Instituto Superior de Engenharia de Lisboa
Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Mestrado em Engenharia Informática e de Computadores
Arquitetura de Sistemas Distribuídos
2013/2014
Relatório Aula Prática Nº 2

Nome	Nº de Aluno	E-mail
Rui Miranda	A32342	a32342@alunos.isel.pt
David Coelho	A21359	a21359@alunos.isel.pt
Frederico Ferreira	A7066	a7066@alunos.isel.pt

DEETC, Novembro de 2013

Introdução

Exercício 1

Comece por garantir que possui três instâncias do Sql Server 2012 a funcionar. Designemo-las por A, B e C. Em A, crie a base de dados BDMIRROR e nela execute o seguinte código SQL:

```
Create table t (i int)
Insert into t values(1)
```

Configure e coloque em funcionamento uma topologia de *mirroring* que inicialmente tenha A como principal, B como *mirror* e C como *witness*.

a) Pare a instância A e verifique que B toma o papel de principal. Em B, execute a instrução `insert into t values(2)` e faça `select * from t`. Faz sentido haver *failover* automático?

b) Arranque a instância A e verifique que papel ela toma. Justifique.

Preparação

No âmbito desta alínea, foram preparados 3 ficheiros:

- `0. run setup inicial.bat`
- `0.1. setup bd, table and execute backup @PrincipalDB.sql`
- `0.2. execute restore @MirrorDB.sql`

O primeiro executa os restantes, sendo o segundo ficheiro – um *script* – responsável pela criação da BD na instância A, criação da tabela t, inserção de uma linha de dados nesta tabela e finalmente *backup* desta BD. O terceiro *script* é responsável pela criação da BD e restauro do *backup* gerado anteriormente no contexto de cada uma das instâncias B e C.

Ao contrário do enunciado a tabela é composta por uma string e uma data. Isto faz-se com o intuito de facilitar os testes de *failover* com ADO.Net, que posteriormente se irão realizar.

Configure e coloque em funcionamento uma topologia de *mirroring* que inicialmente tenha A como principal, B como *mirror* e C como *witness*.

De forma a preparar esta topologia, seguiu-se o Wizard, à semelhança do que foi visto durante as aulas:

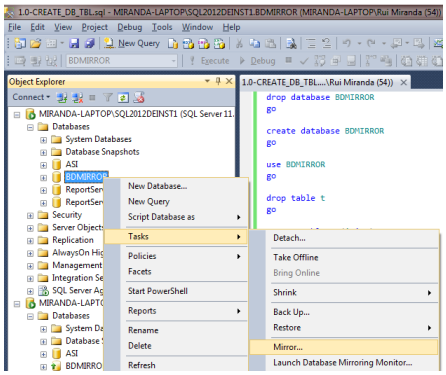


Figura 1: Inicia-se o Wizard de Mirroring

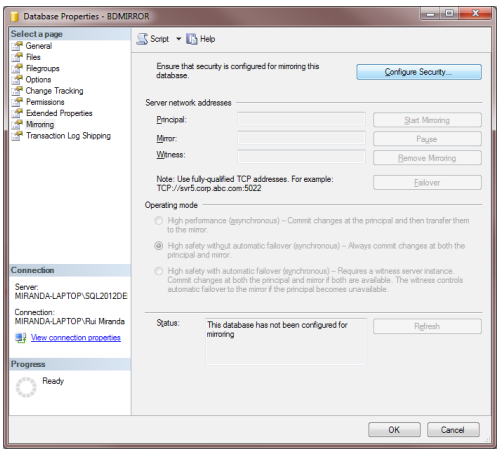


Figura 2: a interface inicial permite a escolha de instâncias

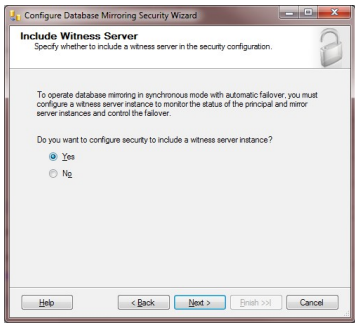


Figura 3: Neste passo determina-se que será necessário configurar uma 3ª instância para que o Failover automático funcione

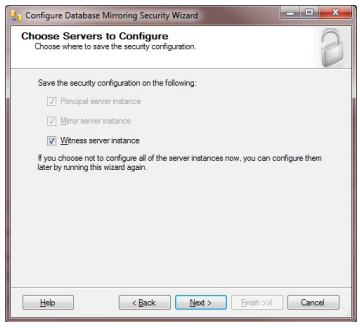


Figura 4: neste ponto indica-se que se pretende configurar a Witness durante este wizard

