Soluções em Programação de Computadores
PLANILHA DE INFORMAÇÕES TÉCNICAS

Trabalhando com Valores Monetários e Fracionários

Resumo:
Eu tenho trabalhado no aperfeiçoamento do código do IBO para suporte de valores do tipo numeric(x,y). Este processo resultou não apenas em valores monetários, mas também valores fracionários dentro do mecanismo do banco de dados Firebird/IB. O que se segue é uma compilação destas informações.

Geoff Worboys
Telesis Computing
(31-Dez-2001 - Atualizado: 02-Jan-2002)

Nota

A descrição a respeito do suporte do IBO a valores numéricos abaixo refere-se à versão IBO 4.3 e posteriores. Versões anteriores do IBO apresentaram algumas dificuldades com valores numéricos grandes.

Fundamentos

Os tipos de dados NUMERIC e DECIMAL do Firebird prevêem o armazenamento exato dos valores numéricos. As declarações são:

NUMERIC(*p*,*s*) e DECIMAL(*p*,*s*)

onde ***p*** é a precisão (a quantidade total de dígitos numéricos do valor) e ***s*** é a fração (o número de dígitos que estão a direita do ponto decimal).

No Firebird e Interbase® v6 o mecanismo para armazenar valores com exatidão é gravar todos os valores NUMERIC e DECIMAL como inteiros escalados. Valores do tipo ponto flutuante (como os DOUBLE PRECISION) não tem a garantia de armazenar os números exatamente como foram originalmente gravados e pior, eles acumulam erros de arredondamento em cálculos subseqüentes.

A diferença conceitual entre NUMERIC e DECIMAL é que NUMERIC vai gravar exatamente ***p*** dígitos enquanto DECIMAL vai gravar não menos que ***p*** dígitos.

A diferença atual entre os 2 tipos (nas versões avaliadas enquanto este artigo é escrito) é mínima. Portanto, deste ponto em diante eu vou discutir os valores do tipo NUMERIC e as mesmas regras serão aplicadas aos valores do tipo DECIMAL.

Como o Firebird/IB 6.x armazena seus valores numéricos fracionados

* Se a precisão (p) for entre 1 e 4, o servidor armazenará o valor como inteiro pequeno escalado (16bit).

* Se a precisão (p) for entre 5 e 9, o servidor armazenará o valor como inteiro escalado (32bit).

* Se a precisão (p) for entre 10 e 18, o servidor armazenará o valor como inteiro largo escalado (64bit).

InterBase 5 e anteriores

No Interbase v5 ou anterior, 15 é o valor máximo suportado para a precisão (***p***). Valores do tipo "Numeric" com precisão de 10 a 15 são armazenados como DOUBLE PRECISION. Se estas bases de dados forem atualizadas para Firebird ou IB6 ou FB1 utilizando a ferramenta de backup - o gbak - e depois restauradas, estes campos numéricos de alta-escala continuarão implementados como DOUBLE PRECISION. Embora eles continuem constando como NUMERIC(15,2) ou que foi definido anteriormente, o armazenamento continuará sendo como DOUBLE PRECISION.

Validação de valores do tipo NUMERIC

No Interbase v5 e anterior, o valor declarado da precisão não era gravado. Agora, no Firebird e Interbase 6, a precisão é gravada, mas não utilizada para a validação. Embora você tenha declarado a precisão para um campo/valor o servidor NÃO irá verificar se aquele valor se encaixa na especificação declarada. Será verificado somente para fins de estouro (*overflow)* do valor do inteiro subjacente.

Por exemplo, se você declarou um campo como

```
AFIELD NUMERIC(7,2),
```

O servidor aceitará o valor '1234567.89' apesar do fato deste valor ter 9 dígitos (incluindo as casas decimais).

Ele rejeitará o valor '123456789' porque após adicionar 2 casas decimais, o valor passa a ter 11 dígitos, o que não é suportado pelo tipo Inteiro de 32 bits, alocado para armazenar o campo AFIELD NUMERIC(7,2).

NOTA: Na implementação corrente do Firebird e do IB 6, o tipo DECIMAL(7,2) também rejeitará o valor '123456789', indiferentemente do fato que supostamente deveria armazenar pelo menos 7 dígitos.

