



Sybase® jConnect for JDBC™
Guide de référence du programmeur

jConnect for JDBC
Versions 4.2 et 5.2

Réf. du document : 38166-01-0520-01

Dernière mise à jour : Octobre 1999

Copyright © 1989-1999 de Sybase, Inc. Tous droits réservés.

Cette publication concerne jConnect pour JDBC versions 4.2 et 5.2, partie intégrante du logiciel de gestion de bases de données de Sybase et toutes les versions ultérieures qui ne feraient pas l'objet d'une réédition de la documentation ou de la publication de notes de mise à jour. Les informations contenues dans ce document pourront faire l'objet de modifications sans préavis. Le logiciel décrit est fourni sous contrat de licence et il ne peut être utilisé ou copié que conformément aux termes de ce contrat.

Pour commander des ouvrages supplémentaires ou acquérir des droits de reproduction, si vous habitez aux Etats-Unis ou au Canada, appelez notre Service Clients au (001-800) 685-8225, télécopie (001-617) 229-9845.

Les clients ne résidant ni aux Etats-Unis ni au Canada et qui disposent d'un contrat de licence pour les U.S.A. peuvent joindre notre Service Clients par télécopie. Ceux qui ne bénéficient pas de cette licence doivent s'adresser à leur revendeur Sybase ou au distributeur le plus proche. Les mises à jour du logiciel ne sont fournies qu'à des dates d'édition périodiques. Tout ou partie de cette publication ne peut être reproduit, transmis ou traduit, sous quelque forme ou par quelque moyen que ce soit (électronique, mécanique, manuel, optique ou autre) sans l'accord écrit préalable de Sybase, Inc.

Sybase, le logo Sybase, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Translator, APT-Library, Backup Server, ClearConnect, Client-Library, Client Services, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, E-Anywhere, E-Whatever, Embedded SQL, EMS, Enterprise Application Server, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Gateway Manager, ImpactNow, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MySupport, Net-Gateway, Net-Library, NetImpact, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, RW-Library, S Designer, S-Designer, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILLS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, Transact-SQL, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server et XP Server sont des marques de Sybase, Inc. 9/99

Unicode et le logo Unicode sont des marques déposées d'Unicode, Inc.

Tous les autres noms de produit, société ou marque apparaissant dans ce document peuvent appartenir à des tiers.

Toute utilisation, duplication ou divulgation par le gouvernement est soumise aux restrictions stipulées au sous-paragraphe (c)(1)(ii) du DFARS 52.227-7013 pour le Ministère américain de la défense (DOD) et à celles stipulées dans la FAR 52.227-19(a)-(d) pour les administrations civiles.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608, Etats-Unis d'Amérique.

Table des matières

Préface	vii
CHAPITRE 1	Introduction 1
	Présentation de JDBC..... 2
	Présentation de jConnect..... 4
CHAPITRE 2	Programmation..... 5
	Configuration de jConnect..... 6
	Configuration de la version de jConnect 6
	Appel du pilote jConnect 10
	Etablissement d'une connexion..... 12
	Définition des propriétés de connexion 12
	Connexion à Adaptive Server Enterprise 18
	Connexion à Adaptive Server Anywhere..... 20
	Connexion à un serveur à l'aide de JNDI 21
	Mise en oeuvre des plug-ins de sockets personnalisés 27
	Propriété de connexion SYB SOCKET_FACTORY 28
	Création et configuration d'un socket personnalisé..... 28
	Gestion de l'internationalisation et de la localisation..... 32
	Convertisseurs de jeu de caractères jConnect..... 32
	Utilisation des bases de données 38
	Mise en oeuvre de la reprise sur le serveur secondaire en mode haute disponibilité 38
	Appels de procédure à distance interserveur..... 44
	Accès aux métadonnées d'une base de données..... 45
	Utilisation de curseurs dans des jeux de résultats 46
	Support pour les mises à jour par batch..... 57
	Mise à jour de la base de données à partir du jeu de résultats d'une procédure stockée 60
	Utilisation des types de données..... 61
	Mise en oeuvre de fonctionnalités avancées 66
	Utilisation de notifications d'événement 66
	Gestion des messages d'erreur 69

Stockage d'objets Java en tant que données de colonne dans une table	74
Chargement de classes dynamique	79
Support des extensions JDBC 2.0 Optional Package	83
Restrictions, limitations et non-conformités aux normes JDBC	95
Réglage pour multithread	95
Utilisation de ResultSet.setCursorName()	96
Utilisation de setLong() avec des valeurs de paramètre élevées	96
Utilisation d'instructions COMPUTE	96
Exécution de procédures stockées	97

CHAPITRE 3	Résolution des incidents	99
	Débogage avec jConnect	100
	Obtention d'une instance de la classe Debug	100
	Activation du débogage dans votre application	101
	Désactivation du débogage dans votre application	101
	Définition de CLASSPATH pour le débogage	101
	Utilisation des méthodes de débogage	102
	Capture des communications TDS	104
	Propriété de connexion PROTOCOL_CAPTURE	104
	Méthodes pause() et resume() dans la classe Capture	105
	Erreurs lors d'échecs de connexion	106
	Connexion refusée à la passerelle	106
	Connexion à un serveur SQL 4.9.2 impossible	107
	Utilisation de la mémoire dans les applications jConnect	108
	Erreurs de procédure stockée	109
	Le RPC renvoie moins de paramètres de sortie que prévu ..	109
	Erreur fetch/state en cas de renvoi de paramètres Output par une procédure stockée	109
	Procédure stockée exécutée en mode non chaîné	109
	Erreur de mise en oeuvre de socket personnalisé	111

CHAPITRE 4	Optimisation des performances	113
	Optimisation des performances de jConnect	114
	Remise à l'échelle de BigDecimal	115
	Propriété de connexion REPEAT_READ	115
	Conversion d'un jeu de caractères	116
	Optimisation des performances des instructions préparées en SQL dynamique	117
	Instructions préparées et procédures stockées	118
	Instructions préparées dans des applications portables	119
	Instructions préparées dans des applications dotées d'extensions jConnect	119