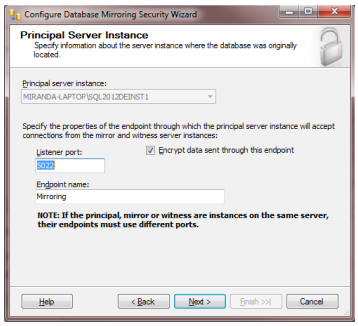


Figura 5: especificação sobre qual será a instância principal

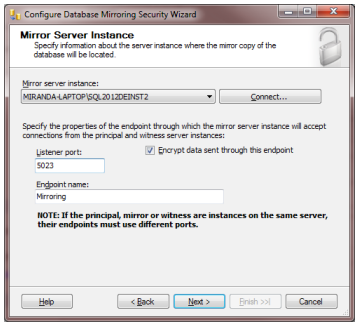


Figura 6: indicação sobre qual será a instância de stand-by e o respectivo porto de comunicações

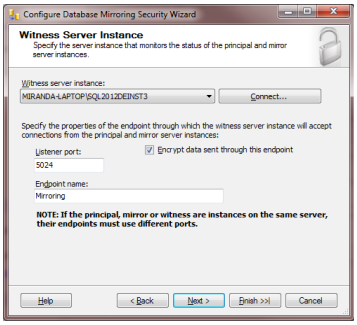


Figura 7: indicação sobre qual será a instância que desempenhará o papel de Witness neste processo

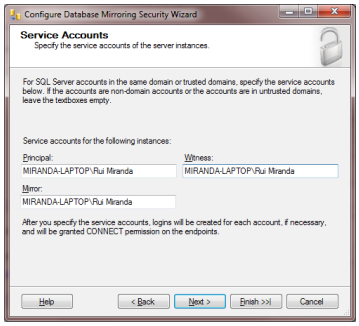


Figura 8: é apresentado um resumo com as credenciais de segurança que cada uma das instâncias usará

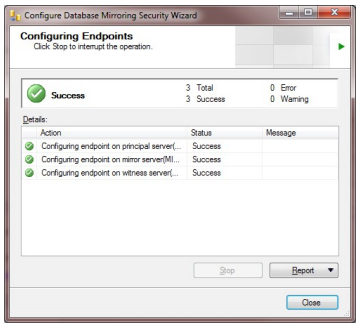


Figura 9: depois de executado é apresentado um relatório que indica que os 3 passos do processo foram executados com sucesso

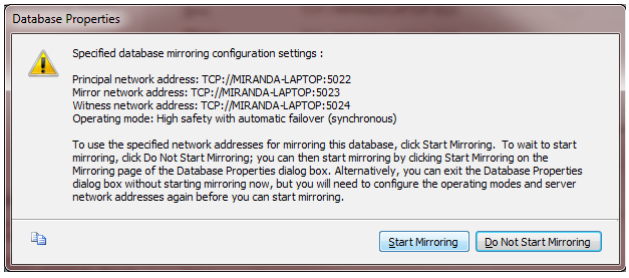


Figura 10: depois de configurado é possível inicial imediatamente o processo de Mirroring ou adiar esse inicio

O Wizard foi utilizado porque não foi encontrado em tempo útil um método de desempenhar esta tarefa via *scripting*.

Alínea a)

Objetivo

Pretende-se determinar se nas circunstâncias propostas – parando a instância A – faz sentido que B assuma o papel de instância principal, ou por outras palavras se faz sentido que haja *failover* automático.

Implementação

Para implementar a solução desta alínea, foram preparados 2 ficheiros:

- `1. realizar failovers.bat`
- `1.a. inserir dados na instancia.sql`

O primeiro executa uma série de instruções, começando pelas presentes no *script*: uma inserção na tabela t com o nome da instância onde o *script* está a correr – a instância principal – e a data actual. Em seguida é ordenada a paragem da instância principal. De seguida corre-se novamente o *script,* desta vez na instância B; a instância *mirror* que entretanto foi promovida a principal.

Com esta execução confirma-se que a tabela t tem uma linha inserida pela instância principal e uma segunda linha de dados inserida pela instância *mirror*. Faz sentido que a instância *mirror* assuma o papel de instância principal porque quando a instância principal parou, tanto a instância *mirror* como a instância *witness* estavam disponíveis e em contacto entre si. Tendo em conta que ambas perderam o contacto com a instância principal, e estando a instância *mirror* actualizada com a mais recente transacção executada – o segundo *insert* – a instância *witness* dá *quorum* à instância *mirror* e assim esta assume o papel de instância principal.

Alínea b)

Objetivo

Pretende-se aqui explicar o motivo pelo qual a instância A, que inicialmente desempenhava o papel de instância principal, se comporta como instância *mirror* quando é reactivada.

Implementação

Depois do utilizador reiniciar a instância principal, constata-se que esta assume o papel de *mirror*, conforme se pode verificar na figura.

