# Replicação de Dados no Interbase

**Por Matt Hopkins, Dunstan Thomas(UK) LTD.Borland Developers Conferece 1998 - nessa época ainda não existia o componente IBReplicator**
Origem: http://www.ibphoenix.com/ibp_howto10.html
Tradução : Marcilio Soares
Revisão/adaptação do texto : Carlos Henrique Cantu - http://www.interbase-br.com

Este documento descreverá a concepção básica para replicação de dados e como eles podem ser implementados de maneira relativamente simples usando semente recursos do próprio Interbase.

## O que é Replicação de Dados?

É a cópia de informações de um ou mais Banco de dados para outro semelhante, depois das informações estarem consistentes.

Há dois tipos básicos de replicação – SÍNCRONA E ASSÍNCRONA

### Replicação Síncrona:

Todas as cópias ou replicações de dados serão feitas no instante da sincronização e consistência. Se alguma cópia do banco é alterada, essa alteração será imediatamente aplicada a todos os outros bancos dentro da transação. A Replicação Síncrona é apropriada em aplicações comerciais onde a consistência exata das informações é de extrema importância.

### Replicação Assíncrona:

Armazena e faz a replicação. A cópia de dados fica fora de sincronia entre os BDs. Se um BD é alterado, a alteração será propagada e aplicada para outro BD num segundo passo, dentro de uma transação separada sendo que esta poderá ocorrer segundos, minutos, horas ou até dias depois. A cópia poderá ficar temporariamente fora de sincronia, mas quando a sincronização ocorrer os dados convergirão para todos os locais especificados.

Este documento trata exclusivamente sobre replicação de dados assíncrono.

## Quais componentes são necessários para a replicação de dados?

### Capturando as alterações no banco de dados de origem.

Toda estratégia de replicação de dados requer um método para capturar as alterações para um banco de dados de replicação. Há dois mecanismos principais – Logs de Transação e Triggers.

*Logs de transação* abordam a utilização de uma técnica chamada "log Sniffing" a qual replica uma transação sinalizada para replicação para um plataforma de transmissão. Esta técnica causa um impacto menor no servidor de Banco de Dados porque requer menor CPU quando da leitura de "Log" na memória e também na gravação para o disco. Este método de replicação é utilizado também por outros fabricantes de Bancos como Sybase, Informix e MS SQL/Server.

A Segunda técnica usa triggers no Banco de dados para propagar as alterações quando elas ocorrem. Como a Linguagem procedural de Banco de dados pode ser utilizada nesse método, ela provê maior flexibilidade a medida que os dados são replicados. Esta implementação de replicação é utilizada pelo Ingres e Oracle, também será a técnica que nós utilizaremos ao trabalharmos com replicação com Interbase.

### Transmitindo alterações de um de Banco de Dados para o(s) Banco de dados Destino

Para a transmição das alterações para o BD destino é requerido um Software que lê as alterações e as transmite ao BD Destino situado em algum lugar da rede e grava as alterações.

Este Software é conhecido como um gerenciador de replicação e também controla erros e conflito de Updates.

## Como posso fazer Replicação de Dados com os componentes do Interbase?

### Registrando a alteração dos dados

Para criar nosso processo de replicação nós precisaremos criar dois exemplos de Banco de Dados com tabelas idênticas em cada um dos Bancos.

```
CREATE DATABASE "source.gdb" PAGE_SIZE 1024

CREATE TABLE TEAMS (
TEAM_ID INTEGER NOT NULL,
TEAM_NAME VARCHAR(100),
TEAM_MGR VARCHAR(40),
PRIMARY KEY (TEAM_ID));

CREATE TABLE CHANGES (
CHANGE_ID INTEGER NOT NULL,
CHANGE_TYPE CHAR(1) NOT NULL,
SOURCE_KEY_FIELD VARCHAR(31) NOT NULL,
SOURCE_KEY INTEGER NOT NULL,
```

```
SOURCE_TABLE VARCHAR(31),
USERNAME VARCHAR(31),
PRIMARY KEY (CHANGE_ID));

CREATE GENERATOR NEW_TEAM_ID;

CREATE GENERATOR NEW_CHANGE_ID;
```

Agora criaremos um TRIGGER CHANGES na tabela para obtermos um único valor do GENERATOR

```
CREATE TRIGGER CHANGES_NEWID FOR CHANGES
BEFORE INSERT
AS
BEGIN
NEW.CHANGE_ID = GEN_ID(NEW_CHANGE_ID,1);
END
```

NOTA: Veja que não adicionaremos um trigger para gerar uma única ID na tabela TEAMS. Discutiremos este assunto na próxima sessão.