	Connection.prepareStatement()	121
	Propriété de connexion DYNAMIC_PREPARE	121
	SybConnection.prepareStatement()	123
	Performances du curseur	124
	Propriété de connexion LANGUAGE_CURSOR	124
CHAPITRE 5	Migration des applications jConnect	127
	Migration des applications jConnect	128
	Migration des applications vers jConnect 4.1	128
	Migration des applications vers jConnect 5.x	128
	Migration des applications vers jConnect 4.2 et 5.2	129
	Modifications apportées aux extensions Sybase	131
	Exemple	131
	Noms de méthodes modifiés	132
	Classe Debug	133
CHAPITRE 6	Passerelles de serveur Web	135
	Introduction	136
	Encapsulation par TDS	136
	Configuration de jConnect et de passerelles	137
	Utilisation de la passerelle Cascade	139
	Installation de la passerelle Cascade	140
	Démarrage de la passerelle Cascade	140
	Test de la passerelle Cascade	141
	Examen du fichier index.html	142
	Exécution de l'applet Isql exemple	143
	Définition d'une connexion à la passerelle Cascade	144
	Utilisation du servlet d'encapsulation par TDS	145
	Configuration système requise pour le servlet d'encapsulation par TDS	146
	Installation du servlet	146
	Appel du servlet	148
	Suivi des sessions TDS actives	148
	Reprise d'une session TDS	149
	Encapsulation TDS et Netscape Enterprise Server 3.5.1 sous Solaris	149
ANNEXE A	Messages d'exception et d'avertissement SQL	151
ANNEXE B	Programmes exemple jConnect	171
	Exécution d'IsqlApp	172
	Exécution de programmes et codes exemple de jConnect	174

	Applications exemple	174
	Code exemple	175
INDEX	177

Préface

Le présent document, *Sybase jConnect for JDBC - Guide de référence du programmeur*, décrit le produit jConnect for JDBC et explique comment l'utiliser pour accéder aux données stockées dans les systèmes de gestion de bases de données relationnelles.

A qui s'adresse ce manuel ?

Ce manuel est destiné aux programmeurs d'applications orientées bases de données qui ont une bonne connaissance du langage de programmation Java, de JDBC et de Transact-SQL[®], version du langage SQL (Structured Query Language) développée par Sybase.

Documentation jConnect

Pour plus d'informations sur jConnect, reportez-vous aux documents suivants :

- *Sybase jConnect for JDBC - Guide d'installation*
- *Sybase jConnect for JDBC - Notes de mise à jour*
- La documentation javadoc sur les extensions jConnect vers JDBC. Le Java Development Kit (JDK) de JavaSoft contient un script javadoc pour l'extraction des commentaires des fichiers du code source. Ce script a été utilisé pour extraire des fichiers source de jConnect la documentation relative aux packages, classes et méthodes jConnect. Lorsque vous installez jConnect en choisissant l'option d'installation complète ou javadocs, les informations javadoc sont placées dans le répertoire *javadocs* :

Répertoire_installation/docs/en/javadocs

Autres sources d'information

Pour plus d'informations, consultez le CD-ROM Sybase Technical Library et le site Web Technical Library :

- Le CD-ROM Technical Library, qui contient des manuels sur les produits et des documents techniques, est livré avec le logiciel. L'explorateur DynaText, également livré sur le CD-ROM, permet d'accéder aux informations techniques sur les produits dans un format facile à utiliser.

Pour plus d'informations sur l'installation et le démarrage de Technical Library, reportez-vous au manuel *Technical Library - Guide d'installation*.

- Le site Web Technical Library comprend le site Manuals, version HTML du CD-ROM Technical Library, accessible via un explorateur Web standard. Vous trouverez en outre des liens vers le site Web Technical Documents (précédemment appelé Tech Info Library), la page Solved Cases et les Newsgroups Sybase/Powersoft.

Pour accéder au site Web Technical Library, rendez-vous sur le site support.sybase.com et cliquez sur l'onglet Manuals. Cliquez ensuite sur le lien qui vous convient.

Certifications Sybase sur le Web

La documentation technique disponible sur le site Web de Sybase est mise à jour fréquemment.

Pour obtenir les dernières informations sur les certifications de produits et/ou les consolidations d'EBF :

Consultez le site Web Technical Documents, à l'adresse :

techinfo.sybase.com

Dans la section Browse section, cliquez sur What's Hot.

Explorez le domaine qui vous intéresse : Hot Docs qui couvre différents sujets ou Hot Links pour consulter des informations techniques, accéder aux rapports de certification, connaître les partenaires certifiés, etc.

Si vous êtes enregistré comme utilisateur Support**Plus** :

Consultez le site Web Technical Documents, à l'adresse :

techinfo.sybase.com

Dans la section Browse section, cliquez sur What's Hot.

Sous Hot Links, cliquez sur EBF Rollups.

Vous pouvez rechercher les EBF qui vous intéressent dans Technical Documents et les télécharger à l'aide d'ESD (Electronic Software Distribution).

Suivez les instructions associées au service en ligne Support*Plus*SM Online Services.

Si vous n'êtes pas enregistré comme utilisateur Support*Plus* et que vous souhaitez le devenir :

Vous pouvez vous enregistrer en suivant les instructions fournies sur le site Web.

Pour utiliser Support*Plus*, vous avez besoin :

- d'un explorateur Web, tel que Netscape Navigator version 1.2 ou supérieure, qui prenne en charge Secure Sockets Layer (SSL) ;
- d'une licence de support active ;
- d'un contact nommé au Support Technique ;
- d'un ID utilisateur et d'un mot de passe.

Que vous soyez ou non enregistré comme utilisateur Support*Plus* :

Vous pouvez accéder aux Technical Documents de Sybase. Les rapports de certification font partie des fonctions documentées sur ce site.

Consultez le site Technical Documents, à l'adresse :

techinfo.sybase.com

Dans la section Browse, cliquez sur What's Hot.

Cliquez sur le sujet qui vous intéresse.

Conventions

Les conventions typographique suivantes sont utilisées dans ce manuel :

- Les classes, interfaces, méthodes et packages apparaissent à l'intérieur d'un paragraphe dans la police **bold Helvetica**. Par exemple :

classe **SybConnection**

interface **SybEventHandler**

méthode **setBinaryStream()**

package **com.sybase.jdbcx**

- Les objets, instances et noms de paramètres apparaissent en italique. Par exemple :

"Dans l'exemple suivant, *ctx* est un objet **DirContext**".