Este comportamento faz sentido porque quando a instância A é reactivada, B encontra-se activa e a desempenhar as tarefas de instância principal. Não tendo *quorum* nem tendo um motivo para o pedir à instância *witness*, resta-lhe apenas resincronizar-se com a instância principal e passar a desempenhar as funções de instância *mirror*.

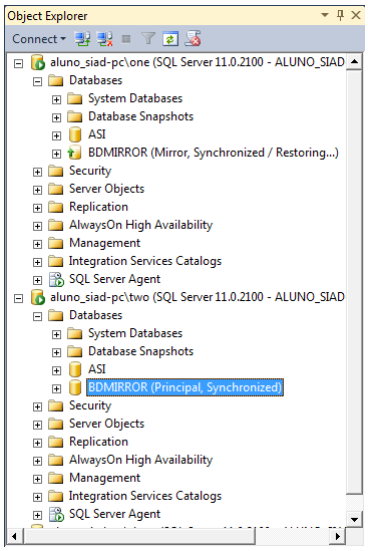


Figura 11: a primeira instância assume o papel de Mirror

Realize *failover* manual de B para A e espera que A e B troquem de papéis. Execute em A `select * from t`.

Depois utilizar a interface gráfica para comutar as funcionalidades das instâncias, a execução do comando `select * from t` em A, permite-nos constatar que A já sincronizou com B e já contem os dados do segundo *insert* – a inserção de dados realizada na instância B.

c) Pare a instância A e espere que B assuma o papel de principal. Execute em B `insert into t values(3)` e verifique que a operação sucede. Pare B. Active A e verifique se ela toma o papel de principal. Justifique o que observa.

d) Pare A e arranque B e espere que B assuma o papel de principal. Arranque A e espere que A e B se sincronizem. Pare C. Pare B. Verifique que não há *failover* automático. Justifique.

Alínea c)

Objetivo

Pretende-se aqui testar as circunstâncias nas quais a instância de *witness* dá *quorum* às instâncias que o pedem.

Implementação

É executado o comando de *shutdown* na instância A e espera-se que B assuma o papel de principal. Em B é inserida na tabela t uma terceira linha de dados com o nome da instância B e com a data actual. Depois de realizada esta escrita, é dado o comando de *shutdown* à instância B.

Depois de inactivada a instância B, activa-se a instância A a qual não assume o papel de instância principal. Tal facto é motivado pela paragem da instância B se ter dado antes da a instância A ter sido activada. Desta forma, não tendo sido actualizada com a mais recente transacção executada – o terceiro *insert* – a instância *witness* não dá *quorum* à instância A e assim esta assume o papel de instância *mirror*.

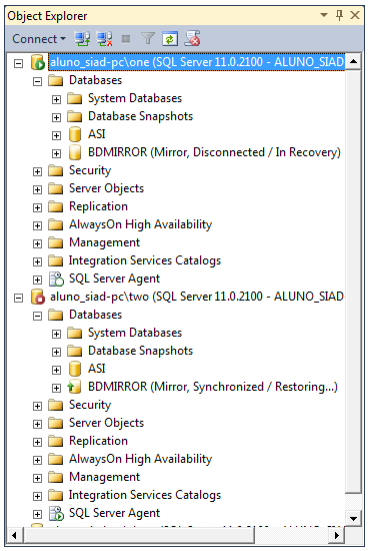


Figura 12: a instância A não pode prosseguir porque não sincronizou com B

Alínea d)

Objetivo

Estando a instância B a desempenhar o papel de instância principal, e a instância A a desempenhar o papel de instância *mirror*, pretende-se verificar o comportamento do sistema quando a *witness* está indisponível.

Implementação

Depois de se ter colocado o sistema no estado indicado (a instância B como instância principal e a instância A como instância *mirror*), é executado o comando de *shutdown* na instância *witness* (C) e na instância principal (B). Seria espectável que A assumisse o papel de instância principal. Infelizmente tal não acontece porque como a instância *witness* está parada, a instância *mirror não* tem forma de saber se ausência de resposta da instância principal é sinal de problemas com na instância principal ou se é sinal de problemas com a própria instância *mirror*. Assim sendo, não assume controlo das operações da BD, e fica no modo *mirror* desconectada da instância principal.

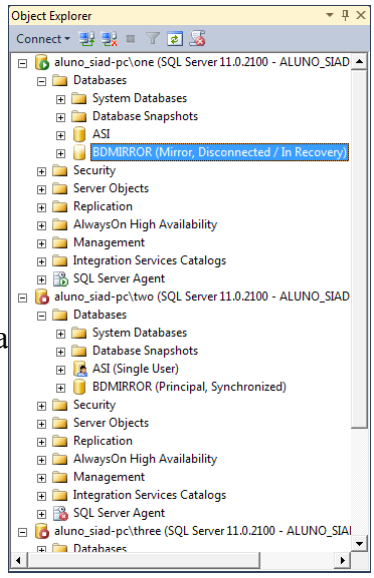


Figura 13: verifica-se a ausência de failover automático

Arranque B e execute um *failover* manual de B para A.

e) Arranque C e teste o mecanismo do ADO.NET para reencaminhamento de conexões na presença de mirroring. Indique como realizou o teste e que conclusões tirou.