Agora nós precisamos criar um mecanismo para registrar todas as alterações de nossa tabela.

Como mencionado anteriormente, nós utilizaremos triggers para registrar as alterações para nossa tabela de Replicação. O benefício é que os triggers estão disponíveis, são fáceis de usar, e diferente dos logs de transação, eles possibilitam o uso de uma lógica de programação utilizando a linguagem procedural do Interbase.

Há três tipos de alteração que poderão ocorrer na tabela – Insert, Update e Delete. Nós precisaremos de três triggers para cada uma das tabelas replicadas.

```
/* After an Insert */

CREATE TRIGGER TEAMS_REPL_INSERT FOR TEAMS
AFTER INSERT
AS
BEGIN
INSERT INTO CHANGES
(CHANGE_TYPE, SOURCE_KEY_FIELD,
SOURCE_KEY, SOURCE_TABLE,USERNAME)
VALUES
("I","TEAM_ID",NEW.TEAM_ID,"TEAMS",USER);
END;

/* After an Update */

CREATE TRIGGER TEAMS_REPL_UPDATE FOR TEAMS
AFTER UPDATE
AS
BEGIN
```

```
INSERT INTO CHANGES
(CHANGE_TYPE, SOURCE_KEY_FIELD,
SOURCE_KEY, SOURCE_TABLE,USERNAME)
VALUES
("U","TEAM_ID",NEW.TEAM_ID,"TEAMS",USER);
END;

/* After a Delete */

CREATE TRIGGER TEAMS_REPL_DELETE FOR TEAMS
AFTER DELETE
AS
BEGIN
INSERT INTO CHANGES
(CHANGE_TYPE, SOURCE_KEY_FIELD,
SOURCE_KEY, SOURCE_TABLE,USERNAME)
VALUES
("D","TEAM_ID",OLD.TEAM_ID,"TEAMS",USER);
END;
```

Isto é tudo que se precisa para registrarmos as alterações de dados para uma replicação no Interbase.

Note que para manter a simplicidade nós requisitamos que todas as tabelas replicadas tanha uma Chave Inteira (INTEGER KEY) única e que este é o único elemento de dados que será gravado na tabela CHANGES. Nós poderíamos armazenar todos os valores dos campos que foram alterados, mas como nós vamos replicar um snapshot dos dados armazenando apenas a chave e a ação, nós podemos recuperar os valores atuais em cada tabela pelo mecanismo de replicação.

Mais um passo é necessário antes de irmos para o mecanismo de replicação. Nós precisaremos criar um Banco de Dados de destino com a mesma Metadata. Isto poderá ser feito fazendo um backup do BD e restaurando-o com o nome de "DESTINATION.GDB".

## Replicando os dados alterados

Agora nós vamos criar um Servidor de Replicação usando Delphi.

A idéia é permitir ao usuário pré-defina um tempo determinado para a replicação ou ele poderá manualmente solicitar a replicação com um botão "Replicar Agora". Um registro de todas as atividades será mostrada num Memo e algumas exceções que ocorram serão exibidas em um Memo de erros.

O Banco Replicado e o Banco de Destino da Replicação serão passados em linha de comando através de parâmetros desta forma:

```
C:\REPLD SOURCEDB TARGETDB
```

Para nos assegurar de que o nosso projeto sempre terá dois parâmetros nós faremos assim:

```
if (ParamStr(1) = '') or (ParamStr(2) = '') then
MessageDlg('Usage: "REPLD.EXE SOURCE_ALIAS TARGET_ALIAS"',mtError,[mbOK],0)
else
begin
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
end;
```