"eventHdlr" est une instance de la classe **SybEventHandler** que vous mettez en oeuvre".

"Le paramètre *classes* est une chaîne qui répertorie les classes à déboguer."

- Les fragments de code apparaissent dans une police à espacement non proportionnel. Les variables incluses dans des fragments de code (termes se substituant aux valeurs à insérer) apparaissent en italique. Par exemple :

```
Connection con = DriverManager.getConnection("jdbc:  
sybase:Tds:hôte:port", props);
```

Si vous avez besoin d'aide

Pour chaque installation Sybase bénéficiant d'un contrat de maintenance, une ou plusieurs personnes désignées sont autorisées à contacter le Support Technique de Sybase. Si vous avez des questions ou besoin d'aide, demandez à la personne désignée de contacter le Support Technique ou la filiale Sybase la plus proche.

Introduction

Le présent chapitre décrit jConnect for JDBC et détaille ses concepts et ses composants.

Il se compose des sections suivantes :

Nom	Page
Présentation de JDBC	2
Présentation de jConnect	4

Présentation de JDBC

JDBC (Java Database Connectivity), de Java Software Division (Sun Microsystems, Inc.) est une spécification d'interface de programmation d'application (API) permettant aux applications Java d'accéder à des systèmes de gestion de bases de données multiples au moyen du langage SQL (Structured Query Language). Le gestionnaire de pilotes JDBC prend en charge les pilotes qui se connectent à différentes bases de données.

L'API JDBC standard intègre un ensemble d'interfaces vous permettant d'établir des connexions à des bases de données, d'exécuter des commandes SQL et de traiter les résultats. Les interfaces sont décrites dans le [tableau 1-1](#).

Tableau 1-1 : Interfaces JDBC

Interface	Description
<code>java.sql.Driver</code>	Localise le pilote d'une URL de base de données.
<code>java.sql.Connection</code>	Etablit une connexion à une base de données spécifique.
<code>java.sql.Statement</code>	Exécute les instructions SQL.
<code>java.sql.PreparedStatement</code>	Gère les instructions SQL paramétrées.
<code>java.sql.CallableStatement</code>	Gère les appels de procédure stockée des bases de données.
<code>java.sql.ResultSet</code>	Collecte les résultats des instructions SQL.
<code>java.sql.DatabaseMetaData</code>	Affiche des informations sur le SGBD et la base de données d'une connexion.
<code>java.sql.ResultSetMetaData</code>	Affiche des informations décrivant les attributs d'un jeu de résultats

Chaque système de gestion de bases de données relationnelles nécessite un pilote pour mettre en oeuvre ces interfaces. Tous les appels JDBC sont transmis au gestionnaire de pilotes JDBC, lequel transmet l'appel au pilote spécifié.

Il existe quatre types de pilote JDBC :

- *1) Pont JDBC-ODBC* : traduit les appels JDBC en appels ODBC et les transmet à un pilote ODBC. Certains logiciels ODBC doivent être résidents sur la machine cliente. Une partie du code de la base de données cliente peut également résider sur la machine cliente.
- *2) API native/partiellement écrite en Java* : convertit les appels JDBC en appels propres à la base de données. Ce pilote, qui communique directement avec le serveur de base de données, requiert également du code binaire sur la machine cliente.

- 3) *Protocole réseau/tout Java* : communique avec un middle-tier server utilisant un protocole de réseau indépendant SGBD. Une passerelle de niveau intermédiaire (middle-tier) convertit alors la demande dans un protocole spécifique du fournisseur.
- 4) *Protocole natif/tout Java* : convertit les appels JDBC dans un protocole de SGBD spécifique du fournisseur, permettant aux applications clientes de communiquer directement avec le serveur de base de données.

Présentation de jConnect

jConnect de Sybase est un pilote JDBC extrêmement performant. Il est à la fois :

- un pilote de protocole réseau/tout Java au sein d'un environnement à trois niveaux, et
- un pilote de protocole natif/tout Java au sein d'un environnement à deux niveaux.

Le protocole utilisé par jConnect est TDS 5.0 (Tabular Data Stream™, version 5), le protocole natif des applications Adaptive Server® et Open Server™. jConnect met en oeuvre la norme JDBC afin d'assurer une connectivité optimale à toute la gamme des produits Sybase et permettre d'accéder à 25 systèmes d'entreprise, notamment :

- Adaptive Server Enterprise
- Adaptive Server Anywhere
- Adaptive Server IQ (anciennement Sybase IQ™)
- Replication Server®
- OmniConnect™

Remarque Depuis le changement de nom de Sybase SQL Server™ en Adaptive Server Enterprise, Sybase peut utiliser les noms Adaptive Server et Adaptive Server Enterprise pour désigner collectivement toutes les versions supportées de Sybase SQL Server et d'Adaptive Server Enterprise.

En outre, jConnect for JDBC peut accéder à Oracle, à l'AS/400 et à d'autres sources de données via Sybase DirectConnect™.

Dans certains cas, la mise en oeuvre jConnect's de JDBC ne correspond pas entièrement aux spécifications de JDBC 1.x ou 2.x. Pour plus d'informations, reportez-vous à la section "[Restrictions, limitations et non-conformités aux normes JDBC](#)", page 95.

Le présent chapitre décrit les éléments principaux et les conditions requises pour la programmation avec jConnect for JDBC. Il explique comment appeler le pilote jConnect, définir les propriétés de connexion et établir une connexion à un serveur de bases de données, et décrit l'utilisation des fonctionnalités de jConnect.

Remarque Pour plus d'informations sur la programmation JDBC consultez le site :

<http://java.sun.com/jdbc>.

Pour accéder au document *JDBC Guide: Getting Started* pour JDBC 1.0, consultez le site :

<http://java.sun.com/products/jdk/1.1/docs/guide/jdbc>.

Pour accéder au document *JDBC Guide: Getting Started* pour JDBC 2.0, consultez le site :

<http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/>.