Alínea e)

Objetivo

Com todas as instâncias activas e estando a instância A a desempenhar o papel de instância principal, e a instância B a desempenhar o papel de instância *mirror*, pretende-se verificar o comportamento de uma aplicação ADO.Net quando se realiza *failover*.

Implementação

Foi criado um pequeno programa que recorre às bibliotecas de ADO.Net para realizar dois *selects* à base de dados: o primeiro determina o nome da instância e o segundo devolve o conteúdo da tabela 't'. Estas duas *queries* são repetidas até o programa ser terminado pelo utilizador. A única particularidade que a aplicação tem relativamente ao suporte a *mirroring* está na sua *connection string*:

```
"Address=aluno_siad-pc\one;Database=BDMIRROR;Failover Partner=aluno_siad-pc\two;Integrated Security = true"
```

A introdução da *keyword* `Failover Partner` é a única diferença relativamente a uma versão deste programa que *não* suporte *mirroring*. Note-se que durante a execução ocorrem dois *failovers* ilustrados pelas mensagens de "erro ao nível do transporte" e "Ocorreu failover":

```
ConnectionDB: BDMIRROR ConnectionDataSrc: aluno_siad-pc\two Conteudo obtido as 2013-11-21T13:38:56:
Running queries @ALUNO_SIAD-PC\TWO
Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53

ConnectionDB: BDMIRROR ConnectionDataSrc: aluno_siad-pc\two Conteudo obtido as 2013-11-21T13:38:56:
Running queries @ALUNO_SIAD-PC\TWO
Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53

A transport-level error has occurred when sending the request to the server. (provider: Shared Memory
Provider, error: 0 - No process is on the other end of the pipe.)

Ocorreu failover!!
ConnectionDB: BDMIRROR ConnectionDataSrc: aluno_siad-pc\one Conteudo obtido as 2013-11-21T13:39:13:
Running queries @ALUNO_SIAD-PC\ONE
Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53

ConnectionDB: BDMIRROR ConnectionDataSrc: aluno_siad-pc\one Conteudo obtido as 2013-11-21T13:39:14:
Running queries @ALUNO_SIAD-PC\ONE
Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53
(...)
ConnectionDB: BDMIRROR ConnectionDataSrc: aluno_siad-pc\one Conteudo obtido as 2013-11-21T13:39:23:
Running queries @ALUNO_SIAD-PC\ONE
Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53

ConnectionDB: BDMIRROR ConnectionDataSrc: aluno_siad-pc\one Conteudo obtido as 2013-11-21T13:39:23:
```

```
  Running queries @ALUNO_SIAD-PC\ONE
  Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
  Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53


  A transport-level error has occurred when sending the request to the server. (provider: Shared Memory
Provider, error: 0 - No process is on the other end of the pipe.)


  Ocorreu failover!!
  ConnectionDB: BDMIRROR ConnectionDataSrc: ALUNO_SIAD-PC\TWO Conteudo obtido as 2013-11-21T13:39:42:
  Running queries @ALUNO_SIAD-PC\TWO
  Instance ALUNO_SIAD-PC\ONE wrote info @2013-11-21T13:30:19
  Instance ALUNO_SIAD-PC\TWO wrote info @2013-11-21T13:32:53
```

O resultado do *failover* está ilustrado na listagem acima: a única preocupação que é necessário ter consiste em realizar as consultas à BD sabendo que a ligação pode falhar e se tal acontecer, graças ao *failover* automático, basta voltar a tentar que a infraestrutura do ADO.Net liga-se ao *failover partner*.

Exercício 2

Usaremos novamente as 3 instâncias (A,B e C) do SqlServer

Em A, crie a base de dados BDLS e nela execute o seguinte código SQL:

```
Create table t (i int)
Insert into t values(1)
```

Configure e coloque em funcionamento uma topologia de *log shipping* que tenha A como primário e B e C como secundários, devendo C ficar em modo *standby*. Defina os escalonamentos para os *jobs* de criação de backups, de cópia e de *restore* de forma a que se executem de 30 em 30 segundos.

Preparação

De forma a preparar o sistema para a execução desta alínea, foram executados diversos comandos registados nos seguintes ficheiros:

- `2.0-run.setup.bat`
- `2.0-CREATE_BDLS_TBL-A.sql`

Script a executar no nó principal (A): cria a base de dados BDLS, cria a tabela t nessa mesma BD e insere o valor 1. Inicialmente, caso os objectos já existam, o *script* elimina-os, de forma a retornar o sistema a um estado conhecido.