Toda replicação será controlada por uma Procedure de replicação desta forma:

```
procedure TForm1.Replicate;

var

qryChangeList : TQuery;
strChangeType : String;

begin
qryChangeList := TQuery.create(Application);
try
with qryChangeList do
begin
DatabaseName := 'SourceDB';
sql.add('select * from changes');
open;
while not eof do
begin
if (fieldByName('CHANGE_TYPE').asString = 'I') then
begin
strChangeType := 'Insert';

ReplicateInsert(
fieldByName('SOURCE_TABLE').asString,fieldByName(
'SOURCE_KEY_FIELD').asString,fieldByName('SOURCE_KEY').asInteger)
end
else if (fieldByName('CHANGE_TYPE').asString = 'U') then
begin
strChangeType := 'Update';

ReplicateUpdate(
fieldByName('SOURCE_TABLE').asString,fieldByName(
'SOURCE_KEY_FIELD').asString,fieldByName('SOURCE_KEY').asInteger)
end
else if (fieldByName('CHANGE_TYPE').asString = 'D') then
begin
strChangeType := 'Delete';

ReplicateDelete(
fieldByName('SOURCE_TABLE').asString,fieldByName(
```

```
'SOURCE_KEY_FIELD').asString,fieldByName('SOURCE_KEY').asInteger)
end;

memoLog.lines.add( DateTimeToStr(Now)+' '+strChangeType+
' '+fieldByName('SOURCE_KEY').asString+
' on '+fieldByName('SOURCE_TABLE').asString);
DeleteFromChanges(fieldByName('CHANGE_ID').asInteger);
next;

end;
close;
end;

finally
qryChangeList.free;
end;
end;
```

E a cada ação(INSERT, DELETE, UPDATE) será chamada uma Procedure individual, como se segue:

**Replicate Inserts:**

```
procedure TForm1.ReplicateInsert(
const strTableName, strKeyField: String; const iKey: Integer);

var

qrySource, qryTarget : TQuery;
i : SmallInt;

begin
qrySource := TQuery.create(Application);
try
with qrySource do
begin
DatabaseName := 'SourceDB';
with sql do
begin
add('select * from '+strTableName);
add(' where ('+strKeyField+' = '+IntToStr(iKey)+');');
end;
try
open;
while not eof do
begin
qryTarget := TQuery.create(Application);
try
with qryTarget do
begin
DatabaseName := 'TargetDB';
with sql do
```

```
begin
add('insert into '+strTableName);
add('(');
for i := 0 to qrySource.fieldCount-1 do
begin
if (i < qrySource.fieldCount-1) then
add(' '+qrySource.Fields[i].FieldName+',')
else
add(' '+qrySource.Fields[i].FieldName)
end;
add(')');
add('values');
add('(');
for i := 0 to qrySource.fieldCount-1 do
begin
if (i < qrySource.fieldCount-1) then
add(' :'+qrySource.Fields[i].FieldName+',')
else
add(' :'+qrySource.Fields[i].FieldName)
end;
add(')');
end;
prepare;
for i := 0 to qrySource.fieldCount-1 do
begin
qryTarget.Params[i].asString := qrySource.Fields[i].asString;
end;
execSQL;
end;

finally
qryTarget.free;
end;
next;
end;
close;

except
on e: Exception do
memoErrors.lines.add(DateTimeToStr(Now)+' '+e.message);
end;
end;

finally
qrySource.free;
end;
end;
```

**Replicate Upates:**

```
procedure TForm1.ReplicateUpdate(
const strTableName, strKeyField: String; const iKey: Integer);
```

```
var

qrySource, qryTarget : TQuery;
i : SmallInt;

begin
qrySource := TQuery.create(Application);
try
with qrySource do
begin
DatabaseName := 'SourceDB';
with sql do
begin
add('select * from '+strTableName);
add(' where ('+strKeyField+' = '+IntToStr(iKey)+');');
end;
try
open;
while not eof do
begin
qryTarget := TQuery.create(Application);
try
with qryTarget do
begin
DatabaseName := 'TargetDB';
with sql do
begin
add('update '+strTableName);
add('set');
for i := 0 to qrySource.fieldCount-1 do
begin
if (i < qrySource.fieldCount-1) then
add(' '+qrySource.Fields[i].FieldName+
' = :'+qrySource.Fields[i].FieldName+',')
else
add(' '+qrySource.Fields[i].FieldName+
' = :'+qrySource.Fields[i].FieldName)
end;
add(' where ('+strKeyField+' = '+IntToStr(iKey)+');');
end;
prepare;
for i := 0 to qrySource.fieldCount-1 do
begin
qryTarget.Params[i].asString := qrySource.Fields[i].asString;
end;
execSQL;
end;
finally
qryTarget.free;
end;
next;
end;
```

```
close;
except
on e: Exception do
memoErrors.lines.add(DateTimeToStr(Now)+' '+e.message);
end;
end;
finally
qrySource.free;
end;
end;
```

