



Guia do Servidor e Administração do Derby

Version 10

Derby Document build:
October 2, 2006, 8:09:39 AM (PDT)

Contents

Direitos autorais reservados	4
Sobre este guia	5
Finalidade deste guia	5
Audiência	5
Como este guia está organizado	5
Parte 1: Guia do servidor Derby	7
O Derby em um ambiente multiusuário	7
O Derby em uma estrutura servidora.....	7
Sobre este guia e a documentação do Network Server.....	10
Utilização do Network Server com aplicativos Derby pré-existent	10
O Network Server e as JVMs.....	10
Instalação dos arquivos jar requeridos e adições ao caminho de classes.....	10
Inicialização do Network Server.....	11
Parar o Network Server.....	12
Obtenção de informações do sistema.....	13
Acesso ao Network Server utilizando o driver cliente da rede.....	15
Acesso ao Network Server utilizando DataSource.....	22
XA e o Network Server.....	23
Utilização das ferramentas do Derby com o Network Server.....	23
Diferenças entre executar o Derby no modo incorporado e utilizando o Network Server.....	24
Definição do número da porta.....	27
Gerenciamento do Network Server do Derby	27
Visão geral.....	27
Definição das propriedades do Network Server.....	28
Verificação da inicialização.....	32
Gerenciamento remoto do Network Server do Derby utilizando a interface servlet	32
Página de inicialização.....	33
Página de execução.....	33
Página de rastreamento da sessão.....	34
Página do diretório de rastreamento.....	34
Definição de parâmetros do Network Server.....	34
Tópicos avançados do Network Server do Derby	34
Segurança do Network Server.....	34
Execução do Network Server sob o gerenciador de segurança.....	35
Configuração do Network Server para tratar as conexões.....	36
Controle do registro pela utilização do arquivo de log.....	37
Controle do rastreamento pela utilização da facilidade de rastreamento.....	37
Programas exemplo para o Network Server do Derby	38
O programa exemplo NsSample.....	38
Programas exemplo do Network Server para conexões incorporada e cliente.....	41
Parte 2: Guia de administração do Derby	44
Verificação da consistência do banco de dados	44
A função SYSCS_CHECK_TABLE.....	44
Exemplos de mensagem de erro de SYSCS_CHECK_TABLE.....	44
Exemplos de comandos SYSCS_CHECK_TABLE.....	45
Cópia de segurança e restauração do banco de dados	45
Cópia de segurança do banco de dados.....	46
Restauração do banco de dados a partir da cópia de segurança.....	49
Criação de um banco de dados a partir de uma cópia de segurança.....	49
Recuperação com rolagem para frente.....	49
Arquivo de log em uma unidade separada	52
Utilização do atributo logDevice.....	52
Exemplo de criação do log em um local diferente do padrão.....	52
Exemplo de mover o log manualmente.....	53

Questões relativas ao log em um local diferente do padrão.....	53
Obtenção de informações sobre bloqueios.....	53
Monitoramento dos impasses.....	53
Recuperação do espaço não utilizado.....	54
Marcas Registradas.....	56

Direitos autorais reservados

Segunda Edição (Julho de 2005)

Copyright 1997, 2005 Apache Software Foundation, ou seus concessionários de licença, conforme se aplique.

Licenciado sob a Licença Apache, Versão 2.0 (doravante chamada apenas de "Licença"); este arquivo não pode ser utilizado a não ser em conformidade com a Licença. Pode ser obtida uma cópia da Licença em

<http://www.apache.org/licenses/LICENSE-2.0>

A menos que seja requerido por lei aplicável, ou concordado por escrito, o programa distribuído sob esta Licença é distribuído na BASE DE "COMO É", SEM GARANTIAS OU CONDIÇÕES DE QUALQUER ESPÉCIE, sejam expressas ou implícitas. Veja na Licença as condições específicas que governam as permissões e limitações sob a Licença.

Sobre este guia

Esta seção descreve a quem este guia se destina, assim como a forma de usá-lo.

Finalidade deste guia

Este guia explica como utilizar o Derby em um ambiente com vários clientes. Também fornece informações que o administrador do servidor pode necessitar para manter o Derby funcionando com alto nível de desempenho e confiabilidade em uma estrutura servidora, ou em um ambiente servidor de aplicativos com vários clientes (Quando estão executando no modo incorporado, os bancos de dados do Derby normalmente não necessitam de qualquer administração).

Para conectar vários clientes ao Derby, o Derby pode ser incorporado à estrutura servidora escolhida, ou pode ser utilizado o *Network Server* do Derby. Este guia descreve estas opções.

Audiência

A primeira parte deste guia destina-se aos desenvolvedores de aplicativos cliente/servidor e de vários clientes. A segunda parte deste guia destina-se aos administradores.

Como este guia está organizado

Este guia inclui as seguintes partes:

Parte 1: Guia do Servidor Derby

- [*O Derby em um ambiente multiusuário*](#)

Descreve as diferentes opções para incorporar o Derby a uma estrutura servidora, e explica a opção *Network Server*.

- [*Utilização do Network Server com aplicativos Derby pré-existent*](#)

Descreve como alterar os aplicativos Derby existentes para trabalhar com o *Network Server*.

- [*Gerenciamento do Network Server do Derby*](#)

Descreve como utilizar scripts do interpretador de comandos (*shell*), a linha de comando, e a API do *Network Server*, para gerenciar o *Network Server*.

- [*Gerenciamento remoto do Network Server do Derby utilizando a interface servlet*](#)

Descreve como utilizar a interface *servlet* para gerenciar o *Network Server*.

- [*Tópicos avançados do Network Server do Derby*](#)

Descreve tópicos avançados para os usuários do *Network Server* do Derby.

Parte 2: Guia de administração do Derby

- [*Verificação da consistência do banco de dados*](#)

Descreve como verificar a consistência dos bancos de dados do Derby.

- [*Cópia de segurança e restauração do banco de dados*](#)

Descreve como fazer cópias de segurança dos bancos de dados enquanto estes estão em linha.

- *Arquivo de log em uma unidade separada*

Descreve como colocar o *log* do banco de dados em uma unidade separada, o que pode melhorar o desempenho de bancos de dados grandes.

- *Obtenção de informações sobre bloqueios*

Descreve como obter informações detalhadas sobre status de bloqueios.

- *Recuperação do espaço não utilizado*

Descreve como identificar e recuperar o espaço não utilizado nas tabelas e seus índices relacionados.

Parte 1: Guia do servidor Derby

Esta parte do guia explica o *Network Server* do Derby, e outras estruturas servidoras.

O Derby em um ambiente multiusuário

Esta seção descreve o Derby em um ambiente multiusuário (ou "servidor")

O Derby em uma estrutura servidora

Em certo sentido, o Derby está sempre incorporado a algum produto. Pode estar incorporado a aplicativos onde os usuários acessam o banco de dados a partir de uma única JVM, ou pode estar incorporado a uma estrutura servidora (um aplicativo que permite usuários em JVMs diferentes se conectarem ao Derby simultaneamente). Quando o Derby está incorporado a um aplicativo, o *driver* de JDBC local chama o banco de dados Derby local. Quando o Derby está incorporado a uma estrutura servidora, o software de conectividade da estrutura servidora fornece dados para vários aplicativos JDBC cliente através da rede ou da Internet.

Para a conectividade multiusuária local ou remota (vários usuários acessando o Derby a partir de JVMs diferentes), é utilizado o *Network Server* do Derby. Havendo necessidade de funcionalidades não incluídas no *Network Server*, o produto Derby básico poderá ser incorporado a outra estrutura servidora.

Configurações de conectividade

Existem diversas maneiras de incorporar o Derby a uma estrutura servidora:

Utilizando o *Network Server*

Esta é a forma mais fácil de fornecer conectividade a vários usuários acessando os bancos de dados do Derby a partir de JVMs diferentes. O *Network Server* do Derby fornece este tipo de conectividade para os bancos de dados do Derby dentro de um único sistema, ou através da rede.

Adquirir outra estrutura servidora

O Derby pode ser utilizado dentro de várias estruturas servidoras, como o *IBM WebSphere Application Server*.

Desenvolver sua própria estrutura

A flexibilidade do Derby permite outras configurações também. Por exemplo, em vez de incorporar o Derby a um servidor que se comunica com cliente que utiliza o JDBC, o Derby pode ser incorporado a uma *servlet* em servidor Web que se comunica com o navegador utilizando HTTP.

Funcionalidades disponíveis no Derby para atender vários clientes

O Derby contém algumas funcionalidades úteis para desenvolver aplicativos multiusuários.

Bloqueio no nível de linha:

Para dar suporte a acesso multiusuário, o Derby utiliza bloqueio no nível de linha. Entretanto, o Derby pode ser configurado para utilizar bloqueio no nível de tabela em ambientes com poucas transações simultâneas (por exemplo, banco de dados apenas para leitura). O bloqueio no nível de tabela é preferível quando existem poucas ou nenhuma escrita no servidor, enquanto o bloqueio no nível de linha é essencial para obter bom desempenho quando vários clientes escrevem no servidor simultaneamente. O otimizador do Derby ajusta a escolha automaticamente para os comandos.

Vários níveis de simultaneidade:

O Derby dá suporte aos níveis de isolamento *SERIALIZABLE* (RR), *REPEATABLE* (RS), *READ COMMITTED* (CS) e *READ UNCOMMITTED* (UR).

CS

CS (o nível de isolamento padrão) fornece o melhor equilíbrio entre simultaneidade e consistência em ambientes com vários clientes.

RS

RS fornece menor consistência que RR, mas permite maior simultaneidade.

RR

RR fornece a maior consistência.

UR

UR fornece a máxima simultaneidade, quando são permitidos valores não efetivados nos comandos. Normalmente é utilizado quando se aceita resultados aproximados.

Para obter mais informações deve ser consultado "Tipos e finalidades dos bloqueios nos sistemas Derby" no *Guia do Desenvolvedor do Derby*.

Várias conexões e vários fluxos de execução:

O Derby permite várias conexões simultâneas com o banco de dados, mesmo no modo incorporado. O Derby também é inteiramente *multi-thread*, podendo haver vários fluxos de execução ativos ao mesmo tempo. Entretanto, a semântica do JDBC impõe algumas limitações ao *multi-thread*. Para obter mais informações deve ser consultado o *Guia do Desenvolvedor do Derby*.

Ferramentas administrativas:

O Derby fornece algumas ferramentas e funcionalidades para auxiliar os administradores de bancos de dados, incluindo:

- Verificador de consistência
- Cópia de segurança em linha
- A capacidade de colocar o *log* do banco de dados em uma unidade separada

Estas ferramentas e funcionalidades são discutidas na segunda parte deste guia. Para obter mais informações devem ser vistas as seções desta parte.

O Network Server do Derby

O *Network Server* do Derby fornece conectividade multiusuário para os bancos de dados do Derby, dentro do mesmo sistema ou através da rede. O *Network Server* utiliza o protocolo padrão Arquitetura de Banco de Dados Relacional Distribuído (Distributed Relational Database Architecture/DRDA), para receber e responder os comandos dos clientes. Os bancos de dados são acessados através do *Network Server* do Derby utilizando o *driver* cliente da rede do Derby.

O *Network Server* é uma solução para várias JVMs fazendo conexão com o banco de dados, diferentemente do cenário incorporado, onde somente uma JVM executa como parte do sistema. Quando o Derby está incorporado a um aplicativo de JVM-única, o *driver* de JDBC incorporado chama o banco de dados Derby local. Quando o Derby está incorporado a uma estrutura servidora, o software de conectividade da estrutura servidora fornece dados para vários aplicativos JDBC clientes, através da rede ou da Internet.

Para executar o *Network Server* do Derby, é necessário instalar os seguintes arquivos:

- No lado servidor, instalar `derby.jar` e `derbynet.jar`.
- No lado cliente, instalar `derbyclient.jar`.

Existem várias maneiras de gerenciar o *Network Server* do Derby, incluindo:

- Através da linha de comando
- Utilizando os scripts `.bat` e `.ksh`
- Através da interface *servlet*
- Com seu próprio programa Java (escrito utilizando a API do *Network Server*)
- Definindo as propriedades do *Network Server*

[Utilização do Network Server com aplicativos Derby pré-existent](#)s explica como alterar os aplicativos Java existentes, que atualmente executam utilizando o Derby no modo

incorporado, para utilizar o *Network Server* do Derby.

[Gerenciamento do Network Server do Derby](#) explica como gerenciar o *Network Server* utilizando a linha de comando, inclusive para inicializar e parar.

[Gerenciamento remoto do Network Server do Derby utilizando a interface servlet](#) explica como utilizar a interface *servlet* para gerenciar o *Network Server*.

[Tópicos avançados do Network Server do Derby](#) contém tópicos avançados para os usuários do *Network Server* do Derby.

Devido às diferenças entre os *drivers* de JDBC utilizados, podem ser encontradas diferenças de funcionalidades ao executar o Derby na estrutura de *Network Server*, se comparado com sua execução incorporado a um aplicativo usuário. Para obter uma lista completa das diferenças entre as configurações *Network Server* e incorporado, deve ser consultado [Utilização do Network Server com aplicativos Derby pré-existentes](#).

Servidores incorporados

Como o Derby é escrito em Java, existe grande flexibilidade na escolha da configuração da implementação. Por exemplo, pode ser executado o Derby, a estrutura servidora JDBC, e outro aplicativo, na mesma JVM como um único processo.

Como inicializar o servidor incorporado a partir de um aplicativo

Em uma *thread*, o aplicativo que faz a incorporação inicializa o *driver* de JDBC local para seu próprio acesso.

```
/* a carga do driver cliente inicializa apenas o driver cliente */
Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();
Connection conn = DriverManager.getConnection(
    "jdbc:derby:amostra");
```

Em outra *thread*, o mesmo aplicativo inicializa a estrutura servidora para permitir o acesso remoto. A inicialização da estrutura servidora de dentro do aplicativo permite que o servidor e o aplicativo executem na mesma JVM.

Exemplo de servidor incorporado

O *Network Server* pode ser inicializado automaticamente em outra *thread* quando o Derby é inicializado, definindo a propriedade *derby.drda.startNetworkServer* (consulte [Definição das propriedades do Network Server](#)), ou pode ser inicializado utilizando um programa. O exemplo a seguir mostra como inicializar o *Network Server* utilizando um programa:

```
import org.apache.derby.drda.NetworkServerControl;
import java.net.InetAddress;
NetworkServerControl servidor = new NetworkServerControl
    (InetAddress.getByName("localhost"),1527);
servidor.start(null);
```

O programa que inicializa o *Network Server* pode acessar o banco de dados usando tanto o *driver* incorporado quanto o *driver* cliente da rede. A tentativa da estrutura servidora de inicializar o *driver* de JDBC local é ignorada, porque o mesmo já foi inicializado dentro da JVM do aplicativo. A estrutura servidora simplesmente acessa a instância do Derby que já está inicializada. Não há conflito entre o aplicativo e a estrutura servidora.

O cliente remoto pode então conectar através do *driver* cliente do Derby:

```
String nsURL="jdbc:derby://localhost:1527/amostra";
java.util.Properties props = new java.util.Properties();
props.put("user","usr");
props.put("password","pwd");

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
Connection conn = DriverManager.getConnection(nsURL, props);

/*interagir com o Derby*/
Statement s = conn.createStatement();

ResultSet rs = s.executeQuery(
"SELECT * FROM RESERVAS_HOTEL");
```

Sobre este guia e a documentação do Network Server

Este guia assume que o leitor está familiarizado com as funcionalidades e ajustes do Derby. Antes de ler este guia, primeiro deve-se aprender as funcionalidades básicas do Derby lendo o *Guia do Desenvolvedor do Derby*. Também, uma vez que normalmente os ambientes multiusuário envolvem questões de desempenho e ajuste, deve ser lido o guia *Ajuste do Derby*.

Utilização do Network Server com aplicativos Derby pré-existent

Os aplicativos Java, que atualmente executam usando o Derby no modo incorporado, precisam ser modificados para trabalhar com o *Network Server* do Derby. Os tópicos desta seção discutem estas modificações.

O Network Server e as JVMs

O *Network Server* do Derby é compatível com a Java(TM) 2 Platform, Standard Edition, v 1.3.1 (J2SE), ou mais recente.