- `2.1-SET-BDLS-RECOVERY-FULL-A.sql`

De modo a garantir que a base de dados BDLS se encontra em FULL ou BULK-LOGGED RECOVERY, é necessário executar este *script*.

- `2.2-BACKUP-BDLS-A.sql`

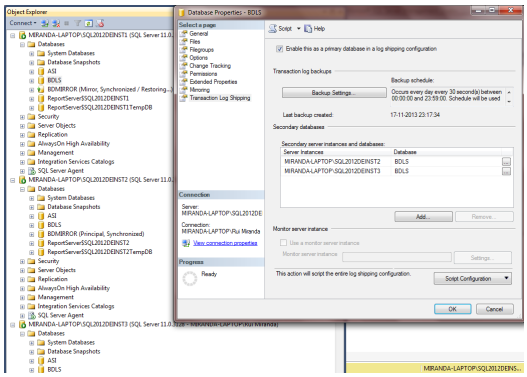
Efectua o *backup* da base de dados BDLS no nó principal (A). Realiza *overwrite* do conteúdo do ficheiro se este já existir.

- `2.3-RESTORE-BDLS-BeC.sql`

Efectua o *restore* do ficheiro de *backup* -- criado a partir da base de dados BDLS do nó principal -- em cada um dos nós secundários. Caso a BD BDLS já exista num dado nó secundário, a base de dados será eliminada logo no inicio da execução do script.

O primeiro ficheiro listado é um *batch* que se encarrega de executar os restantes ficheiros de *script* SQL: O *batch* ordena a criação da BD 'BDLS' na instância A, a execução do *backup* da instância A para um ficheiro numa directoria criada para o efeito e finalmente ordena o restauro desse *backup* para as instâncias B e C. Note-se que o *batch* executa este último *script* nos nós B e C com uma parametrização diferente: em B o *restore* é feito com `NORECOVERY` enquanto em C o *restore* é realizado com `STANDBY`. Esta diferença justifica-se devido ao requisito do enunciado segundo o qual, o nó C deve ficar em modo *standby.*

Depois de, com o auxilio do *Wizard* que consta na figura, determinar quais são os procedimentos que regem a configuração do *log shipping*, foram criados os seguintes ficheiros:



- `2.4-run.setup.LogShiping.bat`
- `2.4-RemoveLogShiping-BDLS @Secondary.sql`
- `2.5-RemoveLogShiping-BDLS @Primary.sql`

Com estes dois *scripts* pretende-se eliminar todas as configurações já existentes relativas a *Log Shipping*, quer no nó primário quer nos secundários.

- `2.6-ConfigLogShiping-BDLS @Primary.sql`

Neste *script* configuram-se o *log shipping* no nó primário (A), o agendamento do backup, e a referência para os nós secundários, B e C.

- `2.7-ConfigLogShiping-BDLS @Secondary.sql`

O último *script* é executado nos nós (B e C) os quais são configuramos como nós secundários no âmbito do *log shipping* proveniente de A. É também configurado o agendamento para *restore* neste nós.

Também aqui o *batch* encarrega-se de executar os *scripts*: os dois primeiros removem qualquer configuração que se possa já ter feito em A ou B e C relativamente a *log shipping*, algo que pode acontecer se se executar o *batch* duas vezes. O *batch* finaliza executando os dois últimos *scripts*, os quais realizam a configuração do *log shipping*: o primeiro na instância A e o segundo nas instâncias B e C.

a) Em A execute a instrução `insert into t values(2)` e faça `select * from t`. Verifique que após algum tempo esse registo aparece em C.

Alínea a)

Objetivo

Com esta alínea pretende-se verificar se o *log shipping* está a funcionar. Realizando uma escrita de dados no nó principal espera-se que, passados no mínimo 30 segundos, seja possível ler esses mesmos dados a partir do nó secundário C, o qual se encontra em *standby*.

Implementação

A implementação inclui dois ficheiros que executam quase todas as acções necessárias. A única actividade que não foi automatizada foi o arranque das instâncias de *SQL Server*, as quais devem ser iniciadas manualmente nos momentos indicados pelo *batch*. Assim, esta alínea é implementada em dois ficheiros:

- `2.a-run.test.bat`
- `2.a-DELETE-INSERT Primary.sql`

O ficheiro *batch* começa por executar o *script sql* o qual é responsável por

- descobrir qual é a linha com o valor v -- o valor mais elevado que existe na tabela 't',
- apagar essa linha,
- inserir em 't' uma linha com o valor v+1

Espera-se que estas acções tenham repercussões nas instâncias secundárias. No entanto o processo de *log shipping* está configurado para ocorrer 2 vezes por minuto, motivo pelo qual o *batch* realiza uma espera de 30 segundos. Passado este tempo é realizada uma *query* sobre a instância secundária C onde se constata que a alteração foi efectivamente propagada a este nó.