**Replicate Deletes:**

```
procedure TForm1.ReplicateDelete(
const strTableName, strKeyField: String; const iKey: Integer);

var

qrySource : TQuery;
i : SmallInt;

begin
qrySource := TQuery.create(Application);
try
with qrySource do
begin
DatabaseName := 'TargetDB';
sql.add('delete from '+strTableName);
sql.add(' where ('+strKeyField+' = '+IntToStr(iKey)+')');
execSQL;
try
execSQL;
except
on e: Exception do
memoErrors.lines.add(DateTimeToStr(Now)+' '+e.message);
end;
end;
finally
qrySource.free;
end;
end;
```

**Nota: Este exemplo não suporta campos do tipo BLOB. Um código adicional será necessário para seu uso, mas isso é totalmente possível.**

Agora atribua o valor do SpinBox ao Timer . Desta forma você poderá ajustar o cliclo da replicação:

```
procedure TForm1.SpinEdit1Change(Sender: TObject);
```

```
begin
Timer1.Interval := (SpinEdit1.Value * 1000);
end;
```

Execute-o e enquanto REPLD estiver executando, faça alguma alteração na tabela TEAMS no Banco de dados Origem tendo certeza de adicionar uma chave única e verifique o banco de dados de destino (Depois de um ciclo de replicação) para ver as alterações replicadas.

## Replicação Bi-Direcional

Cheque a tabela CHANGES no BD destino e voce verá que as alterações replicadas do BD origem também aparecerão nessa tabela.

O problema é que o mecanismo utilizado para logar as alterações não está distinguindo as alterações feitas pelo usuário final das alterações feitas pelo mecanismo de replicação. Se nós iniciarmos uma replicação bi-direcional utilizando nosso aplicativo REPLD sem especificar um "usuário replicador", correremos o risco de entrar em um loop infinito.

Portanto, nós precisaremos definir um "usuário replicador" – para esse exemplo o chamaremos de "REPLICATE" – em todos os servidores Interbase que estivermos usando, e não efetuaremos o log das alterações efetuadas por esse "usuário". Isso significa que precisaremos alterar os triggers para acada tabela replicada da seguinte forma :

```
/* After an Insert */

ALTER TRIGGER TEAMS_REPL_INSERT FOR TEAMS
AFTER INSERT
AS
BEGIN
IF (USER <> "REPLICATE") THEN
INSERT INTO CHANGES
(CHANGE_TYPE, SOURCE_KEY_FIELD,
SOURCE_KEY, SOURCE_TABLE,USERNAME)
VALUES
("I","TEAM_ID",NEW.TEAM_ID,"TEAMS",USER);
END;

/* After an Update */

ALTER TRIGGER TEAMS_REPL_UPDATE FOR TEAMS
AFTER UPDATE
AS
BEGIN
IF (USER <> "REPLICATE") THEN
INSERT INTO CHANGES
(CHANGE_TYPE, SOURCE_KEY_FIELD,
SOURCE_KEY, SOURCE_TABLE,USERNAME)
VALUES
("U","TEAM_ID",NEW.TEAM_ID,"TEAMS",USER);
```

```
END;

/* After a Delete */

ALTER TRIGGER TEAMS_REPL_DELETE FOR TEAMS
AFTER DELETE
AS
BEGIN
IF (USER <> "REPLICATE") THEN
INSERT INTO CHANGES
(CHANGE_TYPE, SOURCE_KEY_FIELD,
SOURCE_KEY, SOURCE_TABLE,USERNAME)
VALUES
("D","TEAM_ID",OLD.TEAM_ID,"TEAMS",USER);
END;
```

Agora nós executaremos duas versões do REPLD:

```
C:\REPLD SOURCEDB TARGETDB

C:\REPLD TARGETDB SOURCEDB
```

Voce deve notar que quando voce sai de um modelo de replicação mestre-escravo para um modelo ponto-a-ponto, diversas complexidades aparecem e devem ser consideradas antes de implementar a replicação. Essas complexidades e como trabalhar com elas serão discutidas na próxima seção.