Instalação dos arquivos jar requeridos e adições ao caminho de classes

Para utilizar o *Network Server* e o *driver* cliente da rede, devem ser adicionados os seguintes arquivos *jar* ao caminho de classes do servidor:

- derbynet.jar

Este arquivo *jar* contém o código do *Network Server*. É necessário apenas para o processo que inicializa o *Network Server*, em adição aos arquivos *jar* padrão do Derby.

- derby.jar

Este arquivo deverá estar no caminho de classes, para que se possa utilizar qualquer função do *Network Server* do Derby.

- derbyclient.jar

Este arquivo *jar* deverá estar no caminho de classes, para que se possa utilizar o *driver* cliente da rede. Este arquivo *jar* é necessário para a comunicação pelo lado cliente com o *Network Server* utilizando o *driver* cliente da rede do Derby. É necessário estar no caminho de classes do lado cliente para que se possa utilizar o *driver* cliente da rede para acessar o Derby.

O Derby fornece arquivos de script para definir o caminho de classes para trabalhar com o *Network Server*. Os scripts estão localizados no diretório

\frameworks\NetworkServer\bin.

- setNetworkClientCP.bat (Windows)
- setNetworkClientCP.ksh (UNIX)
- setNetworkServerCP.bat (Windows)
- setNetworkServerCP.ksh (UNIX)

Para obter informações sobre como definir o caminho de classes deve ser consultado [Gerenciamento do Network Server do Derby](#) e [Introdução ao Derby](#).

Inicialização do Network Server

Deve ser observado que o *Network Server* deve ser sempre parado de forma adequada após ser utilizado, porque não proceder desta maneira poderá ocasionar efeitos colaterais imprevisíveis, como portas bloqueadas no servidor.

Deve ser utilizado o script *startNetworkServer.bat* para inicializar o *Network Server* nas máquinas com Windows, e o script *startNetworkServer.ksh* para inicializar o *Network Server* nos sistemas UNIX. Estes scripts estão localizados no diretório \$DERBY_HOME/frameworks/NetworkServer/bin, onde \$DERBY_HOME é o diretório onde o Derby foi instalado.

Os comandos de *NetworkServerControl* somente podem ser executados a partir do hospedeiro que inicializou o *Network Server*.

Para inicializar o *Network Server*, deverá ser executado o script apropriado a partir da linha de comando. Por exemplo, no sistema Windows, se o Derby foi instalado no diretório padrão na unidade C, e o caminho de classes foi definido corretamente, deverá ser digitado o seguinte comando:

```
$DERBY_HOME\frameworks\NetworkServer\bin\startNetworkserver.bat
```

O diretório de sistema padrão é o diretório onde o Derby foi inicializado (Para obter mais informações sobre o diretório de sistema padrão, deve ser consultado o *Guia do Desenvolvedor do Derby*).

Dica: Pode ser definido \$DERBY_HOME/frameworks/NetworkServer/bin no caminho (PATH) para encurtar o comando.

Por padrão, o *Network Server* escuta as requisições somente no endereço de retorno (*loopback*), significando que somente aceita conexões do hospedeiro local.

Alteração do script startNetworkServer

O script *startNetworkServer* pode ser modificado de uma das seguintes maneiras:

- Especificando um número de porta diferente do padrão (1527), utilizando a opção `-p <número-da-porta>`, conforme mostrado no exemplo a seguir:

```
java org.apache.derby.drda.NetworkServerControl start -p 1368
```

onde 1368 é o novo número da porta.

- Especificando uma determinada interface (nome de hospedeiro ou endereço de IP) para escutar em outra interface que não a padrão (*localhost*), utilizando a opção `-h`, conforme mostrado no exemplo a seguir:

Lembre-se: Antes de utilizar esta opção deve-se executar sob o gerenciador de segurança do Java e habilitar a autenticação de usuário.

```
java org.apache.derby.drda.NetworkServerControl start -h  
meuhospedeiro -p 1368
```

onde *meuhospedeiro* é o nome do hospedeiro, ou o endereço de IP.

Em todas as interfaces, pode ser especificado um nome de hospedeiro, o endereço de IP, ou 0.0.0.0 para escutar.

Inicialização do Network Server sem usar o script

Se não for desejado utilizar o script *StartNetworkServer*, o *Network Server* poderá ser inicializado usando a linha de comando. A sintaxe para o comando é semelhante a:

```
java org.apache.derby.drda.NetworkServerControl start  
[-h <nome-do-hospedeiro>] [-p <número-da-porta>]
```

Inicialização do Network Server a partir de um aplicativo Java

Deve ser observado que o *Network Server* deve ser sempre parado de forma adequada após ser utilizado, porque não proceder desta maneira poderá ocasionar efeitos colaterais imprevisíveis, como portas bloqueadas no servidor.

Existem duas maneiras de inicializar o *Network Server* a partir de um aplicativo Java.

- Pode ser incluída a seguinte linha no arquivo `derby.properties`:

```
derby.drda.startNetworkServer=true
```

Esta linha inicializa o servidor na porta padrão, 1527, escutando em `localhost` (todas as interfaces).

Para especificar uma porta diferente e uma determinada interface no arquivo `derby.properties`, devem ser incluídas as seguintes linhas, respectivamente:

```
derby.drda.portNumber=1110  
derby.drda.host=meuhospedeiro
```

As propriedades `startNetworkServer` e `portNumber` também podem ser especificadas utilizando o comando Java:

```
java -Dderby.drda.startNetworkServer=true  
-Dderby.drda.portNumber=1110  
-Dderby.drda.host=meuhospedeiro seuAplicativo
```

- Pode ser utilizada a API do *NetworkServerControl* para inicializar o *Network Server* a partir de uma *thread* separada no aplicativo Java:

```
NetworkServerControl servidor = new NetworkServerControl();  
servidor.start (null);
```

Inicialização do Network Server em máquinas Windows com pilha dupla IPv6/Ipv4

É necessário adicionar as seguintes propriedades da JVM ao comando quando o servidor é inicializado em máquinas Windows com pilha dupla IPv6/Ipv4:

```
-Djava.net.preferIPv4Stack=false  
-Djava.net.preferIPv6Addresses=true
```

Parar o Network Server

Se a autenticação de usuário estiver desabilitada, o banco de dados Derby irá parar normalmente quando o *Network Server* for parado. Se a autenticação de usuário estiver habilitada, o banco de dados deverá ser parado explicitamente *antes* de parar o *Network Server*, especificando um nome e senha de usuário do Derby válidos.

O banco de dados pode ser parado diretamente, ou pelo servidor Derby.

- Para parar o *Network Server* utilizando os scripts fornecidos no sistema Windows, é utilizado:

```
stopNetworkServer.bat [-h <nome-do-hospedeiro>] [-p  
<número-da-porta>]
```

- Para parar o *Network Server* utilizando os scripts fornecidos no sistema UNIX, é utilizado:

```
stopNetworkServer.ksh [-h <nome-do-hospedeiro>] [-p  
<número-da-porta>]
```

Estes scripts estão localizados no diretório
\$DERBY_HOME/frameworks/NetworkServer/bin .

Parar utilizando a linha de comando

O *Network Server* pode ser parado através da linha de comando utilizando o seguinte comando:

```
java org.apache.derby.drda.NetworkServerControl  
shutdown [-h <nome-do-hospedeiro>] [-p <número-da-porta>]
```

Parar utilizando a API

Pode ser utilizada a API do *NetworkServerControl* para parar o *Network Server* a partir de um aplicativo Java. Por exemplo:

```
shutdown();
```

Por exemplo, o comando a seguir pára o *Network Server* que está executando na máquina corrente utilizando a porta 1527.

```
NetworkServerControl servidor = new NetworkServerControl();  
servidor.shutdown();
```

Obtenção de informações do sistema

Podem ser obtidas informações sobre o *Network Server*, como a versão e valores das propriedades correntes, informações sobre o Java, e informações sobre o servidor de banco de dados Derby, usando o utilitário **sysinfo**. O utilitário **sysinfo** está disponível a partir de scripts, da linha de comando, da API do *NetworkServerControl*, e através da interface *servlet*.

Os scripts que se seguem estão localizados no diretório
\$DERBY_HOME/frameworks/NetworkServer/bin. Antes de executar estes scripts, é necessário ter certeza que o *Network Server* do Derby foi inicializado.

- Deve ser executado o seguinte script **sysinfo** para obter informações sobre o *Network Server* nos sistemas Windows:

```
sysinfo.bat [-h <nome-do-hospedeiro>] [-p <número-da-porta>]
```

- Deve ser executado o seguinte script **sysinfo** para obter informações sobre o *Network Server* nos sistemas UNIX:

```
sysinfo.ksh [-h <nome-do-hospedeiro>] [<-p número-da-porta>]
```

Obtenção de informações do sistema utilizando a linha de comando

Para executar o **sysinfo** a partir da linha de comando, deve ser executado o seguinte comando enquanto o *Network Server* estiver executando:

```
java org.apache.derby.drda.NetworkServerControl  
sysinfo [-h <nome-do-hospedeiro>] [-p <número-da-porta>]
```

Os comandos administrativos, como

`org.apache.derby.drda.NetworkServerControl sysinfo`, somente podem ser executados no hospedeiro onde o servidor foi inicializado, mesmo que o servidor tenha sido inicializado com a opção `-h`.

Obtenção de informações do sistema utilizando a API

O método **sysinfo** produz as mesmas informações que o comando **sysinfo**. A assinatura deste método é

```
String getSysinfo();
```

Por exemplo:

```
NetworkServerControl serverControl = new NetworkServerControl();  
String myinfo = serverControl.getSysinfo();
```

Este método retorna informações sobre o *Network Server* executando na máquina corrente no número de porta padrão (1527).

Obtenção de informações sobre o Network Server em tempo de execução:

É utilizado o comando **runtimeinfo**, ou o método **getRuntimeInfo**, para obter informações do *Network Server* sobre utilização de memória e sessão corrente, incluindo informações sobre usuário, banco de dados e declaração preparada.

- Para executar **runtimeinfo** a partir da linha de comando:

```
java org.apache.derby.drda.NetworkServerControl runtimeinfo  
[-h <nome-do-hospedeiro>] [<-p número-da-porta>]
```

- O método **getRuntimeInfo** retorna as mesmas informações que o comando **runtimeinfo**. A assinatura do método `getRuntimeInfo` é `String getRuntimeInfo()`. Por exemplo:

```
NetworkServerControl serverControl = new NetworkServerControl();  
String myinfo = serverControl.getRuntimeInfo();
```

Obtenção das propriedades do Network Server utilizando o método **getCurrentProperties**:

O método `getCurrentProperties` é um método Java que pode ser utilizado para obter informações sobre o *Network Server*. Este método retorna um objeto *Properties*, com valores de todas as propriedades *NetServer* na forma como estão atualmente definidas.

A assinatura deste método é:

```
Properties getCurrentProperties();
```


Por exemplo:

```
NetworkServerControl servidor = new NetworkServerControl();
Properties p = servidor.getCurrentProperties();
p.list(System.out);
System.out.println(p.getProperty("derby.drda.host"));
```

Conforme mostrado no exemplo anterior, pode ser feita procura nas propriedades correntes e trabalhar com as propriedades individuais se for necessário, utilizando as várias APIs da classe *Properties*. Também podem ser listadas todas as propriedades utilizando o método *Properties.list()*.

Para obter informações sobre a obtenção de informações do sistema utilizando a interface *servlet*, deve ser consultado [Gerenciamento remoto do Network Server do Derby utilizando a interface servlet](#).

Acesso ao Network Server utilizando o driver cliente da rede

Para se conectar ao *Network Server*, o aplicativo precisa carregar o *driver* e a URL de conexão específica para o *Network Server*. Além disso, caso esteja sendo utilizado autenticação, deverá ser especificado nome de usuário e senha.

O *driver* necessário para acessar o *Network Server* é:

```
org.apache.derby.jdbc.ClientDriver
```

A sintaxe da URL requerida para acessar o *Network Server* é:

```
jdbc:derby://<servidor>[:<porta>]/
<nome-do-banco-de-dados>[:<atributo-da-URL>=<valor> [<...>]]
```

onde <atributo-da-UR> é um atributo de cliente incorporado ou da rede do Derby.

Tabela 1. Propriedades de DataSource de JDBC padrão

Propriedade	Tipo	Descrição	Atributo da URL	Notas
databaseName	String	O nome do banco de dados. Esta propriedade é requerida.	'	Esta propriedade também está disponível quando se usa EmbeddedDataSource.
dataSourceName	String	O nome da fonte de dados.	'	Esta propriedade também está disponível quando se usa EmbeddedDataSource.
description	String	A descrição da fonte de dados.	'	Esta propriedade também está disponível quando se usa EmbeddedDataSource.
user	String	O nome da conta do usuário.	user	O valor padrão é APP. Esta propriedade também está disponível quando se usa EmbeddedDataSource.
password	String	A senha de banco de dados do usuário.	password	Esta propriedade também está disponível quando se usa EmbeddedDataSource.
serverName	String	O nome do hospedeiro, ou o endereço de TCP/IP, onde o servidor está escutando as	'	O valor padrão é "localhost".

Propriedade	Tipo	Descrição	Atributo da URL	Notas
		requisições.		
portNumber	Integer	O número da porta onde o servidor está escutando as requisições.	'	O valor padrão é "1527".

Tabela 2. Propriedades de DataSource específicas do cliente

Propriedade	Tipo	Descrição	Atributo da URL	Notas
traceFile	String	O nome do arquivo para saída do rastreamento. Definir esta propriedade ativa o rastreamento. Consulte Rastreamento do cliente da rede .	<i>traceFile</i>	'
traceDirectory	String	O diretório para saída do rastreamento. Cada conexão envia a saída para um arquivo separado. Definir esta propriedade ativa o rastreamento. Consulte Rastreamento do cliente da rede .	<i>traceDirectory</i>	'
traceLevel	Integer	O nível de rastreamento do cliente, se <i>traceFile</i> ou <i>traceDirectory</i> estiver definido.	<i>traceLevel</i>	O valor padrão é TRACE_ALL.
traceFileAppend	Boolean	Verdade se a saída do rastreamento deverá ser anexada ao arquivo de rastreamento existente.	<i>traceFileAppend</i>	O valor padrão é falso.
securityMechanism	Integer	O mecanismo de segurança. Consulte Segurança do cliente da rede .	<i>securityMechanism</i>	O valor padrão é USER_ONLY_SECURITY.
retrieveMessageText	Boolean	Buscar o texto da mensagem no servidor. É chamado um procedimento armazenado para buscar o texto da mensagem a cada	<i>retrieveMessageText</i>	O valor padrão é verdade.

Propriedade	Tipo	Descrição	Atributo da URL	Notas
		<i>SQLException</i> , devendo iniciar uma nova unidade de trabalho. Esta propriedade deverá ser definida como falsa se não for desejado impacto no desempenho, ou quando iniciando novas unidades de trabalho.		

Tabela 3. Propriedades de DataSource específicas do servidor

Propriedade	Tipo	Descrição	Atributos da URL	Notas
connectionAttributes	String	Definido pela lista de atributos da conexão incorporada do Derby, separados por ponto-e-vírgula.	Vários	Esta propriedade também está disponível quando se usa <i>EmbeddedDataSource</i> . Para obter mais informações sobre os vários atributos de conexão, deve ser consultado o <i>Manual de Referência do Derby</i> .

Deve ser observado que `setAttributesAsPassword`, disponível para o DataSource incorporado, não está disponível para o DataSource cliente.

Segurança do cliente da rede

O cliente da rede do Derby permite selecionar o mecanismo de segurança, especificando o valor da propriedade `securityMechanism`.

A propriedade `securityMechanism` pode ser definida de uma das seguintes maneiras:

- Quando está sendo utilizada a interface `DriverManager`, definindo `securityMechanism` no objeto `java.util.Properties`, antes de chamar a forma do método `getConnection` que inclui o parâmetro `java.util.Properties`.
- Quando está sendo utilizada a interface `DataSource` para criar e instalar seus próprios objetos `DataSource`, chamando o método `DataSource.setSecurityMechanism` após criar o objeto `DataSource`.

A tabela [Mecanismos de segurança suportados pelo cliente da rede do Derby](#) lista os mecanismos de segurança que o cliente da rede do Derby suporta, e o valor da propriedade correspondente a ser especificado para obter o mecanismo de segurança. O mecanismo de segurança padrão é apenas a identificação do usuário, se não for definida a senha. Se for definida a senha, o mecanismo de segurança padrão é tanto a identificação do usuário quanto a senha. O usuário padrão é APP, se não for especificado nenhum outro usuário.