Conclusões

Com esta alínea confirma-se que o processo de *log shipping* está funcional e que neste momento o funcionamento do negócio está salvaguardado contra imprevistos que possam ocorrer em **uma** das três instâncias.

b) Imagine uma situação de desastre em A e execute um *role change* para que o controlo passe a estar em B. Verifique qual o estado da tabela em B.

Alínea b)

Objetivo

Com esta alínea pretende-se simular uma situação de indisponibilidade no nó principal (A) permitindo tirar partido do *log shipping*. Se funcionar, o *log shipping* irá permitir que o sistema esteja disponível em circunstancias onde, sem *log shipping*, isso não seria possível.

Implementação

A implementação é concretizada em seis ficheiros:

- `2.b.0-setup.LogShiping.CenarioDesastre.bat`
- `2.b.1-RemoveLogShiping-BDLS @Secondary.sql`
- `2.b.2-RemoveLogShiping-BDLS @Primary.sql`
- `2.b.3-ConfigLogShiping-BDLS @Primary.sql`
- `2.b.4-ConfigLogShiping-BDLS @Secondary.sql`
- `2.b.5-run.test.bat`

À semelhança das alíneas anteriores, o *batch* orquestra as acções a desempenhar no âmbito desta alínea recorrendo para o efeito aos *scripts* de SQL. No entanto, começa por simular um desastre na instância principal (A): emitindo um comando de *shutdown* torna aquela instância inacessível. Usa em seguida os dois primeiros *scripts*, para desfazer as configurações realizadas nas alíneas anteriores, nas duas instâncias restantes (B e C). A instância B (a primeira instância secundária) é levada ao estado *online* e tanto B como C são reconfiguradas como instâncias primária e secundária respectivamente, num esquema de *log shipping* -- precisamente o objectivo dos restantes dois *scripts*.

Depois de realizada a configuração do *log shipping* pelo primeiro *batch*, recorre-se ao segundo *batch* para testar a configuração de *log shipping*. O teste é essencialmente idêntico ao levado a cabo nas alíneas anteriores: realizam-se algumas escritas na instância principal (B) e esperam-se 30 segundos, o tempo que foi pré-programado entre execuções do processo de *log shipping*. Depois desta pausa são realizadas leituras tanto em B como em C. O facto de ambas as instâncias reportarem valores idênticos confirma que o processo de *log shipping* está a funcionar correctamente.

Conclusões

Com esta alínea confirma-se que o processo de *log shipping* criado na alínea anterior cumpriu a sua função: garantir a continuidade do negócio. Uma vez parada a instância A, o processo de activar a instância B e configura-la como nova instância principal realiza-se numa questão de segundos. A alínea presente reconfigurou o processo de *log shipping* de forma a que, caso ocorra outro imprevisto em B, o negócio possa continuar a funcionar normalmente, a partir da instância C.

Exercício 3

Considere uma BD com as tabelas Alunos (número, nome), Disciplinas (código, designação) e Inscrição (Numero[FK], código [FK], Nota, NumSeq [identity, candidate key]).

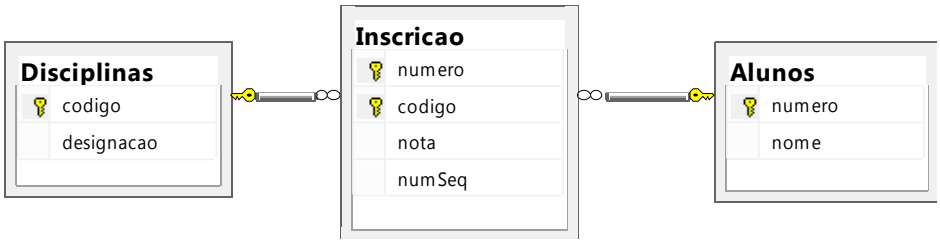
a) Proponha uma solução que permita melhorar o desempenho das leituras e escritas nesta BD, usando *database sharding*(para os alunos) e replicação síncrona (para as disciplinas). Considere um número de partições compatível com o número de instâncias do *Sql Server* de que dispõe. Implemente e teste essa solução, incluindo uma aplicação para a inserção de alunos e de inscrições.

b) Repita a alínea anterior, mas usando replicação P2P para as disciplinas.

c) Discuta as vantagens e desvantagens das duas soluções dos pontos de vista de expansibilidade (leitura e escrita), autonomia local dos processamentos e de consistência dos dados.

Preparação

Este exercício tem por foco a expansão de um sistema informação por via de scale-out para melhoria de desempenho de uma base dados, com a distribuição dos dados (para efeitos de desempenho e não no contexto de base de dados distribuída) por diversos nós, por forma a distribuir a carga, e consequentemente melhorar o desempenho global da base de dados.



Alínea a)

Objetivo

Desenhar uma solução com usando uma técnica de *database sharding* para a tabela de Alunos e um replicação síncrona para a Disciplinas, considerando que temos (no ambiente criado para os exercícios práticos) 3 instâncias do SQL Server.