Reforçando sua declaração junto ao servidor

Para fazer o servidor reforçar a declaração de uma determinada precisão, você precisa criar um domínio com uma "constraint" do tipo CHECK; por exemplo:

```
CREATE DOMAIN NUMERIC_7_2
AS NUMERIC(7,2)
CHECK( VALUE BETWEEN -99999.99 AND +99999.99 );
```

Considero muito importante fazer isso, caso contrário a sua base de dados poderá aceitar valores que podem posteriormente gerar exceções na sua aplicação cliente.

Tipos de Dados NUMERIC no cliente (IBO)

Se a fração de um NUMERIC é 0 (Zero) então ele pode ser lido diretamente como inteiro (conforme sua dimensão), usando as propriedades da TIB_Column - AsSmallint, AsInteger e AsInt64 (ou os equivalentes VCL do Delphi, caso esteja usando Tdataset e Tfield). Quando a fração é 0 (Zero) será criada automaticamente pelo IBO uma derivativa da TIB_Column, cuja propriedade Value se enquadrará de acordo com o tipo de Inteiro requerido.

Quando a fração do NUMERIC é diferente de 0 (Zero) fica mais difícil determinar a melhor maneira de gerenciar o valor no cliente. A dificuldade decorre do fato do Delphi não prover um tipo de dados nativo que seja exatamente compatível com as propriedades dos dados do tipo NUMERIC do Firebird.

Quando o IBO detecta um valor do tipo NUMERIC com a parte fracionária diferente de zero ele cria uma derivativa da TIB_Column, chamada TIB_ColumnNumeric, cuja propriedade Value é do tipo Extended. Esta classe também introduz um suporte especial para as propriedades AsString, AsInteger, AsCurrency e AsExtended para se aproximar o mais possível das propriedades do dado NUMERIC. Para a interação com controles do usuário (TIB_Edit etc) o IBO usa as regras descritas a abaixo. O TIB_Column prove uma propriedade do tipo Boolean, IsCurrencyDataType, que pode ser usada para detectar quando o processamento do IBO utilizará o tipo Currency do Delphi (ao invés dos valores de ponto flutuante).

O IBO arredondará automaticamente qualquer valor de ponto flutuante assinalado para campos do tipo NUMERIC, de acordo com a fração do campo.

Suporte a NUMERIC pelos componentes nativos do IBO.

O processamento padrão para os componentes nativos do IBO (TIB_Dataset, TIB_Query, TIB_Cursor etc) utiliza as seguintes regras...

Se a Fração for 1:4 a coluna implementará suas propriedades AsString e AsInteger utilizando o tipo Currency do Delphi (utilizando a propriedade AsCurrency). Nesta instância a propriedade IsCurrencyDataType da coluna retornará Verdadeiro.

Se a Fração for maior que 4 a coluna implementará suas propriedades AsString and AsInteger utilizando o tipo Extended do Delphi (utilizando a propriedade AsExtended). Nesta instância a propriedade IsCurrencyDataType da coluna retornará Falso. É necessário utilizar o tipo Extended para Fração maior que 4 porque o tipo Currency suporta somente 4 casas decimais, mas fique atento pois tipo Extended não é inteiro escalado e você poderá obter arredondamentos inconsistentes nas faixas limite.

Campos NUMERIC suportados pelos componentes IBO baseados em TDataset

O processamento padrão para componentes IBO baseados em Tdataset (TIBODataset, TIBOQuery, TIBOTable etc) utiliza as seguintes regras:

* Se você está usando Delphi v5 ou posterior e a Fração é 1..4 o Dataset criará uma instância TIBOBCDField, que utiliza o tipo Currency do Delphi (e lê/escreve os dados para a referente TIB_Column, usando a propriedade AsCurrency).

* Se você está usando Delphi v4 ou anterior e a Fração é maior que 4 o dataset criará uma instância TIBOFloatField, que utiliza o tipo Double do Delphi (e lê/escreve os dados para a referente TIB_Column usando a propriedade AsExtended. É necessário utilizar Extended para Fração maior que 4 porque o tipo Currency suporta no máximo 4 casas decimais. Entretando, fique atento pois o tipo Double não é um Scaled Integer e você poderá obter erros de arredondamento nos limites da faixa.

Attributo de Columa NOBCD

Você pode alterar o processamento padrão acima descrito usando o atributo de coluna NOBCD. Se você configurar um atributo de coluna como:

MY_NUMERIC_FIELD=NOBCD

O IBO utilizará a implementação de ponto flutuante para o MY_NUMERIC_FIELD se a fração for maior que zero. Ele não tentará utilizar o tipo Currency do Delphi. Assim, o atributo NOBCD fará com que os componentes nativos IBO revertam o uso de AsExtended para implementar os Numéricos Escalados e fará com que os componentes Tdataset revertam o uso da classe TIBOFloatField para instanciar o objecto do campo persistente.