Ce chapitre traite des sujets suivants :

Nom	Page
Configuration de jConnect	6
Etablissement d'une connexion	12
Mise en oeuvre des plug-ins de sockets personnalisés	27
Gestion de l'internationalisation et de la localisation	32
Utilisation des bases de données	38
Mise en oeuvre de fonctionnalités avancées	66
Restrictions, limitations et non-conformités aux normes JDBC	95

Configuration de jConnect

Cette section décrit les tâches que vous devez exécuter avant d'utiliser jConnect.

Configuration de la version de jConnect

Plusieurs versions de jConnect existent ; vous devez connaître le numéro de version utilisée pour déterminer les éléments suivants :

- la valeur par défaut de la propriété de connexion LANGUAGE ;
- les fonctionnalités disponibles spécifiques de la version ;
- le jeu de caractères par défaut, si aucun jeu n'est défini par la propriété de connexion CHARSET ;
- la valeur par défaut de la propriété de connexion CHARSET_CONVERTER ;
- la valeur par défaut de la propriété de connexion CANCEL_ALL, utilisée pour définir l'action de **Statement.cancel()** qui, par défaut, annule l'objet pour lequel elle est appelée, ainsi que tous les objets **Statement** en cours d'exécution et en attente de résultats.

Le répertoire des différentes versions disponibles de jConnect et leurs caractéristiques.

Tableau 2-1 : Valeurs et caractéristiques des versions de jConnect

Version	Caractéristiques	Commentaires
VERSION_5	<ul style="list-style-type: none">• La valeur par défaut de la propriété de connexion LANGUAGE est NULL.• Si la propriété de connexion CHARSET ne contient aucun jeu de caractères, jConnect utilise le jeu par défaut de la base de données. La valeur par défaut de CHARSET_CONVERTER est la classe PureConverter.• Par défaut, Statement.cancel() annule uniquement l'objet Statement pour lequel elle est appelée.• Les méthodes JDBC 2.0 permettent de stocker et d'extraire des objets en tant que données de colonne.	<p>VERSION_5 est la version par défaut de jConnect 5.0.</p> <p>Pour plus d'informations, voir les commentaires de la VERSION_4.</p>

Version	Caractéristiques	Commentaires
VERSION_4	<ul style="list-style-type: none"> La valeur par défaut de la propriété de connexion LANGUAGE est NULL. Si la propriété de connexion CHARSET ne contient aucun jeu de caractères, jConnect utilise le jeu par défaut de la base de données. La valeur par défaut de CHARSET_CONVERTER est la classe PureConverter. Par défaut, Statement.cancel() annule uniquement l'objet Statement pour lequel elle est appelée. Les méthodes JDBC 2.0 permettent de stocker et d'extraire des objets en tant que données de colonne. 	<p>VERSION_2 est la version par défaut pour jConnect version 4.0 et inférieures.</p> <p>La langue dans laquelle apparaissent les messages émis par le serveur dépend de la configuration de votre environnement de travail. Les langues supportées sont les suivantes : chinois, américain, français, allemand, japonais, coréen, portugais et espagnol.</p> <p>L'action par défaut de Statement.cancel() est conforme à JDBC.</p> <p>Vous pouvez définir l'action de Statement.cancel() à partir de la propriété CANCEL_ALL. Reportez-vous à la section “Propriété de connexion CANCEL_ALL”, page 9.</p> <p>Pour plus d'informations sur les objets Java en tant que données de colonne, reportez-vous à la section “Stockage d'objets Java en tant que données de colonne dans une table”, page 74.</p>
VERSION_3	<ul style="list-style-type: none"> La valeur par défaut de la propriété de connexion LANGUAGE est us_english (américain). Si la propriété CHARSET ne contient aucun jeu de caractères, jConnect utilise le jeu par défaut de la base de données. La valeur par défaut de CHARSET_CONVERTER est la classe PureConverter. Par défaut, Statement.cancel() annule l'objet pour lequel elle est appelée, ainsi que tous les objets Statement déjà en cours d'exécution et en attente de résultats. 	<p>VERSION_2 est la version par défaut.</p> <p>Voir les commentaires de la VERSION_2.</p>

Version	Caractéristiques	Commentaires
VERSION_2	<ul style="list-style-type: none"> La valeur par défaut de la propriété de connexion LANGUAGE est us_english (américain). Si la propriété CHARSET ne contient aucun jeu de caractères, iso_1 est pris par défaut. La valeur par défaut de la propriété CHARSET_CONVERTER est la classe TruncationConverter, à moins que la propriété de connexion CHARSET ne définisse un jeu de caractères codés sur un ou plusieurs octets. Dans ce cas, la valeur par défaut de CHARSET_CONVERTER devient la classe PureConverter. Par défaut, Statement.cancel() annule l'objet pour lequel elle est appelée, ainsi que tous les objets Statement déjà en cours d'exécution et en attente de résultat. 	<p>VERSION_2 est la version par défaut de jConnect 2.x.</p> <hr/> <p>Remarque VERSION_5 est la version par défaut de jConnect version 5.x.</p> <hr/> <p>La propriété de connexion LANGUAGE détermine la langue dans laquelle apparaissent les messages issus du pilote jConnect et du serveur.</p> <p>Pour plus d'informations sur les propriétés de connexion CHARSET et CHARSET_CONVERTER, reportez-vous à la section “Convertisseurs de jeu de caractères jConnect”, page 32.</p> <p>L'action par défaut de la VERSION_2 de Statement.cancel() est conforme à JDBC. Vous pouvez définir l'action de Statement.cancel() à partir de la propriété CANCEL_ALL. Reportez-vous à la section “Propriété de connexion CANCEL_ALL”, page 9.</p>

Les valeurs de version sont des constantes issues de la classe **SybDriver**. Lorsque vous vous référez à la constante de la version, utilisez la syntaxe ci-dessous :

```
com.sybase.jdbcx.SybDriver.VERSION_5
```

Pour définir la version de jConnect, utilisez **SybDriver.setVersion()**. Les codes exemples ci-dessous montrent comment charger le pilote jConnect et définir la version.