A implementação de *sharding* passa por criar uma fragmentação horizontal da tabela Alunos, que permita distribuir uniformemente a atividade (acessos aos dados) pelos nós disponíveis (assumindo neste caso que a capacidade de processamento e armazenamento disponível para esta atividade é igual em todos os nós).

A fragmentação horizontal para melhoria de desempenho é uma técnica conhecida por *sharding* e usa uma chave (key) para controlar a distribuição dos dados pelos diferentes nós. O *sharding* pode ser por "range", por "lista" ou por "hash":

- "range": a partir de uma chave numérica são definidos intervalos disjuntos (mas que cobrem todo o universo possível de valores da chave) para distribuir os registos pelos diferentes nós;
- "lista": com base numa lista pré-definida os registos são distribuídos pelos diferentes nós. Os elementos da lista devem ser escolhidos por forma a tentar balancear a distribuição dos dados pelos diferentes nós, isto considerando que existe uma distribuição uniforme da atividade. Por exemplo no caso do exemplo, poderia ser montada uma lista para distribuir com base na primeira letra do nome, tendo em conta a distribuição por iniciais e sempre com o foco no balanceamento da atividade nos nós;
- "hash": isto é usando uma função de distribuição com um bom resultado de dispersão para distribuir os registos pelos diferentes nós.

Cada uma destas alternativas têm as suas vantagens e desvantagens, em elementos como a uniformidade da distribuição, a flexibilidade para a introdução de mais nós, ou mesmo aproveitar a dispersão geográfica (que não se aplica no caso presente).

Implementação

Sharding

Para a implementação de *sharding* na tabela de alunos e considerando que existe uma afinidade entre o crescimento da numeração e a atividade (os alunos mais velhos vão saindo da escolar os mais recentes são os que geram mais atividade), pareceu-nos que um bom processo de distribuição seria usar a função módulo no campo **numero** o que permite uma distribuição equilibrada por todos os nós.

Neste caso o nó do registo é dado pelo resto da divisão do número de aluno por 3 (número atual de nós).

A solução é equilibrada para a situação atual, uma vez que distribui a atividade por todos os nós, o que não aconteceria facilmente com o uso de bandas de números de alunos (os alunos mais recentes têm mais interações com a escola em comparação com os alunos mais antigos, que já saíram da escola).

Na introdução de novas instâncias (réplicas) será necessário reestruturar todos os existentes para manter o balanceamento da carga.

Ainda em termos de implementação para que o *sharding* seja eficaz a tabela de "inscrições" está também replicada com o mesmo critério da tabela "alunos", enquanto a tabela "disciplinas" por não ter uma lógica de distribuição ligada ao "aluno", precisa de ser replicada por todos os nós. Em termos de consulta um nó é portanto auto-suficiente e beneficia da técnica de *sharding* usada, na alteração da tabela de "disciplinas" será necessário ter uma transação distribuída e síncrona para garantir a atualização simultânea de todos os nós pagando um preço por isso (em termos de desempenho).

Sendo a replicação ("*sharding*") usada em Leitura e Escrita, o encaminhamento é feito no código da aplicação (e não na base de dados).

Na tabela "Inscricao", também ela em "*sharding*" o atributo "numSeq" do tipo "identity", tem uma inicialização diferente para cada nó, mas um valor de salto igual para evitar colisões.

Replicação Síncrona

A tabela "Disciplinas" está replicada em todos os nós e é usada replicação sincrona na sua alteração. As alterações (com replicação sincrona) foram implementadas por procedimentos no nó zero e criados sinónimos nos restantes. Como a replicação é sincrona todos os nós têm de estar disponiveis incluindo o zero, pelo que não há problema de iniciar aí todas alterações.

Testes

Os testes foram implementados com um programa C#, que implementa o mecanismo de "sharding", num objecto que gere as conexões e que as entrega em função do número de aluno (nos comandos para os quais isso é relevante).

Foi usado um mecanismo de "using" e implementado o correspodente médoto "dispose" no objecto usado no "using".

O programa de testes faz primeiro uma limpeza dos dados anteriores e depois uma inserção de Alunos, Disciplinas e Inscrições, usando o "sharding" e procedimentos no caso das "Disciplinas".

As operações que envolvem os diferentes nós numa única transação podem numa primeira execução apresentar o erro MSDTC on server 'xxx' is unavailable, para resolver este erro, caso aconteça é necessário arrancar com os serviços, ver anexo 1, para o forma de o conseguir.

Alínea b)

Objetivo

Substituir a replicação sincrona usada na tabela “Disciplinas” por uma estratégia de P2P estabelecida entre todos os nós, mas que é assíncrona.

Implementação

Na implementação apagámos os procedimentos e os respectivos sinónimos e com o SQL Manager, de forma interativa criamos o cenário P2P.