NOTA : Se você está usando objectos de campo persistentes com componentes Tdataset e aplica ou remove o atributo NOBCD a um campo, a classe Tfield não será afetada automaticamente. Você precisa editar o objeto campo com o Fields Editor e manualmente alterar a classe Tfields de TIBOBCDField para TIBOFloatField ou vice versa.

Atributo Coluna BCD

Este atributo é utilizado somente pelos componentes Tdataset. Foi projetado de forma a permitir que se force qualquer campo do tipo Ponto Flutuante a usar o TIBOBCDField (que implementa o processamento usado o tipo Currency do Delphi). Se você configura um atributo de coluna tal como:

MY_FLOAT_FIELD=BCD

o IBO utilizará a classe TIBOBCDField para tal campo.

Favor consultar a NOTA do tópico anterior.

Números IBO e Variants

Devido aos problemas e inconsistências com o tipo Variant no Delphi, o IBO converte todo valor NUMERIC diferente de zero (e todos os valores inteiros de 64 bits) para o tipo String quando o atribui a uma Variant. Como conseqüência, o método TIB_Column AsVariant retornará o tipo de Variant varString de NUMERIC e campos inteiro longos. Esta parece ser a única maneira confiável de se reter o valor exato dos valores NUMERIC e inteiros longos quando usando Variants.

Quando você assinala uma Variant para o método TIB_Column AsVariant ele aceitará tipos variant numéricos (varInteger etc) mas a atribuição é suportada através da propriedade AsExtended da coluna. Isto introduz a possibilidade de problemas de arredondamento nos limites de faixa de valores grandes.

AsVariant aceitará também (é claro) tipos varString, que serão convertidos para o tipo de dados subjacente, pelo método AsString, assim minimizando os problemas com arredondamento nas faixas limite.

Propriedades Value do TIB_Columns

A maior parte dos TIB_Columns utiliza funções virtuais, provendo suporte consistente através de todas as subclasses derivativas dos TIB_Column. O que significa que o código abaixo é consistente e funcionará conforme o esperado...

```
var
   tmpStr: string;
begin
   tmpStr := IB_Query1.FieldByName( 'MYFIELD' ).AsString;
```

O valor String de 'MYFIELD' sera atribuído a tmpStr, obtido pelo código daquela classe derivativa particular de TIB_Column, o tratamento especial AsString utilizado pela classe TIB_ColumnNumeric, por exemplo.

Isto **NÃO** é válido para a propriedade **Value**!

A propriedade Value não pode utilizar métodos virtuais porque o tipo de dado depende da classe particular TIB_Column. O código a seguir pode não resultar no efeito esperado...

```
var
   tmpExt: extended;
begin
   tmpExt := IB_Query1.FieldByName( 'MY_NUMERIC_FIELD' ).Value;
```

Você sabe que o campo 'MY_NUMERIC_FIELD' terá sido criado como TIB_ColumnNumeric e sabe também que a propriedade Value daquela classe retorna o tipo Extended de dados – portanto você pode esperar que o código acima retorne um valor Extended diretamente. De fato, ele retornará um Variant do tipo varString (veja acima) e, se a Fração era 1..4. tal string terá sido gerada usando o método AsCurrency.

Isto acontece porque a referência à coluna retornada pelo FieldByName() é uma classe genérica TIB_Column. A propriedade Value da classe TIB_Column retorna uma Variant (utilizando a propriedade AsVariant) e devido ao fato dos métodos de acesso a Value não serem virtuais, o resultado decorre diretamente da classe TIB_Column e não de uma derivativa especial.

Um dos códigos a seguir é mais apropriado...

```
var
   tmpExt: extended;
begin
   tmpExt := IB_Query1.FieldByName( 'MY_NUMERIC_FIELD' ).AsExtended;

var
   tmpExt: extended;
begin
   tmpExt := (IB_Query1.FieldByName( 'MY_NUMERIC_FIELD' ) as
TIB_ColumnNumeric).Value;
```

Propriedades Value e AsExtended de TIB_ColumnNumeric

Se você usa a propriedade Value (veja notas sobre as propriedades Value acima) ou a propriedade AsExtended da instância TIB_ColumnNumeric, você está trabalhando com tipos de dados Extended, um valor de ponto flutuante de 80 bits. Valores Extended possuem mais significado que Double, o que significa que trabalhar com dados Extended é mais confiável, especialmente quando trabalhando com colunas NUMERIC com precisão de 9 ou menos. Mais ainda assim existem algumas possíveis inconsistências a considerar quando se usa campos NUMERIC.