Pour jConnect 4.x :

```
import com.sybase.jdbcx.SybDriver;
SybDriver sybDriver = (SybDriver)
    Class.forName ("com.sybase.jdbc.SybDriver").newInstance();
sybDriver.setVersion
    (com.sybase.jdbcx.SybDriver.VERSION_4);
DriverManager.registerDriver(sybDriver);
```

Pour jConnect 5.x :

```
import com.sybase.jdbcx.SybDriver;
SybDriver sybDriver = (SybDriver)
    Class.forName
    ("com.sybase.jdbc2.jdbc.SybDriver").newInstance();
```

```
sybDriver.setVersion  
    (com.sybase.jdbcx.SybDriver.VERSION_5);  
DriverManager.registerDriver(sybDriver);
```

Vous pouvez appeler **setVersion()** autant de fois que nécessaire pour modifier la version utilisée. Les nouvelles connexions héritent des caractéristiques associées à la version définie au moment où la connexion est établie. Le changement de version lors d'une session n'a aucune incidence sur la connexion en cours.

Pour remplacer la version définie par **SybDriver** pour une connexion particulière, vous pouvez utiliser la propriété de connexion **JCONNECT_VERSION** (voir la section suivante).

Propriété de connexion JCONNECT_VERSION

La propriété de connexion **JCONNECT_VERSION** vous permet de définir une version différente de **jConnect** pour une session particulière. Il suffit pour cela d'attribuer les valeurs 2, 3, 4 ou 5 à cette propriété, en fonction des caractéristiques voulues (voir le [tableau 2-1](#)).

Propriété de connexion CANCEL_ALL

La propriété de connexion **CANCEL_ALL** est de type booléen ; elle définit l'action de la méthode **Statement.cancel()**.

Remarque Dans les versions **jConnect 4.0** et versions inférieures, la valeur par défaut de **CANCEL_ALL** est "true". Dans **jConnect version 4.1** et versions supérieures, par souci de conformité à la spécification **JDBC**, si vous attribuez une valeur au moins égale à "4" à la propriété de connexion **JCONNECT_VERSION**, la valeur par défaut de **CANCEL_ALL** est "false".

La valeur de la propriété **CANCEL_ALL** produit les effets suivants sur la méthode **Statement.cancel()** :

- Si **CANCEL_ALL** a la valeur "false," **Statement.cancel()** annule uniquement l'objet **Statement** pour lequel elle est appelée. Par conséquent, si **stmtA** est un objet **Statement**, **stmtA.cancel()** annule l'exécution de l'instruction SQL associée à **stmtA** dans la base de données ; aucune autre instruction n'est concernée. **stmtA** est annulé, qu'il se trouve en cache (en attente d'exécution) ou qu'il soit en cours d'exécution (en attente de résultats).

- Si `CANCEL_ALL` a la valeur "true", **Statement.cancel()** annule non seulement l'objet pour lequel elle est appelée, mais également tous les objets **Statement** de la même connexion qui sont déjà en cours d'exécution et en attente de résultats.

L'exemple suivant attribue la valeur "false" à la propriété `CANCEL_ALL`. Ici, *props* est un objet **Properties** qui définit les propriétés de connexion.

```
...
props.put("CANCEL_ALL", "false");
```

Remarque Pour annuler l'exécution de tous les objets **Statement** sur une connexion, quel que soit leur état d'exécution sur le serveur, utilisez la méthode **SybConnection.cancel()**.

Appel du pilote jConnect

Il existe deux façons d'enregistrer et d'appeler le pilote jConnect de Sybase :

Methode 1

Pour jConnect 4.x :

```
Class.forName("com.sybase.jdbc.SybDriver").newInstance();
```

Pour jConnect 5.x :

```
Class.forName("com.sybase.jdbc2.jdbc.SybDriver").newInstance();
```

Methode 2

Ajoutez le pilote jConnect dans la propriété système **jdbc.drivers**. Au démarrage, la classe **DriverManager** tente de charger les pilotes répertoriés dans la propriété **jdbc.drivers**. Mais cette approche est moins efficace que la précédente. Vous pouvez répertorier les pilotes relatifs à cette propriété, séparés par deux points (:). Les codes exemples ci-dessous montrent comment ajouter un pilote dans la propriété **jdbc.drivers** au sein d'un programme.

Pour jConnect 4.x :

```
Properties sysProps = System.getProperties();
String drivers = "com.sybase.jdbc.SybDriver";
String oldDrivers =
sysProps.getProperty("jdbc.drivers");
if (oldDrivers != null)
    drivers += ":" + oldDrivers;
sysProps.put("jdbc.drivers", drivers.toString());
```


Pour jConnect 5.x :

```
Properties sysProps = System.getProperties();
String drivers = "com.sybase.jdbc2.jdbc.SybDriver";
String oldDrivers =
sysProps.getProperty("jdbc.drivers");
if (oldDrivers != null)
    drivers += ":" + oldDrivers;
sysProps.put("jdbc.drivers", drivers.toString());
```

Remarque applets Java ne reconnaissent pas **System.getProperties()**. A la place, utilisez la méthode **Class.forName()**.

Etablissement d'une connexion

Cette section explique comment établir une connexion à une base de données Adaptive Server Enterprise ou Adaptive Server Anywhere à l'aide de jConnect.

Définition des propriétés de connexion

Le [tableau 2-2](#) répertorie les propriétés de connexion de jConnect et indique leur valeur par défaut. *Ces dernières doivent être définies avant toute tentative de connexion.*

Il existe deux façons de définir des propriétés de connexion de pilote :

- à l'aide de la méthode **DriverManager.getConnection()**, dans votre application,
- lorsque vous définissez l'URL.

Remarque Les propriétés de connexion définies dans une application à l'aide de la méthode **DriverManager.getConnection()** prévalent sur celles indiquées dans l'URL.

Pour obtenir la liste des propriétés d'un pilote, utilisez **Driver.getDriverPropertyInfo(String url, Properties props)**, qui retourne une table d'objets **DriverPropertyInfo**. Cette table affiche :

- les propriétés du pilote,
- la configuration courante définissant les propriétés du pilote,
- l'URL et les **props** transférées.

Aucune distinction majuscules/minuscules n'est effectuée dans les noms de propriété de connexion (jConnect utilise la méthode **String.equalsIgnoreCase(String)** pour comparer les noms).

