tenha certeza que a transação da query “controle” esta setada como tiCommited.
2 – Inclua ou altere um registro no grid da query “Controle” e confirme as alterações (post) mas não clique em nenhum botão p/ commitar a transação.
3 – Selecione a query “Auto” e clique no botão refreshAll no toolbar da query. Você verá que todas as alterações que você fez no grid “controle” NÃO aparecem neste grid.
4 – Clique no botão CommitRetainning (que dará commit nas alterações feitas no grid “controle”) e então repita o item 3. Agora você verá que as alterações feitas no grid “controle” aparecerão neste grid.

NOTA: As alterações apareceram no grid “Auto” porque o isolamento da transação esta setada p/ tiCommited, então para ver as alterações elas devem ser confirmadas (commit).

Isolamento tiConcurrency tira um " Instantâneo " quando a transação inicia.

1 – Altere o isolamento da transação da query “Auto” p/ TiConcurrency.
2 – Inclua ou altere um registro no grid da query “Controle” e confirme as alterações (post) . Como a transação default esta setada p/ Autocommit, você verá o resultado das alterações sendo confirmadas (commit).

3 - Selecione o qrid da query “Auto” e clique no botão RefreshAll da toolbar da query. Você NÃO irá ver as alterações que você fez no grid “controle” aparecerem no grid “Auto”. Isto é porque o Isolamento tiConcurrency tira um “" Instantâneo " quando a transação é iniciada e ignorará qualquer alteração feita por outras transações - mas verifique que este modo de Isolamento não parou as alterações que foram feitas, você apenas não as pode ver. (Você está olhando velhas versões dos registros mantidas pelo Interbase para este propósito - que é o que faz o Interbase “multi-generational “.)
4 – Se você confirmar (commit) ou desfazer (rollback) a transação e reabrir a query voce verá que as alterações aparecerão (Isto não acontecerá se você usar CommitRetaining).

Isolamento tiConsistency bloqueia Alterações

1 – feche e reabra sua aplicação para sabermos em que estado ele esta.
2 – Inicie a transação “Controle” e troque seu isolamento para TiConcurrency então abra a query “controle” (selecione o grid da esquerda e então clique no botão Open Query).
3 – Abra a query “Auto” (selecione o grid da direita e então clique no botão Open Query).
4 – Tente excluir um registro na query “Auto”. Você receberá uma mensagem de exceção “Lock Conflict”.

Isto deveria destacar por que tiConsistency deve ser usado com grande cuidado. Ele travará TODOS os registros selecionados pela transação e prevenirá mudanças para quaisquer destes registros mesmo se você não tivernenhum uso para eles. Assim se você executa um " SELECT * FROM TABLE "; então você travou TODOS os registros da tabela – o que geralmente é um efeito indesejável.

Revisado para IBO4

Pausando a transação

Por causa dos problemas potenciais causados por deixar uma transação física aberta muito tempo, IBO provê várias capacidades embutidas para minimizar o uso de transações físicas e também vários métodos para lhe permitir ter controle preciso da situação.

Este tópico discute como usar o processamento Pause/Resume agora disponível com o TIB_Transaction e se refere ao código do projeto tutorial do TransactionPausing distribuído com IBO.

A idéia atrás deste conceito é interromper uma transação (parar a transação física/servidor (physical/server)) quando não estiver realmente em uso, sem limpar todos os datasets associados, e então permitir retomar (resume) a transação quando o usuário deseja começar ativamente a usar novamente os dados.

Isto realmente é o mesmo que a função TIB_Transaction.Refresh, mas dividido em duas partes. Parte 1 (Pause) fecha a transação física sem notificar o datasets, Parte 2 (Resume) cria uma transação nova e da refresh no datasets associado.