## Replicando uma alteração quando ela ocorre (Near-Synchoronous Replication)

Se desejarmos replicar as alterações de uma tabela sem depender do timer, nós precisamos utilizar o mecanismo de Eventos do Interbase. Ele requer a criação de dois passos: Gerar um evento e responder à um evento.

Para gerar o evento, precisaremos adicionar um novo TRIGGER CHANGES na tabela TEAMS:

```
CREATE TRIGGER CHANGES_ALERT FOR TEAMS
AFTER INSERT
AS
BEGIN
POST_EVENT "NEW_CHANGE";
END;
```

Para responder ao evento, precisaremos adicionar um componente do Delphi chamado IBEventAlerter em nossa aplicação REPLD, registrar nosso interesse no evento "NEW_CHANGE" adicionando-o à propriedade Events e então linkar o evento OnEventAlert à nossa procedure de replicação como se segue:

```
procedure TForm1.IBEventAlerter1EventAlert(Sender: TObject;
```

```
EventName: string; EventCount: Longint; var CancelAlerts: Boolean);
begin
Replicate;
end;
```

Teste isso.

## Replicação em tabelas com chaves compostas

Para simplicidade, no exemplo anterior nós replicamos tabelas simples com chaves inteiras. A mesma técnica, um pouco modificada, pode ser usada para replicar tabelas contendo chaves compostas.

A principal diferença entre manusear chaves compostas e simples é no método que as alterações são logadas. Ao invés de uma simples tabela CHANGES, no caso de chaves compostas é necessário uma tabela de "log" para cada tabela base que será replicada.

## Replicação cruzada em WAN com RAS(Remote Access Services)

O Serviço RAS do Windows NT e Dial-up Netwroking do Windows 9x permite acesso via modem aos serviços que só estariam disponíveis em uma rede. Isto significa que se precisássemos replicar informações em sites remotos através do nosso mecanismo de replicação, ele necessitará suportar o RAS.

A maneira mais fácil de implementar no Delphi o RAS é usar componentes. Há vários disponíveis em forma de shareware ou freeware na internet. Para esse exemplo nós usaremos o TRas de Daniel Polistchuck que está disponível no site www.delphi32.com.

Como toda a complexidade do RAS está sendo trabalhada pelo componente TRAS, para acrescentaro RAS à nossa replicação necessitará apenas algumas linhas de código.

Primeiramente nós precisaremos modificar a procedure Replicate de forma que ocorrerá uma conexão via RAS sempre que uma alteração deverá ser replicada.

```
procedure TForm1.Replicate;

begin
with qryChangeList do
begin
DatabaseName := 'SourceDB';
sql.add('select * from changes');
if active then close;
open;
if (recordCount > 0) then
begin
memoLog.lines.add(DateTimeToStr(Now)+' Dialing DT');
RAS1.EntryName := 'DT';
RAS1.Connect;
end;
```

```
end;
end;
```

Como você pode ver, nós adicionamos um teste para verificar se há registros à serem enviados e, caso seja verdadeiro é criada uma conexão RAS chamada "DT".

Nos também movemos o resto do que previamente existia nesta procedure para o evento "OnConnect" do componente RAS e fizemos a query de alteração (QryChangeList) global para a form :

```
procedure TForm1.RAS1Connect(Sender: TObject);

var

strChangeType : String;

begin
with qryChangeList do
begin
memoLog.lines.add(DateTimeToStr(Now)+' Connected to DT');
try
while not eof do
begin

if (fieldByName('CHANGE_TYPE').asString = 'I') then
begin
strChangeType := 'Insert';

ReplicateInsert(
fieldByName('SOURCE_TABLE').asString,fieldByName(
'SOURCE_KEY_FIELD').asString,fieldByName('SOURCE_KEY').asInteger)
end

else if (fieldByName('CHANGE_TYPE').asString = 'U') then
begin
strChangeType := 'Update';

ReplicateUpdate(
fieldByName('SOURCE_TABLE').asString,fieldByName(
'SOURCE_KEY_FIELD').asString,fieldByName('SOURCE_KEY').asInteger)
end

else if (fieldByName('CHANGE_TYPE').asString = 'D') then
begin
strChangeType := 'Delete';

ReplicateDelete(
fieldByName('SOURCE_TABLE').asString,fieldByName(
'SOURCE_KEY_FIELD').asString,fieldByName('SOURCE_KEY').asInteger)
end;
```

```
memoLog.lines.add( DateTimeToStr(Now)+' '+strChangeType+' '+
fieldByName('SOURCE_KEY').asString+
' on '+fieldByName('SOURCE_TABLE').asString);
DeleteFromChanges(fieldByName('CHANGE_ID').asInteger);
next;
end;
close;
finally
RAS1.Disconnect;
memoLog.lines.add(DateTimeToStr(Now)+' Disconnected from DT');
end;
end;
end;
```

Isto é tudo que se precisa para fazer uma conexão remota e replicar as alterações e fazer a desconexão.