Tableau 2-2 : Propriétés de connexion

Propriété	Description	Valeur par défaut
APPLICATIONNAME	Propriété définie par l'utilisateur. Le serveur peut être configuré pour interpréter la valeur associée à cette propriété.	NULL
CANCEL_ALL	Détermine l'action de la méthode <code>Statement.cancel()</code> . Reportez-vous à la section “Propriété de connexion CANCEL_ALL”, page 9 .	Dépend de la version utilisée. (Reportez-vous à la section “Configuration de la version de jConnect”, page 6 .)
CHARSET	Définit le jeu de caractères des chaînes acheminées par TDS. Tout jeu de caractères spécifié doit correspondre à l'un des jeux répertoriés dans <i>syscharsets</i> . Si cette propriété a la valeur NULL, jConnect utilise le jeu de caractères par défaut du serveur.	NULL
CHARSET_CONVERTER_CLASS	Utilisez cette propriété pour spécifier la classe du convertisseur de jeu de caractères utilisé par jConnect. jConnect utilise le paramètre de version de <code>SybDriver.setVersion()</code> pour déterminer la classe à utiliser par défaut. Pour plus d'informations, reportez-vous au “Sélection d'un convertisseur de jeu de caractères”, page 33 .	Dépend de la version.
CONNECTION_FAILOVER	S'utilise conjointement à JNDI (Java Naming and Directory Interface). Reportez-vous à la section “Propriété de connexion CONNECTION_FAILOVER”, page 24 .	true
DYNAMIC_PREPARE	Détermine si les instructions SQL dynamiques préparées sont précompilées dans la base de données. Reportez-vous à la section “Propriété de connexion DYNAMIC_PREPARE”, page 121 .	false
EXPIRESTRING	Propriété non modifiable qui contient la date d'expiration de la licence. La valeur par défaut est “never” (jamais), sauf si vous disposez d'une copie d'évaluation de jConnect.	Never
HOSTNAME	Nom de l'hôte courant.	Aucun
HOSTPROC	Identifie le processus de l'application sur la machine hôte.	Aucun

Propriété	Description	Valeur par défaut
IGNORE_DONE_IN_PROC	Lorsque cette propriété a la valeur “true,” les résultats des mises à jour intermédiaires (comme dans des procédure stockées) ne sont pas renvoyés, seul le jeu de résultats final l’est.	false
JCONNECT_VERSION	Permet d'activer les caractéristiques spécifiques d'une version. Reportez-vous à la section “Propriété de connexion JCONNECT_VERSION”, page 9.	5
LANGUAGE	Détermine la langue dans laquelle s'affichent les messages jConnect, ainsi que les messages d'erreur renvoyés par le serveur. Elle doit correspondre à l'une des langues répertoriées dans <i>syslanguages</i> .	Dépend de la version. Reportez-vous à la section “Configuration de la version de jConnect”, page 6.
LANGUAGE_CURSOR	Permet à jConnect de remplacer les curseurs de protocole par des curseurs de langue (valeur “true”). Reportez-vous à la section “Performances du curseur”, page 124.	false
LITERAL_PARAMS	Permet de transmettre littéralement des paramètres d’instruction préparés (ne s’applique qu’à Adaptive Server Anywhere). Cette propriété peut être définie à “false” pour toutes les autres bases de données Sybase. Lorsqu’elle a la valeur “true”, les paramètres définis par les méthodes setXXX dans l’interface PreparedStatement sont insérés littéralement dans l’instruction SQL lorsqu’elle celle-ci est exécutée. Lorsque la valeur est “false”, les délimiteurs de paramètre sont maintenus dans l’instruction SQL et les valeurs de paramètres sont transmises séparément au serveur.	false
PACKETSIZE	Taille des paquets réseau.	512
PASSWORD	Mot de passe de connexion. Il est défini automatiquement ou explicitement selon que vous utilisez respectivement la méthode getConnection(String, String, String) ou getConnection(String, Props).	Aucun
PROTOCOL_CAPTURE	Permet de spécifier un fichier devant capturer la communication TDS entre une application et un Adaptive Server.	NULL

Propriété	Description	Valeur par défaut
PROXY	Adresse de passerelle. Pour le protocole HTTP, l'URL est le suivant : <i>http://hôte:port</i> . Pour utiliser le protocole HTTPS qui supporte le cryptage, l'URL est <i>https://hôte:port/alias_servlet</i> .	Aucun
REMOTEPWD	Mots de passe serveur distants permettant l'accès via des appels de procédures à distance interserveur. Reportez-vous à la section " Appels de procédure à distance interserveur ", page 44.	Aucun
REPEAT_READ	Permet au pilote de conserver une copie des colonnes et des paramètres de sortie afin d'autoriser des lectures successives ou dans un ordre quelconque des colonnes. Reportez-vous à la section " Propriété de connexion REPEAT_READ ", page 115.	true
REQUEST_HA_SESSION	Cette propriété indique si le client qui se connecte souhaite démarrer une session haute disponibilité (HD) de reprise sur le serveur secondaire avec un Adaptive Server version 12 ou supérieure configuré pour ce mode. Lorsque cette propriété a la valeur "true", jConnect tente d'effectuer une connexion en mode reprise sur le serveur secondaire. Si vous ne la définissez pas, la session de reprise ne sera pas lancée, même si le serveur est configuré pour ce mode. Une fois que la connexion a été établie, il est impossible de réinitialiser cette propriété. Pour plus de souplesse dans le contrôle des sessions en mode reprise sur le serveur secondaire en environnement haute disponibilité, codez l'application cliente de façon à pouvoir définir REQUEST_HA_SESSION au moment de l'exécution.	false