Os componentes Tdataset não suportam campos do tipo Extended. Os campos TfloatField (posteriormente TIBOFloatField) utilizam o tipo Double do Delphi (que é apenas um ponto flutuante de 64 bits). Por este motivo valores mostrados pelos componentes nativos utilizando AsExtended podem não ser exatamente os mesmos visualizados pelos componentes Tdataset.

Quando se trabalha com precisão de 10 dígitos ou maior, é possível a ocorrência de dificuldades de arredondamento nos limites da faixa de inteiros 64 bits, porque tais limites são muito similares ao Extended. Você precisa estar atento, pois se você precisa manipular grandes valores NUMERIC utilizando funções com parâmetros ou valores de retorno do tipo Extended, os resultados podem não ser os previstos.

Se possível, quanto utilizar valores NUMERIC com Fração de 1..4 utilize funções que usem o tipo Currency do Delphi de forma a minimizar inconsistências devido a arredondamento.

O Tipo Currency do Delphi

O método AsCurrency do TIB_ColumnNumeric utiliza o tipo Currency do Delphi para acessar o valor da coluna. Currency é um tipo inteiro escalado de 64 bits nativo para Delphi/BCB (versões 4 ou posteriores). Seu propósito é minimizar erros de arredondamento. Entretanto, como o tipo Currency não é exatamente o mesmo que o tipo NUMERIC do Firebird, você necessita entender as diferenças se você pretende utilizar o tipo Currency.

O tipo Currency é um inteiro de 64 bits implicitamente fracionado (pelo compilador) para 4 casas decimais. O fracionamento é fixo, portanto não pode ser igualada à Fração do valor NUMERIC que você definiu do banco de dados. O mais próximo ocorre entre o tipo Currency do Delphi e o NUMERIC(18,4) do Firebird. Em outras palavras...

Currency supporta:

14 digitos** à esquerda do separador decimal
4 digitos à direita do separador decimal

Conversões de bases de dados, de NUMERIC para Currency causarão uma exceção – dados existentes excedem os limites fixados**.

** Atualmente o tipo Currency suporta até 15 dígitos a esquerda do separador decimal, somente dentro da faixa:
-922337203685477.5808..922337203685477.5807
portanto, nos termos do tipo NUMERIC declarados na base de dados, o máximo que se pode contar é com 14 dígitos.

Frequentemente pessoas implementam bases de dados utilizando definições de campo do tipo...

AMONEY NUMERIC(18,2)

porque eles só precisam de 2 casas decimais.

A armadilha em se usar tal especificação com o tipo Currency do Delphi é que o NUMERIC(18,2) é capaz de suportar 16 dígitos a esquerda do ponto decimal, mas um valor como este superaria os limites do tipo Currency e causar uma exceção.

Evitando o Estouro (Overflow) com o tipo Currency

Se você deseja 2 casas decimais e está pensando em usar o tipo Currency do Delphi, você precisa configurar um domínio do tipo...

```
CREATE DOMAIN CURRENCY_D
AS NUMERIC(16,2)
CHECK( VALUE BETWEEN -99999999999999.99 AND +99999999999999.99 );
```

Este tipo é apropriado para suportar dados providos pelo tipo Currency e implementa uma checagem para se assegurar que a base de dados não aceitará valores com mais de 16 dígitos de precisão.

Você também pode evitar a limitação definindo o atributo de coluna NOBCD para o campo ou domínio. Entretanto isso reverterá o processamento cliente utilizando o tipo Extended (ou Double se utilizando componentes Tdataset), com os conseqüentes problemas de arredondamento nos valores próximos aos limites da faixa – veja a seguir.

Cálculos com Currency e NUMERIC

Mesmo após ter declarado um domínio apropriado no servidor, ainda existem algumas diferenças entre NUMERIC e Currency...

* Como o tipo NUMERIC é um verdadeiro inteiro escalado, a divisão será truncada (não arredondada). O tipo Currency é implementado parcialmente como valor real/ponto flutuante e o compilador implementa arredondamento para o valor mais próximo.