Tabela 4. Mecanismos de segurança suportados pelo cliente da rede do Derby

Mecanismo de segurança	Valor da propriedade <code>securityMechanism</code>	Comentários
Identificação e senha do usuário	ClientDataSource. CLEAR_TEXT_PASSWORD_SECURITY (0x03)	Padrão se a senha estiver definida

Mecanismo de segurança	Valor da propriedade <code>securityMechanism</code>	Comentários
Somente a identificação do usuário	<code>ClientDataSource.USER_ONLY_SECURITY (0x04)</code>	Padrão se a senha não estiver definida
Identificação do usuário e senha criptografados	<code>ClientDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY (0x09)</code>	A criptografia requer a implementação da JCE que suporta o algoritmo de Diffie-Helman com número primo de 32 bytes.

Rastreamento do cliente da rede

O cliente da rede do Derby fornece facilidade de rastreamento, para coletar informações de rastreamento do JDBC e ver os fluxos do protocolo.

Existem várias maneiras de obter saída de rastreamento. Entretanto, a maneira mais fácil de obter saída de rastreamento é utilizando o atributo `traceFile` na URL no `ij`. O exemplo a seguir mostra o envio de todo o rastreamento para o arquivo `trace.out`, a partir de uma sessão `ij`.

```
ij>connect 'jdbc:derby://localhost:1527/meubanco;
create=true;traceFile=trace.out;user=user1;password=secret4me';
```

Implementação do rastreamento do `ClientDataSource`

Para coletar dados de rastreamento ao obter conexões usando `ClientDataSource`, pode ser empregado um dos três métodos a seguir:

- Utilizar o método `setLogWriter(java.io.PrintWriter)` de `ClientDataSource`, e definir `PrintWriter` com um valor não nulo.
- Utilizar o método `setTraceFile(String filename)` de `ClientDataSource`.
- Utilizar o método `setTraceDirectory(String dirname)` de `ClientDataSource` para rastrear cada fluxo de conexão em seu próprio arquivo, em programas que possuem várias conexões.

Implementação do rastreamento do `DriverManager`

Quando se obtém conexões utilizando o `DriverManager`, pode ser empregada uma das duas opções a seguir para habilitar a coleta de informações de rastreamento:

- Utilizar o método `setLogWriter(java.io.PrintWriter)` de `DriverManager`, e definir `PrintWriter` como um valor não nulo.
- Utilizar o atributo da URL `traceFile` ou `traceDirectory` para definir uma destas propriedades, antes de criar a conexão com o método `DriverManager.getConnection()`.

Alteração do nível de rastreamento padrão

O nível de rastreamento padrão é `ClientDataSource.TRACE_ALL`. O nível de rastreamento pode ser escolhido chamando o método `setTraceLevel(int level)`, ou definindo o atributo da URL `traceLevel`:

```
String url = "jdbc:derby://meuhospedeiro.meudominio.com:1528/meubanco" +
";traceFile=/u/user1/trace.out" +
";traceLevel=" +
org.apache.derby.jdbc.ClientDataSource.TRACE_PROTOCOL_FLOWS;
DriverManager.getConnection(url,"user1","secret4me");
```

Tabela 5. Níveis de rastreamento disponíveis e seus valores

Nível de rastreamento	Valor
<code>org.apache.derby.jdbc.ClientDataSource.TRACE_NONE</code>	0x0
<code>org.apache.derby.jdbc.ClientDataSource.TRACE_CONNECTION_CALLS</code>	0x1

Nível de rastreamento	Valor
org.apache.derby.jdbc.ClientDataSource.TRACE_STATEMENT_CALLS	0x2
org.apache.derby.jdbc.ClientDataSource.TRACE_RESULT_SET_CALLS	0x4
org.apache.derby.jdbc.ClientDataSource.TRACE_DRIVER_CONFIGURATION	0x10
org.apache.derby.jdbc.ClientDataSource.TRACE_CONNECTS	0x20
org.apache.derby.jdbc.ClientDataSource.TRACE_PROTOCOL_FLOWS	0x40
org.apache.derby.jdbc.ClientDataSource.TRACE_RESULT_SET_META_DATA	0x80
org.apache.derby.jdbc.ClientDataSource.TRACE_PARAMETER_META_DATA	0x100
org.apache.derby.jdbc.ClientDataSource.TRACE_DIAGNOSTICS	0x200
org.apache.derby.jdbc.ClientDataSource.TRACE_XA_CALLS	0x800
org.apache.derby.jdbc.ClientDataSource.TRACE_ALL	0xFFFFFFFF;

Para especificar mais de um nível de rastreamento, deve ser utilizada uma das seguintes técnicas:

- Utilizar operadores OR bit-a-bit (|) entre dois ou mais valores de rastreamento. Por exemplo, para rastrear os fluxos do protocolo e chamadas de conexão deve ser especificado o seguinte valor para traceLevel:

```
TRACE_PROTOCOL_FLOWS | TRACE_CONNECTION_CALLS
```

- Utilizar o operador de complemento bit-a-bit (~) com o valor de rastreamento para especificar todos, exceto um determinado rastreamento. Por exemplo, para rastrear tudo, exceto os fluxos do protocolo, deve ser especificado o seguinte valor para traceLevel:

```
~TRACE_PROTOCOL_FLOWS
```

Exemplos de driver cliente da rede

Os exemplos a seguir especificam os atributos usuário e senha da URL. Para habilitar a autenticação de usuário, a propriedade *derby.connection.requireAuthentication* deve ser definida como verdade, senão o Derby não requer o nome de usuário e a senha. Em produtos multiusuário, esta propriedade normalmente é definida no arquivo do sistema *derby.properties* no servidor, uma vez que este está em um ambiente confiável. Abaixo está mostrado um arquivo *derby.properties* de amostra em conformidade com estes exemplos:

```
derby.connection.requireAuthentication=true
derby.authentication.provider=BUILTIN
derby.user.judy=no12see
```

Exemplo 1

Neste exemplo a conexão é feita usando o nome de servidor padrão, *localhost*, a porta padrão, *1527*, e o banco de dados *amostra*.

```
jdbc:derby://localhost:1527/amostra;user=judy;password=no12see
```

Exemplo 2

Neste exemplo são especificados tanto atributos do Derby quanto atributos do *driver* cliente da rede:

```
jdbc:derby://localhost:1527/sample;create=true;user=judy;
password=no12see
```

Exemplo 3

Neste exemplo a conexão é feita usando o nome de servidor padrão, *localhost*, a porta padrão, *1527*, e incluindo o caminho na parte do nome do banco de dados da URL.

```
jdbc:derby://localhost:1527/c:/meudiretorio/meubanco;user=judy;
password=nol2see
```

Exemplo 4

Neste exemplo é mostrado como utilizar o *driver* cliente da rede para conectar o cliente da rede ao *Network Server*.

```
String databaseURL = "jdbc:derby://localhost:1527/amostra";
// Carregar a classe do driver cliente da rede do Derby
Class.forName("org.apache.derby.jdbc.ClientDriver");
// Definir as propriedades usuário e senha
Properties properties = new Properties();
properties.put("user", "judy");
properties.put("password", "nol2see");
// Obter a conexão
Connection conn = DriverManager.getConnection(databaseURL, properties);
```

Acesso ao Network Server utilizando o Universal Driver do DB2

Pode ser utilizado o *Universal Driver* do DB2, em vez do *driver* cliente da rede do Derby, para conectar ao *Network Server*. O aplicativo necessita carregar o *driver*, e a URL de conexão específica para o *Network Server*. Além disso, é especificado o nome do usuário e a senha. Se a autenticação não estiver configurada, poderá ser utilizado qualquer valor para o nome do usuário e a senha. O *driver* utilizado para acessar o *Network Server* é:

```
com.ibm.db2.jcc.DB2Driver
```

Para o *Universal Driver* do DB2 poder ser utilizado, devem estar presentes no caminho de classes os seguintes arquivos:

- db2jcc.jar
- db2jcc_license_c.jar

A sintaxe da URL requerida para acessar o *Network Server* é:

```
jdbc:derby:net://<servidor>[:<porta>]/
<nome-do-banco-de-dados>[:<atributo-da-URL-do-Derby>=<valor> [:...]]
[:<atributo do Universal Driver>=<valor>; [...;]]
```

Após especificar o nome do banco de dados e os atributos, podem ser incluídos atributos para o *driver* de JDBC do DB2. Deve ser incluído um ponto-e-vírgula após o último atributo do *Universal Driver*.

servidor

O nome da máquina onde o servidor está executando. Pode ser o nome da máquina (por exemplo, *buffy*) ou o endereço de IP, por exemplo, *158.58.62.225*.

Note: A menos que o *Network Server* tenha sido inicializado com a opção *-h* ou com a propriedade *derby.drda.host* definida, este valor deverá ser *localhost*.

porta

O número da porta onde o servidor está escutando. O valor padrão é *1527*.

nome do banco de dados

O nome do banco de dados para a conexão. O nome do banco de dados pode ter no máximo 18 caracteres. Para incluir informações de caminho no nome do banco de dados, devem ser usadas aspas ("). Como alternativa, a informação sobre o caminho pode ser especificada definindo a propriedade *derby.system.home* no arquivo *derby.properties* ou no ambiente Java ao inicializar o *Network Server*. Para obter informações sobre como definir a pasta base do sistema (*home*), deve ser consultado o *Guia do Desenvolvedor do Derby*.

atributo da URL do derby=valor

Atributos opcionais da URL de conexão com o banco de dados suportados pelo

Derby. Para obter mais informações deve ser consultado o *Guia do Desenvolvedor do Derby*.

atributo do Universal Driver=valor

Atributos opcionais da URL de conexão com o banco de dados suportados pelo *DB2 Universal JDBC Driver*.

O *DB2 JDBC Universal Driver* requer que sejam definidos os atributos usuário e senha do *Universal Driver* como valores não nulos.

Estão disponíveis os seguintes atributos do *DB2 Universal JDBC Driver* ao executar o *Network Server*.

user

O nome do usuário (requerido pelo *Universal JDBC Driver*).

password

Senha do usuário (requerido pelo *Universal JDBC Driver*).

portNumber

O número da porta TCP/IP onde o *Network Server* escuta as requisições de conexão para esta fonte de dados. O valor padrão é 1527.

retrieveMessagesFromServerOnGetMessage

Mostra as mensagens de erro do servidor.

readOnly

Cria uma conexão de leitura apenas. O valor padrão é falso.

logWriter

Um fluxo de saída caractere. Todas as mensagens de *log* e de rastreamento imprimem para a propriedade *logWriter*.

traceLevel

Especifica a granularidade das mensagens de rastreamento na propriedade *logWriter*.

traceFile

Fornece uma localização de arquivo explícita para a saída de rastreamento.

securityMechanism

Indica o tipo de mecanismo de segurança utilizado.

deferPrepares

Controla o momento em que as declarações preparadas são preparadas fisicamente no servidor de banco de dados. O valor padrão é verdade.

Informações sobre o Universal Driver System

O *Network Server* do Derby é compatível com o *DB2 JDBC Universal Driver* versão 2.4, ou maior.

Exemplos do Universal Driver do DB2:

Exemplo 1

Neste exemplo a conexão é feita usando o nome de servidor padrão, *localhost*, a porta padrão, *1527*, e o banco de dados *amostra*. São especificados os atributos *user*, *password* e *retrieveMessagesFromServerOnGetMessage* da URL. É necessário definir os atributos *user* e *password* do *Universal Driver*.

```
jdbc:derby:net://localhost:1527/amostra:user=judy;password=nol2see;
retrieveMessagesFromServerOnGetMessage=true;
```

Exemplo 2

Neste exemplo são especificados tanto atributos do Derby quanto atributos do *Universal Driver*.

```
jdbc:derby:net://localhost:1527/sample;create=true:user=judy;
password=nol2see;retrieveMessagesFromServerOnGetMessage=true;
```

Exemplo 3

Neste exemplo a conexão é feita usando o nome de servidor padrão, *localhost*, a porta padrão, *1527*, e incluindo o caminho na parte do nome do banco de dados da URL. O nome do banco de dados deve ser delimitado por aspas, e não podem ser especificados atributos do Derby na URL.

```
jdbc:derby:net://localhost:1527/"c:/meudiretorio/meubanco":user=judy;
password=nol2see;retrieveMessagesFromServerOnGetMessage=true;
```

Exemplo 4

Este exemplo é um fragmento de um programa exemplo que conecta ao *Network Server* utilizando o *Universal Driver*.

```
String databaseURL = "jdbc:derby:net://localhost:1527/sample";
// Carga da classe do IBM JDBC Universal Driver
Class.forName("com.ibm.db2.jcc.DB2Driver");
// Definir as propriedades nome do usuário e senha
Properties properties = new Properties();
properties.put("user", "APP");
properties.put("password", "APP");
properties.put("retrieveMessagesFromServerOnGetMessage", "true");
// Obter a conexão
Connection conn = DriverManager.getConnection(databaseURL, properties);
```

Acesso ao Network Server utilizando DataSource

As fontes de dados do *driver* cliente da rede do Derby.org.apache.derby.jdbc.ClientDataSource e org.apache.derby.jdbc.ClientConnectionPoolDataSource são suportadas pelo *Network Server*.

Exemplo de acesso utilizando DataSource

O exemplo a seguir utiliza org.apache.derby.jdbc.ClientDataSource para acessar o *Network Server*.

```
public static javax.sql.DataSource getDS(String database, String user,
String password) throws SQLException
{
    org.apache.derby.jdbc.ClientDataSource ds =
        new org.apache.derby.jdbc.ClientDataSource();

    // O nome do banco de dados pode incluir atributos da URL do Derby
    ds.setDatabaseName(database);

    if (user != null)
        ds.setUser(user);
    if (password != null)
        ds.setPassword(password);

    // O hospedeiro onde o Network Server está executando
    ds.setServerName("localhost");

    // Porta onde o Network Server está escutando
    ds.setPortNumber(1527);
}
```



```
return ds;
}
```

O programa pode então estabelecer a conexão:

```
javax.sql.DataSource ds = getDS("meubanco;create=true", null, null);
// Nota: o usuário e a senha são requeridos na conexão
Connection conn = ds.getConnection("usr2", "pass2");
```

XA e o Network Server

Tanto o *driver* incorporado quanto o *Network Server* do Derby fornecem suporte a XA. O *Network Server* fornece suporte a DRDA no nível 7. Os clientes da DRDA que suportam XAMGR, como o cliente de rede do Derby, podem enviar requisições XA para o *Network Server*.

Utilização de XA com o driver cliente da rede

O suporte a XA para o *Network Server* pode ser acessado utilizando a interface *DataSource XA* do *driver* cliente da rede (`org.apache.derby.jdbc.ClientXADataSource`).

O exemplo a seguir mostra como obter uma conexão XA com o *driver* cliente da rede:

```
import org.apache.derby.jdbc.ClientXADataSource;
import javax.sql.XAConnection;
...

XAConnection xaConnection = null;
Connection conn = null;

String driver = "org.apache.derby.jdbc.ClientDataSource";
ClientXADataSource ds = new ClientXADataSource();

ds.setDatabaseName ("amostra;create=true");
ds.setServerName("localhost");
ds.setPortNumber(1527);
Class.forName(driver);
xaConnection = ds.getXAConnection("auser", "shhhh");
conn = xaConnection.getConnection();
```

Utilização das ferramentas do Derby com o Network Server

As ferramentas *ij* e *dblook* do Derby trabalham no modo incorporado e no modo cliente/servidor.

Utilização da ferramenta *ij* do Derby com o Network Server

Para utilizar a ferramenta *ij* com o *driver* cliente da rede:

1. Inicialize o *ij* de uma das seguintes maneiras:
 - a. Utilizando um script.

Execute o script *ij.bat* nos sistemas Windows, ou o script *ij.ksh* nos sistemas UNIX. Estes scripts estão localizados no diretório `$DERBY_HOME`.

- b. Executando a ferramenta *ij* a partir da linha de comando.

```
java org.apache.derby.tools.ij
```

2. Conecte especificando o URL:

```
ij> CONNECT 'jdbc:derby://localhost:1527/sample'
USER 'judy' PASSWORD 'no12see';
```

Para obter exemplos adicionais de URLs deve ser consultado [Exemplos de driver cliente da rede](#).