Passos para a implementação da replicação P2P:

1. em Replication selecionar New Publication e seguir as instruções do Wizard
2. escolher a base de dados DB2_3
3. selecionar P2P publication e escolher a tabela Disciplinas
4. configurar o Agent Security
5. Criar a DisciplinasPublication
6. De novo em Replication/Local Publications/ selecionar a publicação criada para configurar a topologia
7. incluir os 3 nós e selecionar a replicação entre todos os nós em fomato push
8. Iniciar a replicação

O nome usado para a publicação foi “p2ex3B”.

Os testes que fizemos foram:

- Inserir um registo em “Disciplinas” num nó e ver que a propagamação tem um tempo (um select imediato nos restantes nós permite ver que a a alteração ainda não foi propagada)
- apagar num nó uma “Disciplina”, que era usada numa “Inscricao” noutro nó, cria uma inconsistência, como pode ser vista na figura abaixo

```
select i.numero, i.codigo, a.nome, d.designacao
from dbo.Inscricao i, dbo.Disciplinas d, dbo.Alunos a
where a.numero = i.numero
  and d.codigo = i.codigo

select * from dbo.Inscricao
```

Results | Messages

	numero	codigo	nome	designacao
1	2	2	Antonio Silva	Arq. de Computadores II
2	2	3	Antonio Silva	Análise de Circuitos

	numero	codigo	nota	numSeq
1	2	2	20	27
2	2	3	19	30
3	5	12	14	33

o registo da inscrição continua a existir mas a correspondente “Disciplina” foi apagada.

A tabela de “Inscricoes” foi criada com a seguinte “constraint”:

```
ALTER TABLE [dbo].[Inscricao]  WITH NOCHECK ADD  CONSTRAINT [FK_Inscricao_Disciplinas] FOREIGN
KEY([codigo])
REFERENCES [dbo].[Disciplinas] ([codigo])
NOT FOR REPLICATION
```

Caso contrário teria havido um erro na propagação da alteração para aquele nó.

Erro: sql server is unable to connect to server replication

Durante a implementação obtivemos este erro num dos nós (instância), que não deu qualquer probema nos outros ensaios, mas que impedia a incorporação do nó na replicação P2P (que já havia funcionado antes). A solução passou pelo registar de novo o nó local que passou a apresentar correctamente @@SERVERNAME em resultado de select no próprio nó.

Para mais detalhe e notas sobre a correção ver anexo 1.

Alínea c)

Conclusões

As duas soluções são equivalentes do ponto de vista de leitura e escrita na tabela “Alunos”. Cada nó é autónomo e funciona independentemente dos restantes.Quanto às tabelas de “Disciplinas” e “Inscricao”, as duas soluções diferem nos seguintes termos:

Solução a) Replicação Síncrona:

A tabela “Disciplinas” só é alterável quando todos os nós(réplicas) estiverem no ar e em comunicação.

As vantagens são garantir a consistência em todas as réplicas o que se aplica também às alterações da tabela “Inscricao”, que tem a garantia da ligação com “Disciplinas”.

As desvantagens são a duração da transação síncrona e a necessidade da disponibilidade das réplicas para a sua execução.

Solução b) Replicação Assíncrona:

A tabela “Disciplinas” só é alterável em qualquer nó e em qualquer altura independentemente dos restantes estarem ou não disponíveis e comunicáveis.

As vantagens são a autonomia de cada nó para as alterações, a todas as tabelas e a rapidez de todas as alterações (mesmo a “Disciplinas”).

As desvantagens são a inconsistência dos dados durante o período de propagação das alterações e o mais grave o caso de ser apagada uma “Disciplina”, que esteja apenas em uso num outro nó (de onde não poderá ser removida por violar a integridade referencial). Neste caso a inconsistência não está apenas limitada ao tempo de propagação e terá de ser resolvida manualmente.

Final

No caso de negócio presente onde, não será provável anular uma disciplina depois de criada e as alterações (correcções ?) de nome não seriam relevantes para o negócio, a solução b) Replicação assíncrona seria a mais vantajosa.

ANEXO 1 – Documentos de Apoio

1.Enable Network Access Securely for MS DTC

TechNote: http://technet.microsoft.com/en-us/library/cc753620(WS.10).aspx

To enable Network DTC Access for MS DTC transactions

1. Open the Component Services snap-in.
   To open Component Services, click Start. In the search box, type **dcomcnfg**, and then press ENTER.
2. Expand the console tree to locate the DTC (for example, Local DTC) for which you want to enable Network MSDTC Access.
3. On the Action menu, click Properties.
4. Click the Security tab and make the following changes:
   - In Security Settings, select the Network DTC Access check box.
   - In Transaction Manager Communication, select the Allow Inbound and Allow Outbound check boxes.
5. ClickOK.