Na aplicação de demonstração você achará um timer que inicia a função seguinte (o timer é fixado em 1 segundo para o que a demonstração pretende, eu sugeriria que em uma aplicação real o timer devesse ser fixado em um minuto ou até mais longo).

```
procedure TTransactionMainForm.TransactionTimerTimer(Sender: TObject);
begin
  UpdateTransactionList;
  CheckTransactions;
end;
```

Você pode ignorar a função 'UpdateTransactionList' uma vês que ela simplesmente mantem uma lista das transações no rodapé do form. O que é somente necessária para demonstração.

Pause

Na procedure CheckTransactions você verificará o processo atual introduzido para interromper transações quando eles têm muito mais tempo que desejado. Esta função é grandemente simplificada do que você pode usar em uma aplicação real, mas deve lhe dar uma idéia do que precisa ser feito. A seguir é uma descrição da função

```
procedure TTransactionMainForm.CheckTransactions;
var
  i: integer;
  iList: TList;
  iTran: TIB_Transaction;
  OAT: TDateTime;
  PauseEnabled: boolean;
begin
iList := TIB_Session.DefaultSession.Session_Transactions;

{TIB_Session.DefaultSession dá (e somente) acesso à sessão default para o processo principal.  A vida fica mais complicada em uma aplicação multi- threaded mas isso é um assunto para outra demonstração.  O componente de TIB_Session mantém uma lista de todas as transações para a sessão num TList chamado Session_Transactions. }

TransactionStateGrid.RowCount := iList.Count + 1;
OAT := 0;
PauseEnabled := true;
for i := 0 to iList.Count - 1 do begin
```

```
   iTran := TIB_Transaction(iList.Items[i]);
   if OAT < iTran.TimeActive then
      OAT := iTran.TimeActive;
```
{Acima, eu asseguro que OAT representa o TimeActive para a transação que esta ativa a mais tempo. TimeActive devolverá 0 se a transação não esta ativa, caso contrário TimeActive é a diferença entre agora e o tempo que transação foi iniciada (LastStarted). Também veja LastStopped para o tempo a transação teve sua ultima parada.}

```
   if iTran.IsPauseDisabled or iTran.IsPaused then
      PauseEnabled := false;
```
{Acima eu asseguro que não tentarei pausar a se qualquer transação já estiver em pausa ou a pausa estiver desabilidata}
```
   end;
if PauseEnabled and ((OAT * 86400) > 15) and
```
{Acima estou conferindo o que eu quero pausar (nenhuma transação já foi pausada ou a pausa foi desabilitada), e que eu só tento pausar se a OAT tem mais que 15 segundos. Obviamente em uma aplicação real esta duração provavelmente seria fixada em 15 ou 30 minutos (ou o que for apropriado a uma aplicação em particular).}

```
   (((SysUtils.Now - FLastActivity) * 86400) > 15) then begin
```
{Acima é uma verificação especial para ver se o usuário usou o form ativamente nos últimos 15 segundos (em uma aplicação real este tempo provavelmente seria de 5 ou 10 minutos). Esta verificação é para tentar evitar que o processamento da pausa possa interromper o usuário no meio da digitação. Veja a procedure DoAppMessage (definida para Application.OnMessage), onde FLastActivity é atualizado ao tempo atual a toda atividade do teclado ou do mouse. }

```
   for i := 0 to iList.Count - 1 do begin
      TIB_Transaction(iList.Items[i]).Pause( false );
```
{Acima é a chamada ao Pause, com o false indicando que qualquer alteração será descartada (rollback)}
```
      end;
   PostMessage( Handle, WM_APP + 1, 0, 0 );
```
Em lugar de segurar a resposta para a mensagem do timer eu gravo (post) uma mensagem de volta nesta janela que resultará na apresentação de um form modal - veja a procedure ShowPauseMessage. }

```
   end;
end;
```