Utilização da ferramenta dblook do Derby com o Network Server

Para utilizar a ferramenta *dblook* com o *driver* cliente da rede é necessário ter certeza que o *Network Server* está executando (consulte [Inicialização do Network Server](#)), e depois incluir os atributos de conexão do Derby e do *driver* cliente da rede como parte da URL do banco de dados.

```
java org.apache.derby.tools.dblook -d
'jdbc:derby://localhost:1527/amostra;
user=user1;password=secret4me';
```

Diferenças entre executar o Derby no modo incorporado e utilizando o Network Server

Esta seção descreve as diferenças entre a execução do Derby no modo incorporado e utilizando o *Network Server*. Deve ser observado que podem existir diferenças não documentadas, ainda não identificadas.

Diferenças entre os drivers cliente incorporado e cliente da rede

A seguir estão as diferenças conhecidas existentes entre o *driver* incorporado e o *driver* cliente da rede do Derby. Deve ser observado que podem existir diferenças não documentadas, ainda não identificadas. Algumas diferenças presentes no cliente da rede poderão ser modificadas em versões futuras, para corresponder às funcionalidades do *driver* incorporado.

- As mensagens de erro e *SQLStates* podem diferir entre os *drivers* cliente da rede e incorporado. Alguns *SQLStates* podem ser nulos ao utilizar o cliente da rede, em particular nos erros de conversão de dados.
- As exceções e advertências SQL múltiplas somente retornam o *SQLState* da primeira exceção, quando se utiliza o cliente da rede. Os textos das exceções adicionais são anexados ao texto da primeira exceção. Consulte [Diferenças nas mensagens de erro](#).
- Para utilizar identificação do usuário e senha criptografados, é necessário possuir a Extensão de Criptografia do Java da IBM (Java Cryptography Extension/JCE) versão 1.2.1, ou mais recente.

Conjuntos de resultados atualizáveis

No Derby, as funcionalidades dos conjuntos de resultados atualizáveis em um ambiente servidor são semelhantes às do ambiente incorporado, exceto pelas seguintes diferenças:

- O cliente da rede requer a existência de pelo menos uma coluna da tabela de destino na lista de seleção. Por exemplo, a seguinte declaração falha em um ambiente servidor:

```
select 1, 2 from t1 for update of c11
```

O *driver* cliente da rede olha as duas colunas da lista de seleção, e não consegue determinar a tabela de destino para atualizar/excluir olhando no metadado da coluna. Esta exigência não é necessária no ambiente incorporado.

- O *driver* incorporado permite mudanças no nome da declaração quando existe um conjunto de resultados aberto no objeto da declaração. Isto não é suportado no ambiente servidor.

Outras diferenças entre os conjuntos de resultados atualizáveis em ambiente servidor e incorporado podem ser encontradas na tabela a seguir.

Tabela 6. Comparação entre as funcionalidades dos conjuntos de resultados

atualizáveis nos ambientes servidor e incorporado

Ambiente incorporado	Ambiente servidor
Suportado <i>updateString</i> nos tipos de dado SMALLINT, INTEGER, BIGINT, DECIMAL.	Não suportado
Suportado <i>updateBytes</i> nos tipos de dado CHAR, VARCHAR, LONG VARCHAR.	Não suportado
Suportado <i>updateTime</i> no tipo de dado TIMESTAMP.	Não suportado
Suportado <i>updateObject</i> com valores nulos.	Não suportado
Suportado <i>updateClob</i> e <i>updateBlob</i> .	Não suportado

Diferenças nas mensagens de erro

O *Network Server* relata apenas a primeira mensagem de erro ou de advertência, se ocorrerem vários erros ou advertências para uma determinada declaração. Por exemplo:

```
ij> create table ai (x int, y int generated always as identity
      (increment by 200000000));
ij> insert into ai (x) values (1),(2),(3),(4),(5),(6),(7),
      (8),(9),(10),(11),(12),(13),(14),(15),(16),(17),(18),(19);
```

O *Network Server* gera a seguinte mensagem de erro, com a mensagem de exceção anexada à mensagem de erro:

```
ERROR 42Z24: Overflow occurred in identity for column 'Y' in table 'AI':
SQLSTATE: 22003: The resulting value is outside the range
for the data type INTEGER.
```

Entretanto, o *driver* incorporado do Derby gera duas exceções SQL:

```
ERROR 42Z24: Overflow occurred in identity for column 'Y' in table 'AI'.
```

```
ERROR 22003: The resulting value is outside the range for the data type
INTEGER.
```

Isto se deve ao fato do *driver* cliente da rede somente relatar uma *SQLException* ou *SQLWarning* por declaração.

Diferenças na autenticação do usuário

Quando se executa o Derby no modo incorporado, ou quando se utiliza o *Network Server* do Derby, pode-se habilitar ou desabilitar a autenticação do usuário no lado servidor. Entretanto, quando se utiliza o *Network Server* o mecanismo de segurança padrão (CLEAR_TEXT_PASSWORD) requer o fornecimento tanto do nome do usuário quanto da senha.

Além do mecanismo de segurança padrão que emprega nome de usuário e senha, *org.apache.derby.jdbc.ClientDataSource.CLEAR_TEXT_PASSWORD_SECURITY*, o *Network Server* do Derby suporta os seguintes mecanismos de segurança:

- *Identificação do usuário*
(*org.apache.derby.jdbc.ClientDataSource.USER_ONLY_SECURITY*)
Quando se utiliza este mecanismo, deve ser especificada apenas a propriedade usuário.
- *Identificação do usuário criptografada e senha criptografada*
(*org.apache.derby.jdbc.ClientDataSource.ENCRYPTED_USER_*

AND_PASSWORD_SECURITY)

Quando se utiliza este mecanismo, tanto a senha quanto a identificação do usuário são criptografadas.

O nome de usuário especificado na conexão será o mesmo nome do esquema padrão para a conexão, caso exista um esquema com este nome. Para obter mais informações sobre nomes de esquemas e de usuários, deve ser consultado o *Guia do Desenvolvedor do Derby*.

Se for especificado qualquer outro mecanismo de segurança, será recebida uma exceção.

Para mudar o mecanismo padrão, pode ser especificado outro mecanismo de segurança como uma propriedade ou na URL (utilizando o atributo *securityMechanism*), ao estabelecer a conexão.

O Network Server e a autenticação de usuário, quando a mesma está habilitada no Derby:

Quando a autenticação de usuário está habilitada no Derby, pode ser utilizado o mecanismo de segurança padrão (nome de usuário e senha), ou pode ser especificado que o mecanismo de segurança deve ser nome de usuário e senha criptografados.

O Network Server e a autenticação de usuário, quando a mesma está desabilitada no Derby:

Quando a autenticação de usuário está desabilitada no Derby, pode ser utilizada qualquer uma das opções de mecanismo de segurança.

Deve ser especificado o nome do usuário e a senha em todos os mecanismos de segurança, exceto *USER_ONLY_SECURITY*. Entretanto, como a autenticação de usuário está desabilitada no servidor Derby, o nome de usuário e senha fornecidos não precisam ser reconhecidos como válidos pelo Derby.

Habilitação do mecanismo de segurança de identificação do usuário e senha criptografados:

Para utilizar o mecanismo de segurança de identificação do usuário e senha criptografados, é necessária a JCE (Java Cryptography Extension) da IBM, versão 1.2.1 ou mais recente, que pode ser utilizada com qualquer versão da Java(™) 2 Platform, Standard Edition, Versão 1.2 (J2SE) da IBM ou da Sun.

O *IBM Developer Kit* para a *Plataforma Java 1.4*, ou mais recente, vem com a JCE da IBM, portanto não é necessário instalar a JCE da IBM em separado. Caso se tenha uma versão mais antiga do *IBM Developer Kit* para a plataforma Java, ou outra ferramenta de desenvolvimento de software, devem ser efetuadas as seguintes etapas:

1. Copiar os seguintes arquivos *jar* da JCE da IBM para o diretório *jre/lib/ext* do local de instalação do SDK da IBM:
 - *ibmjceprovider.jar*
 - *ibmjcefw.jar*
 - *ibmpkderby.jar*
 - *ibmpkcs11.jar*
2. Modificar o arquivo *java.security* no diretório *jre/lib/security*. Na seção que lista os provedores (e ordem de preferência), substituir o texto por:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

Note: Se a JCE da IBM estiver sendo instalada no *Java Development Kit* da Sun,

as duas linhas devem ser especificadas na ordem mostrada.

3. Para utilizar o mecanismo de identificação do usuário e senha criptografados durante a conexão utilizando o cliente da rede, deve ser especificado `securityMechanism` na propriedade da conexão.

Se for inicializado um banco de dados criptografado no *Network Server*, os usuários poderão se conectar ao banco de dados sem fornecer a `bootPassword`. A primeira conexão com o banco de dados deverá fornecer a `bootPassword`, mas as conexões subsequentes não precisarão fornecer. Para interromper o acesso ao banco de dados criptografado, é utilizada a opção `shutdown=true` para parar o banco de dados.

Definição do número da porta

Por padrão, o Derby utilizando o *Network Server* escuta na porta TCP/IP de número 1527. Se for desejado utilizar um número de porta diferente, esta pode ser especificada na linha de comando ao inicializar o *Network Server*. Por exemplo:

```
java org.apache.derby.drda.NetworkServerControl start -p 1088
```

1. Entretanto, é melhor especificar o número da porta utilizando um dos seguintes métodos:
 - Alterar os scripts `startNetworkServer.bat` ou `startNetworkServer.ksh`
 - Utilizar a propriedade `derby.drda.portNumber` em `derby.properties`

Para obter mais informações deve ser consultado [Inicialização do Network Server](#).

Gerenciamento do Network Server do Derby

O *Network Server* do Derby pode executar como um servidor autônomo, com o Derby como uma parte incorporada do aplicativo.

O *Network Server* pode ser gerenciado utilizando scripts do interpretador de comandos (*shell*), a linha de comando, ou a API do *Network Server*. O *Network Server* também pode ser gerenciado remotamente utilizando a interface *servlet*. Para obter mais informações sobre como inicializar e parar o *Network Server* utilizando a interface *servlet*, deve ser consultado [Gerenciamento remoto do Network Server do Derby utilizando a interface servlet](#).

Visão geral

O *Network Server* do Derby é inicializado utilizando a linha de comando, ou utilizando a API do servidor Derby (O Derby fornece scripts para inicializar o servidor a partir da linha de comando). Antes de inicializar o servidor, provavelmente deverão ser definidas algumas propriedades do Derby e do *Network Server*.

Utilização da API do NetworkServerControl

É necessário criar uma instância da classe *NetworkServerControl* quando se utiliza a API. Esta classe possui dois métodos construtores:

Note: Antes de habilitar a conexão a partir de outros sistemas, deve haver certeza de estar executando sob o gerenciador de segurança.

- `NetworkServerControl()`

Este método construtor cria uma instância que escuta na porta padrão (1527), ou na porta definida pela propriedade `derby.drda.portNumber`. Também escuta no hospedeiro definido pela propriedade `derby.drda.host`, ou no endereço de retorno (*loopback*) se a propriedade não estiver definida. Este é o construtor

padrão; não permite conexão remota. Equivale a chamar `NetworkServerControl(InetAddress.getByName("localhost"),1527)`, se nenhuma propriedade estiver definida.

- `NetworkServerControl (InetAddress address, int portNumber)`

Este método construtor cria uma instância que escuta no número da porta especificado e no endereço especificado. `InetAddress` é passado para `ServerSocket`. `NULL` é um valor de endereço inválido. Os exemplos a seguir mostram como deve ser feito para permitir o *Network Server* aceitar conexões de outros hospedeiros:

```
// Aceitar conexões de outros hospedeiro em um sistema IPv4
NetworkServerControl serverControl =
    new NetworkServerControl(InetAddress.getByName("0.0.0.0"),1527);
```

```
// Aceitar conexões de outros hospedeiro em um sistema IPv6
NetworkServerControl serverControl =
    new NetworkServerControl(InetAddress.getByName(":::"),1527);
```

Definição das propriedades do Network Server

As propriedades do *Network Server* podem ser especificadas de três maneiras:

- Na linha de comando
- Nos arquivos `.bat` e `.ksh` (definindo as propriedades executando `java -D`)
- No arquivo `derby.properties`.

As propriedades presentes na linha de comando, ou nos arquivos `.bat` e `.ksh`, têm precedência sobre as propriedades no arquivo `derby.properties`. Os argumentos incluídos nos comandos emitidos na linha de comando têm precedência sobre os valores das propriedades.

derby.drda.host

Faz o *Network Server* escutar na interface de rede especificada. Esta propriedade permite que várias instâncias do *Network Server* executem em uma mesma máquina, cada uma utilizando sua própria combinação única de hospedeiro:porta. É necessário definir o hospedeiro com conexões remotas habilitadas. Por padrão, o *Network Server* escuta apenas no endereço de retorno (*loopback*). Se a propriedade for definida como `0.0.0.0`, o *Network Server* escutará em todas as interfaces. Deve haver certeza de estar executando sob o gerenciador de segurança, e que a autorização de usuário está habilitada, antes de habilitar as conexões remotas com esta propriedade.

Sintaxe

```
derby.drda.host=nome-do-hospedeiro
```

Padrão

Se o nome do hospedeiro não for especificado, o *Network Server* escutará no endereço de retorno da máquina corrente (`localhost`).

Exemplo

```
derby.drda.host=meuhospedeiro
```

Estático ou dinâmico

Estático. O *Network Server* deverá ser reinicializado para a alteração produzir efeito.

derby.drda.keepAlive

Indica se `SO_KEEPALIVE` estará habilitado nos soquetes. O mecanismo *keepAlive* é utilizado para detectar clientes desconectados inesperadamente. É solicitado do cliente uma prova de que está vivo (*keepalive probe*), se passar um longo tempo (por padrão, mais de duas horas) sem que nenhum dado seja enviado ou recebido. A propriedade `drda.keepAlive` é utilizada para detectar e limpar conexões de clientes em máquinas desligadas, ou que se desconectaram inesperadamente.

Se a propriedade for definida como falsa, o Derby não tentará limpar clientes desconectados. O mecanismo *keepAlive* deverá ser desabilitado se os clientes precisarem continuar o trabalho sem reconectar, mesmo após terem se desconectado da rede por algum tempo. Para desabilitar os testes de *keepAlive* nas conexões do *Network Server*, esta propriedade deverá ser definida como falsa.

Sintaxe

```
derby.drda.keepAlive=[true|false]
```

Padrão

True.

Exemplo

```
derby.drda.keepAlive=false
```

Estático ou dinâmico

Estático. O *Network Server* deverá ser reinicializado para a alteração produzir efeito.

derby.drda.logConnections

Indica se as conexões serão registradas. Também controla o registro do número da conexão. Se o rastreamento do número da conexão estiver habilitado, irá tanto para o arquivo `derby.log` quanto para a console do servidor de rede.

Sintaxe

```
derby.drda.logConnections=[true|false]
```

Padrão

Falso.

Exemplo

```
derby.drda.logConnections=true
```

Estático ou dinâmico

Dinâmico. O valor no sistema pode ser alterado utilizando um comando ou a interface *servlet*, após o *Network Server* ter sido inicializado.

derby.drda.maxThreads

A propriedade `derby.drda.maxThreads` é utilizada para definir o número máximo de *threads* de conexão que o *Network Server* irá disponibilizar. Se todas as *threads* de conexão estiverem sendo usadas no momento, e o *Network Server* já tiver disponibilizado o número máximo de *threads*, as *threads* serão compartilhadas utilizando a propriedade `derby.drda.timeslice` para determinar quando as sessões serão alternadas.

Sintaxe

```
derby.drda.maxThreads=número-de-threads
```

Padrão

0

Exemplo

```
derby.drda.maxThreads=50
```

Estático ou dinâmico

Estático. O *Network Server* deverá ser reinicializado para a alteração produzir efeito.

derby.drda.minThreads

A propriedade *derby.drda.minThreads* é utilizada para definir o número mínimo de *threads* de conexão que o *Network Server* irá disponibilizar. Por padrão, as *threads* de conexão são disponibilizadas de acordo com a necessidade.

Sintaxe

```
derby.drda.minThreads=número-de-threads
```

Padrão

0

Exemplo

```
derby.drda.minThreads=10
```

Estático ou dinâmico

Static. You must restart the Network Server for changes to take effect.

derby.drda.portNumber

Indica o número da porta a ser utilizado.