# Isso é simples ?

Não. Com a replicação podem aparecer diversos problemas e por isso é necessário um bom design e planejamento prévio e às vezes um pouco de programação. Abaixo estão algumas orientações:

## *Conflitos de Update*

A segurança para replicação Assíncrona é crítica para quase todas as aplicações. Porém, o que aconteceria se o mesmo elemento de dados (por exemplo, a mesma coluna de um mesmo registro) for atualizada em dois locais diferentes ao mesmo tempo, ou mais precisamente no mesmo intervalo de replicação?

Isto é conhecido como conflito de Update. Sendo assim quanto menor for o tempo de atualização da réplica (ou seja, menor o intervalo(ciclo) de atualização) menor será a possibilidade de ocorrer conflitos de atualização.

Alternativamente, os conflitos de atualizações podem ser evitados limitando o "ownership" ou o direito para atualizar um elemento de dado para um determinado local. Na realidade, muitos acreditam que resolução de conflito é uma questão de processamento e não uma questão para os desenvolvedores de software. Eliminando a possibilidade de conflitos através de design e procedimentos, o software não precisaria resolver conflitos de updates pois os mesmos nunca ocorreriam, na teoria.

Essa visão no entanto é muito limitada quando os clientes estão exigindo que várias opções para resolução de conflitos sejam incorporadas nas ferramentas de replicação de dados.

## Unique Keys e Generators

Tipicamente quando se trabalha com chaves únicas inteiras, uma trigger de BEFORE INSERT é utilizada adcionar e recuperar um novo valor de um generator. Quando se trabalha com replicação isso pode gerar alguns problemas.

O problema está em manter dois ou mais Banco de Dados sincronizados. Veja o exemplo abaixo:

Em um BD, o generator contém o valor 5. Quando um novo registro é adicionado, a trigger incrementa o valor do generator em 1 e o valor para a chave do novo registro passa à ser 6.

Este registro é replicado para um outro Banco de Dados.

Nesse segundo BD, o valor do generator é de 100. Quando o registro replicado for inserido no segundo BD, uma trigger sobrepõe o valor da chave (6) com o próximo valor do generator (101).

A solução tem duas possibilidades:

A primeira é confiar que o Cliente irá dar um valor a chave e não usar trigger.
A Segunda é dar intervalos de valores para cada BD, sendo que cada local possuirá certos valores de chave. Como no Interbase um valor do tipo Interger pode chegar a 2 bilhões de números, os intervalos de números poderiam ser divididos em partes de milhões se for necessário. O Generator para cada local poderá ser inicializado com um intervalo diferente usando para isso o comando para alterar o valor inicial do Generator:

SET GENERATOR GEN_NAME TO 1000000;

### Inicialização

Se você replicar uma origem de dados para um destino, então você precisará inicializar o destino pois ele estará fora de sincronia com a origem.. Tipicamente, você poderá fazer isto simplesmente implementando uma recarga dos dados da origem para o destino, ou seja do Banco de Dados para sua Replicação.

### Alterações na DDL

O que acontece quando você muda o Metadata? Se você adicionar um campo em uma tabela de um lado, esta alteração terá de ser efetuada também nos outros BDs.

## Resumo

Replicação de dados é um assunto importante nos dias de hoje e está sendo solicitado por um número cada vez maior de clientes em todo mundo. Na realidade, mais de cinquenta por cento do projetos iniciados desde dezembro de 1996 por mim (Dunstan Thomas) era requerido replicação.

A Replicação tem o potencial para prover um melhor processamento, respostas mais rápidas, redução de comunicação, e evitar os gargalos de rede entre usuários geograficamente espalhados.

Com o Interbase e o Microsoft RAS, a replicação de dados não é somente uma possibilidade mas sim uma realidade já disponível.