Resume

Uma vez você interrompeu uma transação é importante impedir que o usuário tente interagir com os dados até a transação for reativada(resume). Qualquer tentativa para interagir com dados em uma transação pausada resultará em uma exceção "Transaction Paused " o que não é muito ruim mas no

momento você pode receber também várias mensagens "Cursor Unknown" que podem confundir o usuário. Eventualmente pode ser possível impedir que estes erros inesperados venham a acontecer, mas mesmo assim somente é realmente apropriado assegurar o que é visualmente óbvio para usuário que é, os dados não estão disponíveis no momento. Na demonstração eu faço isto com a seguinte procedure...

```
procedure TTransactionMainForm.ShowPauseMessage( var Msg: TMessage );
var
  i: integer;
  iList: TList;
begin
Application.MessageBox( 'Transactions Paused.  Click OK to Resume',
   'Paused', MB_OK );
{Acima Eu mostro uma mensagem Modal. Que pode parao o precessamento ate que o susuario clique no botão OK. Isto previne que o usuário não esta tentando mexer nos dados até que a mensagem for mostrada}

iList := TIB_Session.DefaultSession.Session_Transactions;

TransactionStateGrid.RowCount := iList.Count + 1;
for i := 0 to iList.Count - 1 do begin
   TIB_Transaction(iList.Items[i]).Resume( true );

{Acima nós chamam Resume(true) que resulta em todos os datasets serem atualizados (refresh). Você poderia chamar Resume(false) quando você não precisar que os datasets serem reiniciados/atualizados (restarted/refreshed) (e neste caso eles serão fechados) - talvez você esteja fechando o form ou algo parecido. É seguro chamar Resume até mesmo quando a transação não está em pausa – uma vez que simplesmente ele será ignorado. }
   end;
end;
```

DisablePause e EnablePause

Na maioria das aplicações haverá vezes quando você não quer a transação seja pausada. Talvez você tenha um relatório ou um longo processo rodando que você não quer que sejam interrompidos. Para fazer isso você pode simplesmente usar os métodos DisablePause e EnablePause do TIB_Transaction. Na aplicação de demonstração eu uso uma simples checkbox para prevenir a pausa, mas na sua aplicação rela você pode ter algo como...

```
 try
   Transaction.DisablePause;
   RunLongPocedure;
 finally
   Transaction.EnablePause;
 end;
```

Verifique que você tem que ter uma chamada a EnablePause para toda chamada para DisablePause - ou a pausa não poderá ser reativada. Por isto você geralmente usará blocos de try/finally como mostrado acima.

Se você chama DisablePause quando a transação já esta em pausa você receberá uma exceção "Transaction Paused ".

Usando a aplicação de Demonstração

Eu sugiro que você experimente a aplicação de demonstração, ambos com o checkbox 'Pause Enabled' marcado e desmarcado. Isto deve ajudar a destacar a importância de encontrar um modo de assegurar que as transações físicas sejam regularmente resetadas - e os perigos de confiar em transações '<default>'. O processamento Pause descritos acima é apenas um modo de alcançar isto. Uma alternativa seria o simples uso do TIB_Transaction.Refresh (true/false) que resulta na transação física ser parada e reiniciada.

Aplicações reais

Em uma aplicação real voce pode precisar levar em conta a possibilidade que um usuário deixará um dataset em um modo Edit (edit/insert). Em minha própria aplicação eu uso dois valores de timeout diferentes. O menor interromperá a transação se só está visualizando (PostPendingCount = 0). Eu uso um valor de timeout maior quando uma transação está editando (PostPendingCount> 0) e se este timeout é alcançado eu desfaço (rollback) as mudanças. (E tudo isso é protegido por um timer de atividade para assegurar que eu não tento interromper um usuário no meio da digitação das suas alterações.)

Texto original Working with Transactions:

Informação Técnica
Jason Wharton
Computer Programming Solutions
CPS - Mesa - AZ
http://www.ibobjects.com

Artigo Original: http://www.ibobjects.com **Jason Wharton** jwharton@ibobjects.com	
Tradução e adaptação: **Cláudio Sergio Gonçalves** claudio@clarosistemas.com.br	**Comunidade Firebird de Língua Portuguesa** **Visite a Comunidade em:** **http://www.comunidade-firebird.org**
A Comunidade Firebird de Língua Portuguesa foi autorizada pelo Autor do Original para elaborar esta tradução.	