Sintaxe

```
derby.drda.portNumber=número-da-porta
```

Padrão

Se não for especificado nenhum número de porta, o valor padrão é 1527.

Exemplo

```
derby.drda.portNumber=1110
```

Estático ou dinâmico

Estático. O *Network Server* deverá ser reinicializado para a alteração produzir efeito.

derby.drda.startNetworkServer

A propriedade *derby.drda.startNetworkServer* é utilizada para simplificar a incorporação do *Network Server* ao aplicativo Java. Quando a propriedade *derby.drda.startNetworkServer* está definida como verdade, o *Network Server* inicializa automaticamente quando o Derby é inicializado (neste contexto, o Derby é inicializado quando o *driver* incorporado é carregado). Somente pode ser inicializado um *Network Server* em uma mesma JVM.

NOTA: Se o *Network Server* for inicializado com esta propriedade definida como verdade, o *Network Server* irá parar quando o aplicativo terminar, ou se for parado de

outra maneira (por exemplo, utilizando a API do Java, a interface de linha de comando, ou parando o sistema Derby), o que ocorrer primeiro.

Sintaxe

```
derby.drda.startNetworkServer=[true | false]
```

Padrão

Falso.

Exemplo

```
derby.drda.startNetworkServer=true
```

Estático ou dinâmico

Estático. O *Network Server* deverá ser parado e o Derby reinicializado para a alteração produzir efeito.

derby.drda.timeslice

A propriedade *derby.drda.timeslice* é utilizada para definir o número de milissegundos que cada conexão irá utilizar antes de alternar para outra conexão. Esta propriedade é relevante apenas quando a propriedade *derby.drda.maxThreads* está definida com um valor maior que zero.

Sintaxe

```
derby.drda.timeslice=milissegundos
```

Padrão

0

Exemplo

```
derby.drda.timeslice=2000
```

Estático ou dinâmico

Estático. O *Network Server* deverá ser reinicializado para a alteração produzir efeito.

derby.drda.traceAll

Habilita o rastreamento de todas as sessões.

Sintaxe

```
derby.drda.traceAll=[true|false]
```

Padrão

Falso.

Exemplo

```
derby.drda.traceAll=true
```

Estático ou dinâmico

Dinâmico. O valor no sistema pode ser alterado utilizando um comando ou a interface *servlet*, após o *Network Server* ter sido inicializado.

derby.drda.traceDirectory

Indica o local dos arquivos de rastreamento.

Sintaxe

```
derby.drda.traceDirectory=diretório-dos-arquivos-de-rastreamento
```

Padrão

Se a propriedade *derby.system.home* tiver sido definida, este é o valor padrão. Caso contrário, o padrão é o diretório corrente.

Exemplo

```
derby.drda.traceDirectory=c:/Derby/rastreamento
```

Estático ou dinâmico

Dinâmico. O valor no sistema pode ser alterado utilizando um comando ou a interface *servlet*, após o *Network Server* ter sido inicializado.

Verificação da inicialização

Para verificar se o *Network Server* do Derby está executando no momento, é utilizado o comando *ping*.

O comando *ping* pode ser utilizado das seguintes maneiras:

- Através do script *NetworkServerControl.bat* nos sistemas Windows, ou do script *NetworkServerControl.ksh* nos sistemas UNIX, com o comando **ping**. Por exemplo:

```
NetworkServerControl ping [-h <nome-do-hospedeiro>] [-p  
<número-da-porta>]
```

- Através do comando *NetworkServerControl*:

```
java org.apache.derby.drda.NetworkServerControl  
ping [-h <nome-do-hospedeiro>] [-p <número-da-porta>]
```

- Através da API do *NetworkServerControl* para verificar a inicialização a partir de um aplicativo Java:

```
ping();
```

O exemplo a seguir utiliza um método para verificar a inicialização. Este método tenta a verificação durante um determinado número de segundos:

```
private static boolean isServerStarted(NetworkServerControl server, int  
ntries)  
{  
    for (int i = 1; i <= ntries; i ++)  
    {  
        try {  
            Thread.sleep(500);  
            server.ping();  
            return true;  
        }  
        catch (Exception e) {  
            if (i == ntries)  
                return false;  
        }  
    }  
    return false;  
}
```

Gerenciamento remoto do Network Server do Derby utilizando a interface servlet

Pode ser utilizada a interface *servlet* para gerenciar o *Network Server* remotamente. Para utilizar a interface *servlet*, a *servlet* deverá estar registrada no servidor Web, e o servidor Web deverá ter conhecimento de *derby.system.home*.

O arquivo *Web Application Archive* (WAR), *derby.war*, para o *Network Server* do Derby, está disponível em *\$DERBY_HOME/lib*. Este arquivo registra a *servlet* do *Network Server* no caminho relativo */derbynet*. Para obter instruções sobre como instalar este arquivo, deve ser consultada a documentação do servidor de aplicativos.

Por exemplo, se *derby.war* for instalado no *WebSphere Application Server* com o contexto raiz *derby*, a URL do servidor será:

```
http://<servidor>[:porta]/derby/derbynet
```

Note: A máquina *servlet* não faz parte do *Network Server*.

A *servlet* aceita os seguintes parâmetros de configuração opcionais:

portNumber

Especifica o número da porta a ser utilizada pelo *Network Server*.

startNetworkServerOnInit

Especifica que o *Network Server* deverá ser inicializado quando a *servlet* for inicializada.

tracingDirectory

Especifica o local dos arquivos de rastreamento. Se o diretório de rastreamento não for especificado, os arquivos de rastreamento serão colocados em *derby.system.home*.

Esta seção descreve as páginas da *servlet*.

Página de inicialização

A página de inicialização é utilizada para inicializar o servidor.

Além de inicializar o *Network Server*, a página de inicialização pode ser utilizada para realizar as seguintes ações:

- Habilitar o *log* quando o servidor for inicializado.
- Habilitar o rastreamento de todas as sessões quando o servidor for inicializado.

Página de execução

Se o *Network Server* estiver executando (inicializado pela inicialização da *servlet*, ou de alguma outra forma), é mostrada a página de execução. A página de execução indica se o *log* está habilitado ou desabilitado, se o rastreamento está habilitado ou desabilitado, e se o rastreamento estiver habilitado indica para qual sessão.

A página de execução pode ser utilizada para parar o servidor, e habilitar ou desabilitar o *log* e o rastreamento. Estão disponíveis as seguintes opções na página de execução:

- Habilitar ou desabilitar o *log*.
- Habilitar ou desabilitar o rastreamento de todas as sessões.
- Especificar a sessão a ser rastreada (Se esta opção for escolhida, será exibida a página de rastreamento de sessão).
- Mudar o diretório de rastreamento (Se esta opção for escolhida, será exibida a

- página de diretório de rastreamento).
- Especificar os parâmetros de *thread* do *Network Server* (Se esta opção for escolhida, será exibida a página de parâmetros de *thread*).
- Parar o *Network Server*.

Página de rastreamento da sessão

Se na página de execução for escolhido especificar a sessão a ser rastreada, esta página será mostrada. Deve ser fornecido o identificador da sessão.

É dada a opção de habilitar ou desabilitar o rastreamento, ou retornar para a página principal da *servlet*. Quando se clica no botão *Trace On/Off*, é mostrada uma informação indicando o estado corrente do rastreamento.

Página do diretório de rastreamento

Se na página de execução for escolhido mudar o diretório de rastreamento, esta página será mostrada. Deve ser fornecido o diretório de rastreamento.

Pode-se tanto definir o diretório de rastreamento, quanto retornar para a página principal da *servlet*. São exibidas informações adicionais, indicando o diretório corrente de rastreamento, quando se clica no botão *Trace Directory*.

Definição de parâmetros do Network Server

Se na página de execução for escolhido definir parâmetros do *Network Server*, esta página será mostrada. Esta página é utilizada para definir os novos parâmetros. Devem ser fornecidas as seguintes informações:

- Novo número máximo de *threads*
- Nova fatia de tempo (*time slice*) da *thread*

Se o parâmetro número máximo de *threads* ou o parâmetro fatia de tempo for deixado em branco, seu valor será deixado inalterado na definição corrente.

Deve-se clicar no botão *Set Network Server Parameters*, para ver os valores atualizados dos parâmetros número máximo de *threads* e fatia de tempo.

Tópicos avançados do Network Server do Derby

Esta seção discute vários tópicos avançados para os usuários do *Network Server* do Derby.

Segurança do Network Server

Por padrão, o *Network Server* do Derby escuta apenas no *localhost*. Os clientes devem utilizar o nome de hospedeiro *localhost* para se conectarem. Por padrão, os clientes não podem acessar o *Network Server* a partir de outro hospedeiro. Para habilitar conexões a partir de outros hospedeiros, a propriedade *derby.drda.host* deve ser definida, ou o *Network Server* deve ser inicializado com a opção *-h* no comando `java org.apache.derby.drda.NetworkServerControl start`.

No exemplo a seguir, o servidor escuta apenas no *localhost* e os clientes não podem acessar o servidor a partir de outro hospedeiro.

```
java org.apache.derby.drda.NetworkServerControl start
```

No exemplo a seguir, o servidor executa na máquina hospedeira *servidor-amostra.dominio-amostra.com*, e também escuta clientes em outros hospedeiros. Os clientes devem especificar o servidor na URL ou no DataSource como *servidor-amostra.dominio-amostra.com*:

```
java org.apache.derby.drda.NetworkServerControl start
-h servidor-amostra.dominio-amostra.com
```

Para inicializar o *Network Server* de tal forma que escute em todas as interfaces, este deve ser inicializado com o endereço de IP *0.0.0.0*, conforme mostrado no exemplo a seguir:

```
java org.apache.derby.drda.NetworkServerControl start -h 0.0.0.0
```

O servidor inicializado com a opção *-h 0.0.0.0* escutará as requisições de clientes originadas tanto em *localhost* quanto em qualquer outra máquina da rede.

Porém, os comandos administrativos (por exemplo, *org.apache.derby.drda.NetworkServerControl shutdown*) somente podem ser executados no hospedeiro onde o servidor foi inicializado, mesmo que o servidor tenha sido inicializado com a opção *-h*.

Execução do Network Server sob o gerenciador de segurança

O *Network Server* deve ser executado sob o gerenciador de segurança do Java. Os próximos exemplos mostram a utilização de um arquivo de política de segurança exemplo. Deve ser feito um ajuste fino nesta política de segurança para adequá-la às próprias necessidades.

CUIDADO: Abrir o servidor para todos os clientes, sem limitar o acesso utilizando uma política de segurança semelhante a do exemplo a seguir, é um sério risco de segurança.

```
// Conjunto de permissões recomendadas para inicializar o Network Server,
// assumindo que o diretório 'd:/derby/lib' seja seguro.
// Deve ser feito um ajuste fino baseado no próprio ambiente.
grant codeBase "file:d:/derby/lib/-" {
permission java.io.FilePermission "${derby.system.home}${/}-",
    "read, write, delete";
permission java.io.FilePermission "${derby.system.home}", "read";
permission java.io.FilePermission "${user.dir}${/}-", "read, write,
delete";
permission java.util.PropertyPermission "derby.*", "read";
permission java.util.PropertyPermission "user.dir", "read";
permission java.lang.RuntimePermission "createClassLoader";
permission java.net.SocketPermission "minhamaquinacliente", "accept";
};

// Conjunto de permissões requerido para parar o Network Server,
// assumindo
// que o diretório 'd:/derby/lib' seja seguro.
// Deve ser feito um ajuste fino baseado no próprio ambiente.
grant codeBase "file:d:/derby/lib/-" {
// O que vem a seguir é requerido quando o servidor é inicializado com
// "-h localhost", ou sem a opção -h.
permission java.net.SocketPermission "localhost", "accept, connect,
resolve";
permission java.net.SocketPermission "127.0.0.1", "accept, connect,
resolve";
// O que vem a seguir é requerido apenas quando o servidor é inicializado
// com a
// opção -h <hospedeiro> (senão o acesso para "shutdown" será negado).
permission java.net.SocketPermission "<hospedeiro>:*", "accept, connect,
resolve";
};
```

- O exemplo a seguir mostra como inicializar o *Network Server* no gerenciador de segurança padrão (escutando apenas os clientes no *localhost*, que é o comportamento padrão quando a opção *-h* não é utilizada para inicializar o servidor). Este exemplo assume que o arquivo de política de segurança se

encontra em `d:/nwsvr.policy`.

```
java -Djava.security.manager -Djava.security.policy=d:/nwsvr.policy
org.apache.derby.drda.NetworkServerControl start
```

- O mesmo comportamento pode ser obtido utilizando a opção `-h` ao inicializar o servidor, conforme mostrado no exemplo a seguir:

```
java -Djava.security.manager -Djava.security.policy=d:/nwsvr.policy
org.apache.derby.drda.NetworkServerControl start -h localhost
```

- O exemplo a seguir mostra como inicializar o *Network Server* (assumindo que o servidor será inicializado na máquina hospedeira *meuservidor*) no gerenciador de segurança padrão (escutando requisições de clientes originadas a partir de outras máquinas apenas). Este exemplo assume que o arquivo de política de segurança se encontra em `d:/nwsvr.policy`.

```
java -Djava.security.manager -Djava.security.policy=d:/nwsvr.policy
org.apache.derby.drda.NetworkServerControl start -h meuservidor
```

Configuração do Network Server para tratar as conexões

O *Network Server* pode ser configurado para utilizar um número específico de *threads* para tratar as conexões. A configuração pode ser alterada na linha de comando, ou utilizando a interface *servlet*.

O número mínimo de *threads* é o número de *threads* inicializadas quando o *Network Server* é inicializado. Este valor é especificado como uma propriedade, `derby.drda.minThreads = <min>`. O número máximo de *threads* é o número máximo de *threads* que serão utilizadas pelas conexões. Se existirem mais conexões ativas que o número de *threads* disponíveis, as conexões extras deverão aguardar até uma *thread* ficar disponível. As *threads* podem se tornar disponíveis após um tempo específico, que é verificado apenas quando a *thread* termina o processamento da comunicação.

- O número máximo de *threads* pode ser alterado utilizando o seguinte comando:

```
java org.apache.derby.drda.NetworkServerControl maxthreads <max> [-h
<nome-do-hospedeiro>]
[-p <número-da-porta>]
```

Também pode ser utilizada a propriedade `derby.drda.maxThreads` para definir o valor máximo. Um valor `<máximo>` igual a zero significa que não há valor máximo, e que será gerada uma nova *thread* para a conexão se não houver uma *thread* disponível no momento. Este é o padrão. Os valores `<máximo>` e `<mínimo>` são armazenados como inteiros, portanto o valor máximo teórico é 2147483647 (o tamanho máximo de um inteiro). Mas na prática o valor máximo é determinado pela configuração da máquina.

- Para alterar o tempo que uma *thread* deve trabalhar na requisição de uma sessão e verificar se existem sessões aguardando, deve ser utilizado o seguinte comando:

```
java org.apache.derby.drda.NetworkServerControl
timeslice <milissegundos> [-h <nome-do-hospedeiro>] [-p
<número-da-porta>]
```

Também pode ser utilizada a propriedade `derby.drda.timeSlice` para definir esse valor. O valor de 0 milissegundos indica que a *thread* não vai parar de trabalhar para a sessão enquanto a sessão não terminar. O valor de -1 milissegundos indica que deve ser utilizado o valor padrão. O valor padrão é 0. O número máximo de milissegundos que pode ser especificado é 2147483647 (o tamanho máximo de um

inteiro).

Controle do registro pela utilização do arquivo de log

O *Network Server* utiliza o arquivo `derby.log` para registrar os problemas encontrados. Também registra as conexões, quando a propriedade `derby.drda.logConnections` está definida com o valor `true`. O arquivo `derby.log` é criado quando o servidor Derby é inicializado. O *Network Server* então registra a hora e a versão. Se o arquivo de *log* existir, será sobrescrito, a menos que a propriedade `derby.infolog.append` esteja definida com o valor `true`.

Quando o *Network Server* está registrando as conexões, também registra o número da conexão; a mensagem de registro é escrita tanto no arquivo `derby.log`, quanto na console do *Network Server*.