* Por causa da implementação da parte real do tipo Currency você poderá ainda ter erros de arredondamento nos extremos da faixa Currency. O código a seguir foi rodado em Delphi 5:

```
procedure TForm1.TestCalcs(Sender: TObject);
var
  c, m: currency;
begin
  m := 100;
  c := 12345678901234.0000;
  c := c / m;
  c := c * m;
  Label1.Caption := FormatCurr('#,#0.0000', c);
end;
```

O resultado apresentado no Label1.Caption foi:
12,345,678,901,233.9984

Isso foi causado pela declaração como Currency. Se o código fosse implementado simplesmente como...

```
c := 12345678901234.0000;
c := c / 100;
c := c * 100;
```

o problema de arredondamento não apareceria.

O mecanismo exato que produz estes resultados é parte das “ mágicas do compilador “ e eu não pretendo entende-la. Entretanto é importante saber que estes problemas existirão se você estiver manipulando grandes valores Currency em seu código.

NOTA:: Alterações feitas no IBO (v. 4.3 e posteriores) evitarão este problema ao alterar a Fração dos valores Currency de forma a se igualar a declaração do NUMERIC, transformando primeiro para Int64 e depois fazendo os devidos ajustes na Fração. Para consistência com o tipo Currency do Delphi a implementação do IBO continuará a arredondar quando necessário, mas o arredondamento do IBO é para o valor mais próximo e não para o valor par mais próximo.

Stored Procedures e Triggers

NO curso da discussão da manipulação de grandes valores NUMERIC/Currency é apropriado destacar o potencial de problemas com stored procedures e triggers.

Um código como...

```
  DECLARE VARIABLE C NUMERIC(18,4);
  DECLARE VARIABLE M NUMERIC(18,4);
BEGIN
  C = 12345678901234.0000;
  M = 100;
  C = C / M;
  C = C * M;
```

Causará um estouro - “arithmetic overflow”. Isto porque o servidor seguirá estritamente as regras matemáticas da divisão e tentará colocar o resultado de C/M numa variável interna declarada como NUMERIC(18,8). Claro, isto tem apenas 10 dígitos a esquerda do ponto decimal porque 18 dígitos é o total máximo para o dado tipo.

O código abaixo evita este problema:

```
  DECLARE VARIABLE C NUMERIC(18,4);
BEGIN
  C = 12345678901234.0000;
  C = C / 100;
  C = C * 100;
```

porque 100 tem Fração 0 (zero) portanto a variável interna se torna NUMERIC(18,4) o que é aceitável.

Pode ser útil NÃO declarar a máxima resolução disponível do tipo, o que deixará espaço à esquerda para os cálculos internos.

Por exemplo, o código abaixo está OK:

```
  DECLARE VARIABLE C NUMERIC(16,2);
  DECLARE VARIABLE M NUMERIC(16,2);
BEGIN
  C = 12345678901234.0000;
  M = 100;
  C = C / M;
  C = C * M;
```

CUIDADO com cálculos compostos usando NUMERIC. Por exemplo:

```
  C = (C / M) * M;
```

Haverá uma tentativa se de usar valores do tipo NUMERIC(18,6) para os cálculos internos e o resultado será um Estouro (overflow). Se você necessita cálculos compostos, você provavelmente necessitará transformar cada parte do resultado para se manter dentro da faixa...

```
C = CAST( (C/M) AS NUMERIC(16,2)) * M;
```

Alternativas

Não é essencial usar tipos NUMERIC (apesar de que quando a Fração é Zero ou os valores forem pequenos, você está razoavelmente a salvo).

Em vez disso, você pode implementar seus valores monetários usando precisão dupla. Tem uma significância de apenas 15, mas é em geral suficiente.

Entretanto se você usar precisão dupla você provavelmente necessitará implementar uma biblioteca de funções UDF, contendo funções de arredondamento, de forma que você pode configurar suas triggers e stored procedures para forçar a gravação de valores apropriadamente arredondados.

Como sempre, se você tiver comentários ou questões sinta-se a vontade para contatar a lista pública de discussão do IBO.

Jason Wharton
http://www.ibobjects.com
mailto:jwharton@ibobjects.com

Artigo Original: http://www.ibobjects.com **Geoff Worboys**	
Tradução e adaptação: **Luiz Carlos Arakaki** larakaki@usp.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.