Propriété	Description	Valeur par défaut
SELECT_OPENS_CURSOR	<p>Si cette propriété a la valeur “true”, les appels vers <code>Statement.executeQuery()</code> génèrent automatiquement un curseur lorsque la requête comporte une clause “FOR UPDATE”.</p> <p>Si <code>Statement.setFetchSize()</code> ou <code>Statement.setCursorName()</code> a déjà été appelée sur la même instruction, la valeur “true” affectée à <code>SELECT_OPENS_CURSOR</code> n’aura aucun effet.</p> <hr/> <p>Remarque Le système risque d’être moins performant si <code>SELECT_OPENS_CURSOR</code> a la valeur “true”.</p> <hr/> <p>Pour plus d’informations sur l’utilisation des curseurs avec <code>jConnect</code>, reportez-vous à la section “Utilisation de curseurs dans des jeux de résultats”, page 46.</p>	false
SERIALIZE_REQUESTS	Lorsque cette propriété a la valeur “true”, <code>jConnect</code> attend les réponses du serveur avant d’envoyer des requêtes supplémentaires.	false
SERVICENAME	Nom du serveur de bases de données principal géré par une passerelle <code>DirectConnect</code> . Permet également d’indiquer la base de données de connexion d’ <code>Adaptive Server Anywhere</code> .	Aucun
SESSION_ID	Lorsque cette propriété est définie, <code>jConnect</code> suppose qu’une application tente de reprendre la communication sur une session TDS existante, maintenue ouverte par la passerelle d’encapsulation par TDS. <code>jConnect</code> passe alors outre la phase initiale de la connexion et fait suivre toutes les requêtes adressées par une application à l’ID de session spécifié.	NULL
SESSION_TIMEOUT	Permet de définir le délai (en secondes) pendant lequel une connexion encapsulée http (créée avec le servlet d’encapsulation par TDS de <code>jConnect</code>) reste ouverte tout en étant inactive. Au terme de ce délai, la connexion est automatiquement fermée. Pour plus d’informations sur le servlet d’encapsulation par TDS, reportez-vous à la page 145.	NULL

Propriété	Description	Valeur par défaut
SQLNITSTRING	Permet de définir un ensemble de commandes à transmettre au serveur de bases de données principal. Il doit s'agir de commandes SQL exécutables à l'aide de la méthode <code>Statement.executeUpdate()</code> .	NULL
SYB SOCKET_FACTORY	<p>Permet à jConnect d'utiliser un socket personnalisé.</p> <p>Vous pouvez attribuer à SYB SOCKET_FACTORY :</p> <ul style="list-style-type: none"> le nom d'une classe qui met en oeuvre <code>com.sybase.jdbcx.SybSocketFactory</code> ou la valeur "DEFAULT" qui instancie un nouveau <code>java.net.Socket()</code>. <p>Reportez-vous à la section "Mise en oeuvre des plug-ins de sockets personnalisés", page 27.</p>	NULL
STREAM_CACHE_SIZE	Taille maximale du cache destiné aux flux de réponse aux instructions.	NULL (taille de cache illimitée)
USE_METADATA	<p>Lorsque la propriété a la valeur "true", un objet <code>DatabaseMetaData</code> est créé et initialisé lors de l'établissement d'une connexion. Cet objet <code>DatabaseMetaData</code> est nécessaire pour établir une connexion à une base de données spécifiée.</p> <p>jConnect utilise <code>DatabaseMetaData</code> dans certaines fonctionnalités, telles que le support de Gestion des transactions distribuées (Distributed Transaction Management) (JTA/JTS) et le Chargement de classe dynamique (Dynamic Class Loading, DCL).</p> <p>Si vous constatez une erreur 010SJ, indiquant que votre application a besoin de métadonnées, installez les procédures stockées (fournies avec jConnect) qui renvoient des métadonnées (reportez-vous à la section "Installing Stored Procedures" du chapitre 3 de <i>jConnect for JDBC - Guide d'installation</i>).</p>	true
USER	<p>ID de connexion.</p> <p>Il est défini automatiquement ou explicitement selon que vous utilisez respectivement la méthode <code>getConnection(String, String, String)</code> ou <code>getConnection(String, Props)</code>.</p>	Aucun
VERSIONSTRING	Informations non modifiables relatives à la version du pilote JDBC.	Version du pilote jConnect

Le code suivant montre comment définir des propriétés de connexion. Vous trouverez d'autres exemples dans les programmes exemple fournis avec jConnect.

```
Properties props = new Properties();
props.put("user", "id_utilisateur");
props.put("password", "mot_de_passe_utilisateur");
/*
 * Si le programme est un applet qui doit accéder à
 * un serveur installé sur un hôte différent
 * de celui du serveur Web, il utilise une
 * passerelle proxy./
props.put("proxy", "localhost:port");
/*
 * Veuillez à définir les propriétés de connexion
 * avant toute tentative de connexion. Il est
 * également possible de définir les propriétés
 * dans l'URL./
Connection con = DriverManager.getConnection
("jdbc:sybase:Tds:hôte:port", props);
```

Connexion à Adaptive Server Enterprise

Dans votre application Java, définissez un URL en utilisant le pilote jConnect pour établir la connexion à un Adaptive Server. Le format élémentaire de l'URL est le suivant :

```
jdbc:sybase:Tds:hôte:port
```

où :

jdbc:sybase est le nom du pilote.

Tds est le protocole de communication Sybase pour Adaptive Server.

hôte:port désigne le nom d'hôte et le numéro de port récepteur d'Adaptive Server. Reportez-vous à la section relative à *SYBASE/interfaces* (UNIX) ou à *%SYBASE%\ini\sql.ini* (Windows) selon l'entrée utilisée par votre base de données ou par Open Server. Pour obtenir les valeurs de *l'hôte:port*, examinez l'entrée "query".

Vous pouvez vous connecter à une base de données spécifique en entrant l'instruction suivante :

```
jdbc:sybase:Tds:hôte:port/basededonnées
```

Remarque Pour vous connecter à une base de données spécifique à l'aide d'Adaptive Server Anywhere 6.x ou de DirectConnect, utilisez les [Propriétés de connexion](#) et remplacez “/database” par le nom de la base de données

Exemple

Le code ci-dessous crée une connexion à un Adaptive Server sur l'hôte “monserveur” par l'intermédiaire du port 3697 :

```
SysProps.put("user", "id_utilisateur");
SysProps.put("password", "mot_de_passe_utilisateur");
String url = "jdbc:sybase:Tds:monserveur:3697";
Connection_con =
    DriverManager.getConnection(url, SysProps);
```

Valeurs de propriété de connexion dans un URL

Vous pouvez spécifier les valeurs de propriété de connexion du pilote jConnect lorsque vous définissez un URL.