- Para habilitar o registro da conexão, pode ser utilizada a interface *servlet*, ou emitido o seguinte comando:

```
java org.apache.derby.drda.NetworkServerControl
    logconnections on [-h <nome-do-hospedeiro>] [-p
<número-da-porta>]
```

- Para desabilitar o registro da conexão, pode ser utilizada a interface *servlet*, ou emitido o seguinte comando:

```
java org.apache.derby.drda.NetworkServerControl
    logconnections off [-h <nome-do-hospedeiro>] [-p
<número-da-porta>]
```

Para obter mais informações sobre o arquivo `derby.log`, deve ser consultado o *Guia do Desenvolvedor do Derby*.

Controle do rastreamento pela utilização da facilidade de rastreamento

A facilidade de rastreamento deve ser utilizada apenas se estiver trabalhando com o suporte técnico, e for pedida a informação de rastreamento.

Para obter informações sobre o gerenciamento da facilidade de rastreamento utilizando a interface *servlet*, deve ser consultado [Gerenciamento remoto do Network Server do Derby utilizando a interface servlet](#).

Habilitação da facilidade de rastreamento

1. Habilitar o rastreamento para todas as sessões especificando a seguinte propriedade:

```
derby.drda.traceAll=true
```

Como alternativa pode ser utilizado o seguinte comando, enquanto o *Network Server* estiver executando, para habilitar a facilidade de rastreamento:

```
java org.apache.derby.drda.NetworkServerControl
    trace on [-s <número-da-conexão>] [-h <nome-do-hospedeiro>] [-p
<número-da-porta>]
```

Se for especificado o `<número-da-conexão>`, o rastreamento será habilitado apenas para esta conexão.

2. Definir o local dos arquivos de rastreamento especificando a seguinte propriedade:

```
derby.drda.traceDirectory=<diretório-dos-arquivos-de-rastreamento>
```

É necessário especificar apenas o diretório onde os arquivos de rastreamento vão residir. Os nomes dos arquivos de rastreamento são determinados pelo sistema. Se não for definido o diretório de rastreamento, os arquivos de rastreamento serão colocados em *derby.system.home*.

3. Enquanto o *Network Server* estiver executando, pode ser utilizado o seguinte comando para definir o diretório de rastreamento:

```
java org.apache.derby.drda.NetworkServerControl traceDirectory
<diretório-dos-arquivos-de-rastreamento> [-h <nome-do-hospedeiro>]
[-p <número-da-porta>]
```

Desabilitação da facilidade de rastreamento

Deve ser utilizado o seguinte comando para desabilitar o rastreamento:

```
java org.apache.derby.drda.NetworkServerControl trace off [-s
<número-da-conexão>]
[-h <nome-do-hospedeiro>] [-p <número-da-porta>]
```

Os arquivos de rastreamento recebem o nome *ServerX.trace*, onde X é o número da conexão.

Programas exemplo para o Network Server do Derby

Esta seção descreve vários programas exemplo para o *Network Server* do Derby, para os usuários do *Network Server*.

O programa exemplo NsSample

O programa de demonstração *NsSample* é um aplicativo JDBC simples que interage com o *Network Server*.

O programa *NsSample* realiza as seguintes tarefas:

- Inicializa o *Network Server*.
- Verifica se o *Network Server* está executando.
- Carrega o *driver* cliente da rede.
- Cria o banco de dados *NsSampledb*, se este não existir.
- Verifica se o esquema já existe, e se não existir cria o esquema que inclui a tabela *SAMPLETBL* e seus índices correspondentes.
- Conecta ao banco de dados.
- Carrega o esquema inserindo dados.
- Inicializa as *threads* cliente para realizar as operações de banco de dados relacionadas.
- Faz com que cada cliente realize operações da DML (seleção, inserção, exclusão e atualização) utilizando chamadas JDBC. Por exemplo, uma *thread* cliente estabelece uma conexão incorporada para realizar operações de banco de dados, enquanto outra *thread* cliente estabelece uma conexão cliente com o *Network Server* para realizar operações de banco de dados.
- Aguarda as *threads* cliente terminarem as tarefas.
- Encerra o *Network Server* e termina a demonstração.

Devem ser instalados os seguintes arquivos no diretório

%DERBY_HOME%\demo\nservdemo\ antes de executar o programa exemplo:

- NsSample.java

Este é o ponto de entrada do programa exemplo. O programa inicializa duas *threads* cliente. A primeira *thread* cliente estabelece uma conexão incorporada para realizar as operações de banco de dados, e a segunda *thread* cliente estabelece uma conexão cliente com o *Network Server* para realizar operações de banco de dados.

As seguintes constantes podem ser alteradas para modificar o programa de amostra:

NUM_ROWS

O número de linhas que devem ser carregadas inicialmente no esquema.

ITERATIONS

O número de interações nas quais cada *thread* cliente realiza um trabalho relacionado de banco de dados.

NUM_CLIENT_THREADS

O número de clientes com que se deseja executar o programa.

NETWORKSERVER_PORT

A porta onde o *Network Server* está executando.

- `NsSampleClientThread.java`

Este arquivo contém duas classes Java:

- A classe *NsSampleClientThread* estende *Thread* e cria uma instância de *NsSampleWork*.
- A classe *NsSampleWork* contém tudo que é requerido para realizar as operações da DML utilizando chamadas JDBC. O método *doWork* da classe *NsSampleWork* representa todo o trabalho realizado como parte deste programa exemplo.

- `NetworkServerUtil.java`

Este arquivo contém métodos auxiliares para inicializar o *Network Server*, e para parar o servidor.

Os arquivos de classe compilados para o programa *NsSample* são:

- `NsSample.class`
- `NsSampleClientThread.class`
- `NsSampleWork.class`
- `NetworkServerUtil.class`

Execução do programa exemplo NsSample

Para executar o programa *NsSample*:

1. Abra a linha de comando e torne o diretório `%DERBY_HOME%\demo\` o diretório corrente, onde `%DERBY_HOME%` é o diretório onde o Derby foi instalado.
2. Defina a variável de ambiente `CLASSPATH` incluindo o diretório corrente ("`.`"), e incluindo também os seguintes arquivos *jar* para que se possa utilizar o *Network Server* e o *driver* cliente da rede:

derbynet.jar

O arquivo *jar* do *Network Server*. Deverá estar presente na variável de ambiente `CLASSPATH`, para que se possa utilizar qualquer uma das funções do *Network Server*.

derbyclient.jar

Este arquivo *jar* deverá estar presente na variável de ambiente `CLASSPATH`, para que se possa utilizar o *driver* cliente da rede.

derby.jar

O arquivo *jar* do motor de banco de dados Derby.

derbytools.jar

O arquivo *jar* das ferramentas do Derby.

3. A definição da variável de ambiente `CLASSPATH` pode ser testada executando o seguinte comando Java:

```
java org.apache.derby.tools.sysinfo
```

Este comando mostra os arquivos *jar* do Derby presentes no caminho de classes, assim como suas respectivas versões.

4. Após o ambiente ter sido definido corretamente, o programa *NsSample* será executado a partir deste diretório:

```
java nserverdemo.NsSample
```

Se a execução do programa for bem-sucedida, será recebida uma saída semelhante à mostrada a seguir:

```
Using JDBC driver: org.apache.derby.jdbc.ClientDriver
Derby Network Server created
[NsSample] Unable to obtain a connection to network server, trying
again after 3000 ms.
O servidor está pronto para aceitar conexões na porta 1621.
Número da conexão: 1.
[NsSample] Derby Network Server started.
[NsSample] Sample Derby Network Server program demo starting.
Please wait .....
Número da conexão: 2.
[NsSampleWork] Begin creating table - SAMPLETBL and necessary
indexes.
[NsSampleClientThread] Thread id - 1; started.
[NsSampleWork] Thread id - 1; requests database connection, dbUrl
=jdbc:derby:NSSampled;
[NsSampleClientThread] Thread id - 2; started.
[NsSampleWork] Thread id - 2; requests database connection, dbUrl
=jdbc:derby://localhost:1621/NSSampled;create=true;
Número da conexão: 3.
[NsSampleWork] Thread id - 2; deleted 1 row with t_key = 9631
[NsSampleWork] Thread id - 2; deleted 1 row with t_key = 9595
[NsSampleWork] Thread id - 2 selected 1 row [754,Derby29
,4.0290892E7,9389]
[NsSampleWork] Thread id - 2; updated 1 row with t_key = 9389
[NsSampleWork] Thread id - 2; inserted 1 row.
[NsSampleWork] Thread id - 2; deleted 1 row with t_key = 9476
[NsSampleWork] Thread id - 2 selected 1 row [51,Derby29
,4.0290892E7,9389]
[NsSampleWork] Thread id - 2; deleted 1 row with t_key = 9389
[NsSampleWork] Thread id - 2 selected 1 row [486,Derby25
,6.138386E23,9383]
[NsSampleWork] Thread id - 2; inserted 1 row.
[NsSampleWork] Thread id - 2; closed connection to the database.
[NsSampleClientThread] Thread id - 2; finished all tasks.
[NsSampleWork] Thread id - 1; deleted 1 row with t_key = 9383
[NsSampleWork] Thread id - 1 selected 1 row [948,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1 selected 1 row [948,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1 selected 1 row [948,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1; updated 1 row with t_key = 9100
[NsSampleWork] Thread id - 1 selected 1 row [54,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1 selected 1 row [54,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1 selected 1 row [54,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1; updated 1 row with t_key = 9100
[NsSampleWork] Thread id - 1 selected 1 row [55,Derby33
,7609340.0,9100]
[NsSampleWork] Thread id - 1; closed connection to the database.
[NsSampleClientThread] Thread id - 1; finished all tasks.
[NsSample] Shutting down network server.
Número da conexão: 4.
Encerramento bem-sucedido.
[NsSample] End of Network server demo.
```

A execução do programa *NsSample* também cria o seguinte diretório e arquivo:

NSSampled

Este diretório contém o banco de dados *NSSampled*.

derby.log

Este arquivo de *log* contém as mensagens de progresso e erro do Derby.

Programas exemplo do Network Server para conexões incorporada e cliente

Estes programas exemplo do *Network Server* do Derby mostram como obter uma conexão incorporada e várias conexões cliente para o mesmo banco de dados utilizando o *Network Server*. Estes programas mostram, também, como utilizar tanto *DriverManager* quanto *DataSource* para obter conexões cliente.

Para o banco de dados permanecer consistente, somente pode ser acessado por uma JVM de cada vez. O *driver* incorporado é carregado quando o *Network Server* é inicializado. A JVM que inicializa o *Network Server* pode obter uma conexão incorporada para o mesmo banco de dados que o *Network Server* está acessando para servir os clientes de outras JVM. Esta solução fornece os benefícios de desempenho do *driver* incorporado, e também permite que conexões cliente de outras JVMs se conectem ao mesmo banco de dados.

Visão geral do programa SimpleNetworkServerSample

O programa SimpleNetworkServerSample inicializa o *Network Server* do Derby, assim como o *driver* incorporado, e aguarda os clientes se conectarem. O programa realiza as seguintes tarefas:

- Inicializa o *Network Server* do Derby utilizando uma propriedade e também carrega o *driver* incorporado.
- Determina se o *Network Server* está executando.
- Cria o banco de dados NSSimpleDB, se este não existir.
- Obtém uma conexão incorporada com o banco de dados.
- Testa a conexão com o banco de dados utilizando uma consulta de exemplo.
- Permite conexões cliente se conectarem ao servidor, até que se decida parar o servidor e sair do programa.
- Fecha a conexão.
- Pára o *Network Server* antes de sair do programa.

Para executar este programa exemplo, devem ser instalados os seguintes arquivos no diretório %DERBY_HOME%\demo\nserverdemo\:

- O arquivo fonte: SimpleNetworkServerSample.java
- O arquivo de classe compilado: SimpleNetworkServerSample.class

Execução do programa SimpleNetworkServerSample

Para executar este programa de exemplo do *Network Server* do Derby:

1. Abra a linha de comando e torne o diretório %DERBY_HOME%\demo\nserverdemo o diretório corrente, onde %DERBY_HOME% é o diretório onde o Derby foi instalado.
2. Defina a variável de ambiente CLASSPATH contendo o diretório corrente ("."), e incluindo também os seguintes arquivos *jar*:

derbynet.jar

O arquivo *jar* do *Network Server*. Deverá estar presente na variável de ambiente CLASSPATH, porque o *Network Server* será inicializado por este programa.

derby.jar

O arquivo *jar* do motor de banco de dados.

derbytools.jar

O arquivo *jar* das ferramentas do Derby.

3. A definição da variável de ambiente CLASSPATH pode ser testada executando o seguinte comando Java:

```
java org.apache.derby.tools.sysinfo
```

Este comando mostra os arquivos *jar* do Derby presentes no caminho de classes,

assim como suas respectivas versões.

4. Após o ambiente ter sido definido corretamente, o programa *SimpleNetworkServerSample* será executado a partir deste diretório:

```
java SimpleNetworkServerSample
```

Se a execução do programa for bem-sucedida, será vista uma saída semelhante à mostrada abaixo:

```
Starting Network Server
Testing if Network Server is up and running!
Derby Network Server now running
Got an embedded connection.
Testing embedded connection by executing a sample query
number of rows in sys.systables = 16
While my app is busy with embedded work, ij might connect like this:

    $ java -Dij.user=me -Dij.password=pw -Dij.protocol=
      jdbc:derby:\\localhost:1527\
org.apache.derby.tools.ij
ij> connect 'NSSimpleDB';

Clients can continue to connect:
Press [Enter] to stop Server
```

A execução do programa *SimpleNetworkServerSample* também cria o seguinte diretório e arquivo:

NSSimpleDB

Este diretório contém o banco de dados *NSSimpleDB*.

derby.log

Este arquivo de *log* contém as mensagens de progresso e erro do Derby.

Conexão de um cliente ao Network Server com o programa SimpleNetworkClientSample

O programa *SimpleNetworkClientSample* é um programa cliente que interage com o *Network Server* do Derby a partir de outra JVM. Este programa realiza as seguintes tarefas:

- Carrega o *driver* cliente da rede.
- Obtém uma conexão cliente utilizando *DriverManager*.
- Obtém uma conexão cliente utilizando *DataSource*.
- Testa as conexões com o banco de dados executando uma consulta de exemplo.
- Fecha as conexões e termina o programa.

Para executar este programa exemplo, devem ser instalados os seguintes arquivos no diretório %DERBY_HOME%\demo\nserverdemo\:

- O arquivo fonte: *SimpleNetworkClientSample.java*
- O arquivo de classe compilado: *SimpleNetworkClientSample.class*

Execução do programa SimpleNetworkClientSample

Para conectar ao *Network Server* inicializado pelo programa *SimpleNetworkServerSample*:

1. Abra a linha de comando e torne o diretório %DERBY_HOME%\demo\nserverdemo o diretório corrente, onde %DERBY_HOME% é o diretório onde o Derby foi instalado.
2. Defina a variável de ambiente CLASSPATH contendo o seguinte:
 - O diretório corrente (".")
 - O arquivo derbyclient.jar
3. Após o ambiente ter sido definido corretamente, o programa *SimpleNetworkClientSample* será executado a partir deste diretório:

```
java SimpleNetworkClientSample
```

Se a execução do programa for bem-sucedida, será vista uma saída semelhante à mostrada abaixo:

```
Starting Sample client program
Got a client connection via the DriverManager.
connection from datasource;
Got a client connection via a DataSource.
Testing the connection obtained via DriverManager by executing a
sample query
number of rows in sys.systables = 16
Testing the connection obtained via a DataSource by executing a
sample query
number of rows in sys.systables = 16
Goodbye!
```

Parte 2: Guia de administração do Derby

Esta seção do guia está dividida em várias tarefas administrativas.

Verificação da consistência do banco de dados

Caso ocorra uma falha do equipamento ou do sistema operacional, pode ser utilizada a função `SYSCS_UTIL.SYSCS_CHECK_TABLE` para verificar se o banco de dados ainda permanece consistente.

A verificação da consistência deve ser realizada apenas quando existem indicações que este tipo de verificação é necessária, porque a verificação da consistência pode durar um longo tempo em um banco de dados grande.

A função `SYSCS_CHECK_TABLE`

A função `SYSCS_UTIL.SYSCS_CHECK_TABLE()` verifica a consistência de uma tabela do Derby. Em particular, a função `SYSCS_UTIL.SYSCS_CHECK_TABLE` verifica as seguintes condições:

- Se a tabela base está consistente internamente;
- Se a tabela base e todos os seus índices associados contêm o mesmo número de linhas;
- Se o valor e localização das linhas de cada índice correspondem aos da tabela base;
- Se todos os índices BTREE estão consistentes internamente.

Esta função é executada como uma declaração SQL, conforme mostrado a seguir:

```
VALUES SYSCS_UTIL.SYSCS_CHECK_TABLE(  
    nome-do-esquema, nome-da-tabela)
```

onde *nome-do-esquema* e *nome-da-tabela* são expressões que resultam em um tipo de dado cadeia de caracteres. Se o nome do esquema ou da tabela for criado como um identificador não delimitado, seus nomes deverão ser informados com todas as letras em maiúsculo. Por exemplo:

```
VALUES SYSCS_UTIL.SYSCS_CHECK_TABLE('APP', 'CIDADES')
```

A função `SYSCS_UTIL.SYSCS_CHECK_TABLE` retorna um `SMALLINT`. Se a tabela estiver consistente (ou se for executada para uma visão), a função `SYSCS_UTIL.SYSCS_CHECK_TABLE` retornará um valor diferente de zero. Caso contrário, esta função lançará uma exceção na primeira inconsistência encontrada.

Para uma tabela consistente, é mostrado o seguinte resultado:

```
1  
-----  
1  
  
1 row selected
```

Exemplos de mensagem de erro de `SYSCS_CHECK_TABLE`

Esta seção fornece exemplos de mensagens de erro que a função `SYSCS_UTIL.SYSCS_CHECK_TABLE()` pode retornar.

Se a contagem de linhas da tabela base e de um índice estiverem diferentes, será emitida a mensagem de erro X0Y55:

```
ERROR X0Y55: The number of rows in the base table does not match
the number of rows in at least 1 of the indexes on the table. Index
'T1_I' on table 'APP.T1' has 4 rows, but the base table has 5 rows.
The suggested corrective action is to recreate the index.
```

Se o índice fizer referência a uma linha que não existe na tabela base, será emitida a mensagem de erro X0X62:

```
ERROR X0X62: Inconsistency found between table 'APP.T1' and index
'T1_I'. Error when trying to retrieve row location '(1,6)' from the
table. The full index key, including the row location, is '{ 1, (1,6) }'.
The suggested corrective action is to recreate the index.
```

Se o valor da coluna chave for diferente na tabela base e no índice, será emitida a mensagem de erro X0X61:

```
ERROR X0X61: The values for column 'C10' in index 'T1_C10' and
table 'APP.T1' do not match for row location (1,7). The value in the
index is '2 2 ', while the value in the base table is 'NULL'. The full
index key, including the row location, is '{ 2 2 , (1,7) }'. The
suggested corrective action is to recreate the index.
```

Exemplos de comandos SYSCS_CHECK_TABLE

Esta seção fornece exemplos que mostram como utilizar a função SYSCS_UTIL.SYSCS_CHECK_TABLE em comandos.

Para verificar a consistência de uma única tabela, deve ser executado um comando semelhante ao do exemplo a seguir:

```
VALUES SYSCS_UTIL.SYSCS_CHECK_TABLE('APP', 'VÔOS')
```

Para verificar a consistência de todas as tabelas do esquema, parando na primeira falha, deve ser executado um comando semelhante ao do exemplo a seguir:

```
SELECT tablename, SYSCS_UTIL.SYSCS_CHECK_TABLE(
  'APP', tablename)
FROM sys.sysschemas s, sys.systables t
WHERE s.schemaname = 'APP' AND s.schemaid = t.schemaid
```

Para verificar a consistência de todo o banco de dados, parando na primeira falha, deve ser executado um comando semelhante ao do exemplo a seguir:

```
SELECT schemaname, tablename,
SYSCS_UTIL.SYSCS_CHECK_TABLE(schemaname, tablename)
FROM sys.sysschemas s, sys.systables t
WHERE s.schemaid = t.schemaid
```

Cópia de segurança e restauração do banco de dados

O Derby fornece maneira de efetuar cópia de segurança do banco de dados enquanto este está em linha. Também pode ser restaurada uma cópia de segurança completa a

partir de um local especificado.

Cópia de segurança do banco de dados

Os tópicos desta seção descrevem realizar cópia de segurança do banco de dados.

Cópias de segurança fora de linha

Para realizar uma cópia de segurança fora de linha de um banco de dados, são utilizados comandos do sistema operacional para copiar o diretório do banco de dados. O banco de dados deve ser parado antes de ser feita a cópia de segurança fora de linha.

Por exemplo, nos sistemas Windows, o seguinte comando do sistema operacional faz a cópia de segurança do banco de dados (fechado) chamado *amostra*, localizado em `d:\meusbancos`, copiando o banco de dados para o diretório `c:\minhascopias\2005-06-01`:

```
xcopy d:\meusbancos\amostra c:\minhascopias\2005-06-01\amostra /s /i
```

Se não estiver sendo utilizado o Windows, o comando *xcopy* deverá ser substituído pelo comando apropriado do sistema operacional que copia o diretório, e todo o seu conteúdo, para um novo local.

Note: Nos sistemas Windows, não deve-se tentar atualizar o banco de dados enquanto está sendo feita a cópia de segurança desta maneira. A tentativa de atualizar o banco de dados durante uma cópia de segurança fora de linha irá gerar uma `java.io.IOException`. A utilização de cópias de segurança em linha evita que isto ocorra.

Em sistemas grandes, parar o banco de dados pode não ser conveniente. Para fazer uma cópia de segurança do banco de dados sem ter que pará-lo, deve ser feita a cópia de segurança em linha.

Cópias de segurança em linha

Deve ser utilizada a cópia de segurança em linha, para fazer cópia de segurança de um banco de dados enquanto este está em execução.

As cópias de segurança em linha podem ser feitas utilizando vários tipos de procedimento de cópia de segurança, ou utilizando comandos do sistema operacional junto com os procedimentos de congelar e descongelar do sistema.

Utilização do procedimento de cópia de segurança para realizar cópia de segurança em linha:

O procedimento `SYSCS_UTIL.SYSCS_BACKUP_DATABASE` é utilizado para criar a cópia de segurança do banco de dados em um local especificado.

O procedimento `SYSCS_UTIL.SYSCS_BACKUP_DATABASE` recebe como argumento uma cadeia de caracteres representando o local onde será armazenada a cópia de segurança do banco de dados. Normalmente é fornecido o caminho completo para o diretório de cópia de segurança (os caminhos relativos são interpretados como sendo relativos ao diretório corrente, e não ao diretório `derby.system.home`).

Por exemplo, para especificar o local de cópia de segurança `c:\minhascopias\2005-06-01` para um banco de dados que se encontra aberto no momento, deve ser utilizada a seguinte declaração (são utilizadas barras inclinadas para frente como separadores do caminho nos comandos SQL):

```
CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE('c:/minhascopias/2005-06-01')
```

O procedimento `SYSCS_UTIL.SYSCS_BACKUP_DATABASE()` coloca o banco de dados em um estado no qual este pode ser copiado com segurança, depois copia o

diretório de banco de dados original por completo (incluindo os arquivos de dados, arquivos de *log* de transação em linha e os arquivos *jar*) para o diretório de cópia de segurança especificado. Os arquivos que não estão dentro do diretório de banco de dados original (por exemplo, `derby.properties`) *não* são copiados.

O exemplo a seguir mostra como fazer a cópia de segurança de um banco de dados em um diretório cujo nome reflete a data corrente:

```
public static void backUpDatabase(Connection conn)throws SQLException
{
    // Obter a data de hoje como uma cadeia de caracteres:
    java.text.SimpleDateFormat dataHoje =
        new java.text.SimpleDateFormat("yyyy-MM-dd");
    String diretorioCopia = "c:/minhascopias/" +
        dataHoje.format((java.util.Calendar.getInstance()).getTime());

    CallableStatement cs = conn.prepareCall("CALL
    SYSCS_UTIL.SYSCS_BACKUP_DATABASE(?)");
    cs.setString(1, diretorioCopia);
    cs.execute();
    cs.close();
    System.out.println("cópia de segurança colocada no diretório
    "+diretorioCopia);
}
```

Para um banco de dados cuja cópia de segurança foi realizada em 2005-06-01, os comandos anteriores copiarão o banco de dados corrente para o diretório com o mesmo nome em `c:\minhascopias\2005-06-01`.

As transações não efetivadas não aparecem na cópia de segurança do banco de dados.

Note: Não devem ser feitas cópias de segurança de bancos de dados diferentes, mas com mesmo nome, no mesmo diretório de cópia de segurança. Se já existir um banco de dados com o mesmo nome no diretório de cópia de segurança, é assumido como sendo uma versão antiga e sobrescrito.

O procedimento `SYSCS_UTIL.SYSCS_BACKUP_DATABASE` emitirá uma mensagem de erro se existirem operações não registradas (*unlogged*) na mesma transação do procedimento de cópia de segurança.

Caso exista no sistema, quando a cópia de segurança iniciar, operações não registradas em andamento em outras transações, este procedimento ficará bloqueado até que estas transações completem, antes de realizar a cópia de segurança.

O Derby converte, automaticamente, as operações não registradas para o modo registrado, quando estas são iniciadas quando a cópia de segurança está em andamento (exceto as operações que fazem manutenção de arquivos *jar* de aplicativos no banco de dados). Os procedimentos que instalam, substituem e removem arquivos *jar* no banco de dados são bloqueados quando a cópia de segurança está em andamento.

Se não for desejado que a cópia de segurança fique bloqueada até que as operações não registradas em outras transações completem, deve ser utilizado o procedimento `SYSCS_UTIL.SYSCS_BACKUP_DATABASE_NOWAIT`. Esse procedimento emite um erro logo no início da cópia de segurança caso existam transações em andamento com operações não registradas, em vez de aguardar estas transações completarem.

Utilização de comandos do sistema operacional com os procedimentos do sistema de congelar e descongelar para realizar cópias de segurança em linha:

Normalmente, estes procedimentos são utilizados para acelerar a operação de cópia envolvida na cópia de segurança em linha. Neste cenário, o Derby não realiza a operação de cópia. É utilizado o procedimento `SYSCS_UTIL.SYSCS_FREEZE_DATABASE` para bloquear o banco de dados, e depois realizada a cópia do diretório do banco de dados utilizando comandos do sistema

operacional.

Por exemplo, como o comando *tar* do UNIX utiliza rotinas de cópia de arquivo do sistema operacional, e o procedimento SYSCS_UTIL.SYSCS_BACKUP_DATABASE utiliza chamada de E/S do Java, com sincronização interna adicional para permitir que ocorram atualizações durante a cópia de segurança, o comando *tar* pode gerar cópias de segurança mais rapidamente que o procedimento SYSCS_UTIL.SYSCS_BACKUP_DATABASE.

Para utilizar comandos do sistema operacional para realizar cópias de segurança em linha, é chamado o procedimento do sistema SYSCS_UTIL.SYSCS_FREEZE_DATABASE. O procedimento do sistema SYSCS_UTIL.SYSCS_FREEZE_DATABASE coloca o banco de dados em um estado onde este pode ser copiado com segurança. Após o banco de dados ter sido copiado, é utilizado o procedimento do sistema SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE para continuar trabalhando com o banco de dados. Somente após SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE ter sido utilizado, as transações poderão escrever novamente no banco de dados. As operações de leitura podem prosseguir enquanto o banco de dados estiver "congelado."

Note: Para garantir uma cópia de segurança do banco de dados consistente, o Derby deverá bloquear os aplicativos que tentarem escrever no banco de dados congelado, até que a cópia de segurança esteja completa e o procedimento do sistema SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE seja chamado.

O exemplo a seguir demonstra como os procedimentos de congelar e descongelar são utilizados para envolver o comando de cópia do sistema operacional:

```
public static void backUpDatabaseWithFreeze(Connection conn)
    throws SQLException
{
    Statement s = conn.createStatement();
    s.executeUpdate(
        "CALL SYSCS_UTIL.SYSCS_FREEZE_DATABASE()");
    // Copiar o diretório do banco de dados durante este intervalo
    s.executeUpdate(
        "CALL SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE()");
    s.close();
}
```

Quando o log está em um local diferente do padrão

Note: Para se informar sobre o local padrão do *log* do banco de dados deve ser lido [Arquivo de log em uma unidade separada](#).

Se o *log* do banco de dados for colocado em um local diferente do padrão antes de fazer a cópia de segurança do banco de dados, deve-se ter em mente os seguintes requisitos:

- Se estiver sendo utilizado um comando do sistema operacional para realizar a cópia de segurança do banco de dados, o arquivo de *log* deverá ser copiado também, conforme mostrado no exemplo a seguir:

```
xcopy d:\meusbancos\amostra c:\minhascopias\2005-06-01\amostra /s /i
xcopy h:\janet\tourslog\log c:\minhascopias\2005-06-01\amostra\log
/s /i
```

Se não estiver sendo utilizado o Windows, o comando de cópia deverá ser substituído pelo comando do sistema operacional apropriado para copiar o diretório, assim como todo o seu conteúdo, para um novo local.

- A entrada *logDevice* de *service.properties* deverá ser editada no banco de dados da cópia de segurança, para que aponte para o local correto do *log*. No exemplo anterior, o *log* foi movido para o local padrão do *log*, portanto a entrada *logDevice* poderá ser removida, ou deixado a entrada *logDevice* como está, e aguardar até que o banco de dados seja restaurado para editar a entrada.

Para obter informações sobre como colocar o *log* em um local diferente do padrão, deve

ser consultado [Arquivo de log em uma unidade separada](#).

Cópia de segurança de um banco de dados criptografado

Quando se realiza a cópia de segurança de um banco de dados criptografado, tanto os arquivos da cópia de segurança quanto o arquivo de *log* permanecem criptografados.

Para restaurar um banco de dados criptografado, é necessário conhecer a senha de inicialização.

Restauração do banco de dados a partir da cópia de segurança

Para restaurar um banco de dados utilizando uma cópia de segurança completa presente em um determinado local, deve ser especificado o atributo `restoreFrom=caminho` na URL de conexão em tempo de inicialização.

Se existir um banco de dados com o mesmo nome no local *derby.system.home*, o sistema irá remover este banco de dados, copiá-lo a partir do local da cópia de segurança, e reinicializá-lo.

Os arquivos de *log* são copiados para o mesmo local onde se encontravam quando a cópia de segurança foi realizada. Pode ser utilizado o atributo `logDevice` junto com o atributo `restoreFrom=caminho` para armazenar os *logs* em um local diferente.

Por exemplo, para restaurar o banco de dados amostra utilizando a cópia de segurança armazenada em `c:\minhascopias\amostra`, a URL de conexão deve ser:

```
jdbc:derby:sample;restoreFrom=c:\minhascopias\amostra
```

Criação de um banco de dados a partir de uma cópia de segurança

Para criar um banco de dados utilizando uma cópia de segurança completa presente em um determinado local, deve ser especificado o atributo `createFrom=caminho` na URL de conexão em tempo de inicialização.

Caso exista um banco de dados com o mesmo nome no local *derby.system.home*, ocorrerá um erro e o banco de dados existente será deixado intacto. Caso não exista um banco de dados com o mesmo nome no local *derby.system.home* corrente, o sistema irá copiar todo o banco de dados do local da cópia de segurança para o local *derby.system.home*, e inicializá-lo.

Os arquivos de *log* também são copiados para o local padrão. Pode ser utilizado o atributo `logDevice` junto com o atributo `createFrom=caminho` para armazenar os *logs* em um local diferente. Com o atributo `createFrom=caminho` não há necessidade de copiar os arquivos de *log* individualmente para o diretório de *log*.

Por exemplo, para criar o banco de dados amostra a partir da cópia de segurança armazenada em `c:\minhascopias\amostra`, a URL deve ser:

```
jdbc:derby:amostra;createFrom=c:\minhascopias\amostra
```

Recuperação com rolagem para frente

O Derby suporta a recuperação com rolagem para frente (*roll-forward recovery*) para restaurar um banco danificado para o estado mais recente anterior à ocorrência da falha.

O Derby restaura o banco de dados a partir da cópia de segurança completa, e refaz todas as transações posteriores à cópia de segurança. São necessários todos os

arquivos de *log* posteriores à cópia de segurança, para refazer as transações posteriores à cópia de segurança. Por padrão, o banco de dados mantém apenas os *logs* requeridos para a recuperação de queda. Para a recuperação com rolagem para frente ser bem-sucedida, todos os arquivos de *log* posteriores à cópia de segurança devem ser guardados. Os arquivos de *log* podem ser guardados utilizando chamadas de função de cópia de segurança que habilitam guardar o *log*.

Na recuperação com rolagem para frente o modo que guarda o *log* garante que todos os arquivos de *log* antigos ficam disponíveis. Os arquivos de *log* ficam disponíveis somente a partir do momento em que o modo que guarda o *log* é habilitado.

O Derby utiliza as seguinte informações para restaurar o banco de dados:

- A cópia de segurança do banco de dados
- O conjunto de *logs* guardados
- O *log* em linha ativo no momento

A recuperação com rolagem para frente não pode ser utilizada para restaurar tabelas individuais. A recuperação com rolagem para frente recupera todo o banco de dados.

Para restaurar um banco de dados utilizando a recuperação com rolagem para frente deverá existir uma cópia de segurança do banco de dados, todos os *logs* guardados desde que a cópia de segurança foi criada, e os arquivos de *log* ativos. Todos os arquivos de *log* deverão estar no diretório de *log* do banco de dados.

Existem dois tipos de arquivo de *log* no Derby: os *logs* ativos e os *logs* em linha guardados.

Logs ativos

Os *logs* ativos são utilizados durante a recuperação de queda para evitar que uma falha que deixe o banco de dados em um estado inconsistente. A recuperação com rolagem para frente também pode utilizar os *logs* ativos para recuperar até o final dos arquivos de *log*. Os *logs* ativos estão localizados no diretório de caminho de *log* do banco de dados.

Logs em linha guardados

Arquivos de *log* guardados para uso pela recuperação com rolagem para frente, após não serem mais necessários para recuperação de queda. Os *logs* em linha guardados também são mantidos no diretório de caminho de *log* do banco de dados.

Habilitação do modo que guarda o *log*

Os *logs* em linha guardados estarão disponíveis somente se o banco de dados for habilitado para o modo de guarda o *log*. Pode ser utilizado o seguinte procedimento do sistema para habilitar o banco de dados no modo que guarda o *log*:

```
SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE  
(IN BACKUPDIR VARCHAR(32672), IN SMALLINT DELETE_ARCHIVED_LOG_FILES)
```

Os parâmetros de entrada para a chamada do exemplo anterior especificam o local onde a cópia de segurança deverá ser armazenada, e especifica se o banco de dados deverá manter os *logs* em linha guardados para a cópia de segurança. Os arquivos de *log* em linha guardados existentes, criados antes desta cópia de segurança, serão eliminados se o valor do parâmetro de entrada para o parâmetro *deleteOnlineArchivedLogFiles* for diferente de zero. Os arquivos de *log* são eliminados somente após a cópia de segurança ter sido bem-sucedida.

Note: Tenha certeza de armazenar a cópia de segurança do banco de dados em um local seguro ao escolher a opção de remoção do arquivo de *log*.

O procedimento

SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE emite uma mensagem de erro quando existem operações não registradas na mesma transação do procedimento de cópia de segurança.

Caso exista no sistema, quando a cópia de segurança iniciar, operações não registradas em andamento em outras transações, este procedimento ficará bloqueado até que estas transações completem, antes de realizar a cópia de segurança. O Derby converte, automaticamente, as operações não registradas para o modo registrado, quando estas são iniciadas quando a cópia de segurança está em andamento (exceto as operações que fazem manutenção de arquivos *jar* de aplicativo no banco de dados). Os procedimentos que instalam, substituem e removem arquivos *jar* no banco de dados são bloqueadas quando a cópia de segurança está em andamento.

Se não for desejado que a cópia de segurança fique bloqueada até que as operações não registradas em outras transações completem, deve ser utilizado o procedimento SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE_NOWAIT. Esse procedimento emite um erro logo no início da cópia de segurança caso existam transações em andamento com operações não registradas, em vez de aguardar estas transações completarem.

Desabilitação do modo que guarda o *log*

Após o modo que guarda o *log* ter sido habilitado, o banco de dados ficará para sempre com o modo que guarda o *log* habilitado, mesmo que posteriormente seja recarregado ou seja feita uma cópia de segurança. A única maneira de desabilitar o modo que guarda o *log* é executar o seguinte procedimento:

```
SYSCS_UTIL.SYSCS_DISABLE_LOG_ARCHIVE_MODE(IN SMALLINT  
DELETE_ARCHIVED_LOG_FILES)
```

Este procedimento do sistema desabilita o modo que guarda o *log*, e remove todos os arquivos de *log* guardados existentes, se o parâmetro de entrada *DELETE_ARCHIVED_LOG_FILES* for diferente de zero.

Realização da recuperação com rolagem para frente:

Utilizando a cópia de segurança completa, os *logs* guardados, e os *logs* ativos, o banco de dados pode ser restaurado até seu estado mais recente realizando a recuperação com rolagem para frente. A recuperação com rolagem para frente é realizada especificando o atributo da URL de conexão *rollForwardRecoveryFrom=<caminho-da-cópia-de-segurança>* em tempo de inicialização. Com isto o banco de dados retorna ao seu estado mais recente utilizando a cópia de segurança completa, os *logs* guardados e os *logs* ativos. Todos os arquivos de *log* deverão estar no diretório do caminho de *log* do banco de dados.

Cópia de segurança do banco de dados:

No exemplo a seguir é realizada a cópia de segurança do banco de dados chamado *wombat* no diretório *d:/backup* com o modo que guarda arquivo de *log* habilitado:

```
connect 'jdbc:derby:wombat;create=true';  
create table t1(a int not null primary key);  
-----DML/DDI Operations  
CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE  
( 'd:/backup', 0);  
insert into t1 values(19);  
create table t2(a int);  
-----Operações de DML/DDI  
-----Queda do banco de dados (Mídia corrompida nos discos de  
dado)
```

Restauração do banco de dados utilizando a recuperação com rolagem para frente:

No exemplo a seguir, o banco de dados é restaurado utilizando a recuperação com

rolagem para frente após uma falha da mídia:

```
connect 'jdbc:derby:wombat;rollForwardRecoveryFrom=d:/backup/wombat';  
select * from t1;  
-----Operações de DML/DDDL
```

Pode ser especificado o seguinte atributo na URL de conexão em tempo de carga do JDBC:

rollForwardRecoveryFrom=caminho

Para obter mais informações deve ser consultada a seção *rollForwardRecoveryFrom=caminho* no *Manual de Referência do Derby*.

Após o banco de dados ser restaurado a partir da cópia de segurança completa, as transações dos *logs* em linha arquivados e dos *logs* ativos são refeitas.

Arquivo de log em uma unidade separada

Pode ser melhorado o desempenho de bancos de dados grandes com muitas atualizações colocando o *log* do banco de dados em uma unidade separada, o que reduz a contenção de E/S.

Por padrão, o registro da transação fica no subdiretório *log* do diretório do banco de dados. Deve ser utilizado um dos seguintes métodos para armazenar este subdiretório *log* em outro local:

- Especificar um local diferente do padrão utilizando o atributo *logDevice* na URL de conexão com o banco de dados, ao criar o banco de dados.
- Se o banco de dados já tiver sido criado, mover o diretório de *log* manualmente e atualizar o arquivo *service.properties*.

Utilização do atributo logDevice

Para especificar um local diferente do padrão para o diretório de *log*, deve ser definido o atributo *logDevice* na URL de conexão com o banco de dados ao criar o banco de dados.

Este atributo só faz sentido ao se criar o banco de dados. O atributo *logDevice* pode ser especificado como um caminho absoluto, ou como um caminho relativo ao diretório onde a JVM é executada.

Definir *logDevice* na URL de conexão com o banco de dados adiciona uma entrada no arquivo *service.properties*. Se o diretório de *log* for movido manualmente, será necessário alterar esta entrada no arquivo *service.properties*. Se o diretório de *log* for movido de volta para o local padrão, a entrada *logDevice* deverá ser removida do arquivo *service.properties*.

Para verificar o local do *log* de um banco de dados existente, pode ser consultado o atributo *logDevice* como uma propriedade do banco de dados utilizando a seguinte declaração:

```
VALUES SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY('logDevice')
```

Exemplo de criação do log em um local diferente do padrão

A URL de conexão com o banco de dados a seguir cria um banco de dados no diretório

d:/meusbancos, mas coloca o diretório de *log* em h:/janets/logTurismo:

```
jdbc:derby:d:/meusbancos/bancoTurismo;  
create=true;logDevice=h:/janets/logTurismo
```

Exemplo de mover o log manualmente

Caso se deseje mover o *log* para g:/discogrande/logTurismo, o mesmo deverá ser movido utilizando comandos do sistema operacional:

```
move h:\janets\logTurismo\log\*. * g:\discogrande\logTurismo\log
```

Em seguida, a entrada *logDevice* no arquivo *service.properties* deverá ser alterada para ficar com a seguinte forma:

```
logDevice=g:/discogrande/logTurismo
```

Note: Pode ser usado tanto uma barra inclinada para frente quanto duas contrabarras como separador do caminho.

Se posteriormente for desejado mover o *log* de volta para o local padrão (neste caso d:\meusbancos\bancoTurismo\log), o diretório de *log* deverá ser movido manualmente da seguinte maneira:

```
move g:\discogrande\logTurismo\log\*. * d:\meusbancos\bancoTurismo\log
```

Em seguida, a entrada *logDevice* no arquivo *service.properties* deverá ser removida.

Note: Este exemplo utiliza comandos específicos do sistema operacional Windows. Deverão ser empregados os comandos apropriados do sistema operacional utilizado, para copiar o diretório e todo o seu conteúdo para um novo local.

Questões relativas ao log em um local diferente do padrão

Quando o diretório de *log* não está no local padrão, a cópia de segurança e a restauração do banco de dados poderão requerer passos adicionais. Para obter detalhes, deve ser consultado [Cópia de segurança e restauração do banco de dados](#).

Obtenção de informações sobre bloqueios

O Derby fornece uma ferramenta para monitorar e mostrar informações sobre bloqueios. Esta ferramenta ajuda criar aplicativos que minimizam os impasses (*deadlocks*). Também pode ajudar a localizar a causa do impasse quando este ocorre.

Para diagnosticar os problemas de bloqueio, deve ser monitorado constantemente o tráfego de bloqueio registrando todos os impasses utilizando a propriedade *derby.locks.monitor*.

Monitoramento dos impasses

A propriedade *derby.stream.error.logSeverityLevel* determina o nível de erro informado.

Por padrão, a propriedade *derby.stream.error.logSeverityLevel* é definida com o valor

40000. Se a propriedade *derby.stream.error.logSeverityLevel* for definida para mostrar os erros no nível de transação (ou seja, se for definida com um valor inferior a 40000), os erros de impasse serão registrados no arquivo *derby.log*. Se for definida com um valor igual a 40000, ou maior, os erros de impasse não serão registrados no arquivo *derby.log*.

A propriedade *derby.locks.monitor* garante que os erros de impasse serão registrados, independentemente do valor de *derby.stream.error.logSeverityLevel*. Quando *derby.locks.monitor* está definida como verdade, todos os bloqueios envolvidos nos impasses são escritos no arquivo *derby.log*, junto com um número único que identifica o bloqueio.

Para ver o acompanhamento da pilha de *thread* quando o bloqueio é requisitado, a propriedade *derby.locks.deadlockTrace* deverá ser definida como verdade. Esta propriedade é ignorada quando *derby.locks.monitor* está definida como falso.

Note: A propriedade *derby.locks.deadlockTrace* deve ser utilizada com cuidado. Definir esta propriedade pode alterar o tempo do aplicativo, afetar severamente o desempenho, e produzir um arquivo *derby.log* muito grande.

Para obter informações sobre como definir estas propriedades, e informações sobre as propriedades específicas mencionadas neste tópico, deve ser consultado *Ajuste do Derby*.

Abaixo está um exemplo de uma mensagem de erro quando o Derby interrompe uma transação por causa de um impasse:

```
--SQLException Caught--
SQLState: 40001 =
Error Code: 30000
Message: A lock could not be obtained due to a deadlock,
cycle of locks and waiters is: Lock : ROW, DEPARTMENT, (1,14)
Waiting XID : {752, X} , APP, update department set location='Boise'
           where deptno='E21'
Granted XID : {758, X} Lock : ROW, EMPLOYEE, (2,8)
Waiting XID : {758, U} , APP, update employee set bonus=150 where
salary=23840
Granted XID : {752, X} The selected victim is XID : 752
```

Note: Podem ser utilizadas as propriedades *derby.locks.waitTimeout* e *derby.locks.deadlockTimeout* para configurar quanto tempo o Derby vai aguardar o bloqueio ser liberado, ou quando iniciar a verificação do impasse. Para obter mais informações sobre estas propriedades, deve ser consultada a seção que discute o controle do comportamento de aplicativos do Derby no *Guia do Desenvolvedor do Derby*.

Recuperação do espaço não utilizado

As tabelas e índices do Derby (algumas vezes chamados de *conglomerados*) podem conter espaço não utilizado, depois de grandes quantidades de dados terem sido excluídas ou atualizadas.

Isto acontece porque, por padrão, o Derby não devolve o espaço não utilizado para o sistema operacional. Após a página ter sido alocada para uma tabela ou um índice, o Derby não devolve automaticamente a página para o sistema operacional até que a tabela ou índice seja removido, mesmo que o espaço não seja mais necessário. Entretanto, o Derby fornece um mecanismo para recuperar o espaço não utilizado nas tabelas e nos seus índices associados.

Se for determinado que a tabela e seus índices possuem uma quantidade significativa de espaço não utilizado, deverá ser utilizado o procedimento *SYSCS_UTIL.SYSCS_COMPRESS_TABLE*, ou o procedimento *SYSCS_UTIL.SYSCS_INPLACE_COMPRESS_TABLE*, para recuperar o espaço não utilizado. O procedimento *SYSCS_COMPRESS_TABLE* garante recuperar a quantidade máxima de espaço livre, às custas de criar temporariamente novas tabelas e índices

antes do procedimento ser efetivado. O procedimento SYSCS_INPLACE_COMPRESS tenta recuperar o espaço dentro da mesma tabela, mas não pode garantir que irá recuperar todo o espaço disponível. A diferença entre estes procedimentos é que, diferentemente de SYSCS_COMPRESS_TABLE, o procedimento SYSCS_INPLACE_COMPRESS não utiliza arquivos temporários, e move linhas dentro do mesmo conglomerado.

Como exemplo, após determinar que a tabela VÔOS_DISPONIBILIDADE e seus índices relacionados possuem muito espaço não utilizado, este espaço poderá ser recuperado através do seguinte comando:

```
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('APP', 'VÔOS_DISPONIBILIDADE', 0);
```

O terceiro parâmetro no procedimento SYSCS_UTIL.SYSCS_COMPRESS_TABLE() determina se a operação será executada em modo seqüencial ou não seqüencial. Se for especificado 0 para o terceiro argumento do procedimento, a operação executará no modo não seqüencial. No modo seqüencial, o Derby comprime a tabela e seus índices seqüencialmente, um de cada vez. A compressão seqüencial utiliza menos memória e espaço em disco, mas é mais lenta. Para forçar a operação executar no modo seqüencial, deverá ser colocado um valor diferente de zero do terceiro argumento. O exemplo a seguir mostra como forçar o procedimento a executar no modo seqüencial:

```
call SYSCS_UTIL.SYSCS_COMPRESS_TABLE('APP', 'VÔOS_DISPONIBILIDADE', 1);
```

Para obter mais informações sobre este comando, deve ser consultado o *Manual de Referência do Derby*.

Marcas Registradas

Os seguintes termos são marcas registradas de outras empresas e foram utilizados em pelo menos um dos documentos da biblioteca de documentação do Apache Derby:

Cloudscape, DB2, DB2 Universal Database, DRDA e IBM são marcas registradas da International Business Machines Corporation nos EUA, outros países, ou ambos.

Microsoft, Windows, Windows NT e o logotipo do Windows são marcas registradas da Microsoft Corporation nos EUA, outros países, ou ambos.

Java e todas as marcas registradas baseadas no Java são marcas registradas da Sun Microsystems, Inc. nos EUA, outros países, ou ambos.

UNIX é uma marca registrada do *The Open Group* nos EUA e outros países.

Outros nomes de empresas, produtos ou serviços podem ser marcas registradas ou marcas de serviços de terceiros.