Remarque Les propriétés de connexion définies dans une application à l'aide de la méthode **DriverManager.getConnection()** prévalent sur celles indiquées dans l'URL.

Pour définir une propriété de connexion dans l'URL, ajoutez le nom de la propriété et sa valeur à la suite de la définition de l'URL. Utilisez la syntaxe ci-dessous :

```
jdbc:sybase:Tds:hôte:port/basededonnées?
nom_propriété=valeur
```

Pour définir plusieurs propriétés de connexion, séparez-les par le caractère et commercial (&). Par exemple :

```
jdbc:sybase:Tds:monserveur:1234/mabasededonnées?
LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=monhôte
```

Lorsque la valeur d'une des propriétés de connexion contient le caractère “&”, faites précéder ce dernier d'une barre oblique inverse (\). Par exemple, si le nom de votre hôte est “a&bhôte”, utilisez la syntaxe ci-dessous :

```
jdbc:sybase:Tds:monserveur:1234/mabasededonnées?
LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=
a\&bhôte
```

N'utilisez pas de guillemets dans les valeurs de propriété de connexion, même dans le cas de chaînes de caractères. Par exemple, utilisez :

```
HOSTNAME=monhôte
```

et non :

```
HOSTNAME="monhôte"
```

Connexion à Adaptive Server Anywhere

Pour utiliser jConnect avec Adaptive Server Anywhere, il est nécessaire d'effectuer la mise à niveau vers Adaptive Server Anywhere 6.x.

Connexion à Adaptive Server Anywhere 5.x.x

Pour vous connecter à Adaptive Server Anywhere version 5.x.x via jConnect, vous devez exécuter Open Serveur Gateway dbos50, fourni avec Adaptive Server Anywhere.

Remarque La version gratuite d'Adaptive Server Anywhere qui peut être téléchargée depuis le site Web de PowerSoft ne comprend pas Open Server Gateway. Pour vous procurer ce dernier sur CD-ROM, accompagné des DLL nécessaires, appelez Powersoft au (800) 265-4555. Seuls les frais d'expédition et de traitement seront facturés.

- 1 Installez Open Server Gateway 5.5.x3 (ou une version supérieure) et utilisez les DLL d'Open Server, version 11.1.
- 2 Ajoutez une entrée correspondant à la passerelle dans le fichier `%SYBASE%\ini\sql.ini` (par exemple, à l'aide de **sqledit**).
- 3 Démarrez la passerelle en tapant :

```
start dbos50 passerelle-demo
```

où *passerelle-demo* est le nom de la passerelle définie à l'étape 2.

- 4 Lorsque Open Server Gateway est actif, vous pouvez définir une connexion comme suit :

```
jdbc:sybase:Tds:hôte:port
```

hôte est le nom de l'hôte sur lequel Adaptive Server Anywhere et Open Server Gateway sont exécutés et *port* est le numéro de port défini dans *sql.ini*.

Remarque Pour supporter plusieurs bases de données Adaptive Server Anywhere, ajoutez une entrée correspondant à chaque base (en précisant un port différent) à l'aide de **sqledit**, puis exécutez Open Server Gateway pour chaque base de données

Connexion à un serveur à l'aide de JNDI

jConnect version 4.0 et supérieures permet de configurer des connexions à l'aide de JNDI (Java Naming and Directory Interface), ce qui présente les avantages suivants :

- Vous pouvez centraliser les noms d'hôte et les numéros de port de connexion à un serveur. Il n'est pas nécessaire de coder en dur dans une application le nom d'un hôte et le numéro de port spécifiques.
- Vous pouvez centraliser les propriétés de connexion et une base de données par défaut destinées à être utilisées par toutes les applications.
- Vous pouvez utiliser la propriété jConnect CONNECTION_FAILOVER pour gérer les tentatives de connexion infructueuses. Lorsque la propriété CONNECTION_FAILOVER a la valeur "true", jConnect tente de renouveler l'opération avec plusieurs adresses de serveur (nom d'hôte et numéro de port) définies dans l'espace nom JNDI jusqu'à ce qu'une connexion aboutisse.

Pour utiliser jConnect avec JNDI, vous devez vous assurer de la présence de certaines informations dans les services de répertoire accessibles à JNDI ; ces informations doivent également être également se trouver dans la classe **javax.naming.Context**. La section ci-dessous aborde les sujets suivants :

- [URL de connexion permettant l'utilisation de JNDI](#)
- [Informations nécessaires dans un service de répertoire](#)
- [Propriété de connexion CONNECTION_FAILOVER](#)
- [Informations contextuelles destinées à JNDI](#)

URL de connexion permettant l'utilisation de JNDI

Pour que jConnect utilise JNDI afin d'obtenir des informations de connexion, ajoutez "jndi" comme sous-protocole de l'URL après "sybase" :

```
jdbc:sybase:jndi:informations-protocole-à-utiliser-avec-JNDI
```

Tout élément qui suit l'expression "jndi" dans l'URL est traité par JNDI. Par exemple, pour utiliser JNDI conjointement au protocole LDAP (Lightweight Directory Access Protocol), entrez :

```
jdbc:sybase:jndi:ldap://nom_hôte_IDAP:numéro_port/servername=
Sybase11,o=MaSociété,c=FR
```

Cet URL indique à JNDI de récupérer des informations à partir d'un serveur LDAP, fournit le nom d'hôte et le numéro de port du serveur LDAP à utiliser, et signale le nom d'un serveur de bases de données dans un format spécifique du protocole LDAP.

Informations nécessaires dans un service de répertoire

Lorsqu'il est utilisé avec jConnect, JNDI doit renvoyer les informations suivantes relatives au serveur de bases de données cible :

- un nom d'hôte et un numéro de port auxquels se connecter,
- le nom de la base de données à utiliser,
- toute propriété de connexion non définissable par des applications individuelles.

Ces informations doivent être enregistrées selon un format fixe dans tout service de répertoire fournissant des informations de connexion. Ce format se compose d'un identificateur d'objet numérique (OID) qui identifie le type d'informations fournies (par exemple, la base de données cible), et des informations. Reportez-vous au [tableau 2-3](#) pour connaître le format requis.

Tableau 2-3 : Informations de service de répertoire requises par JNDI

Type d'informations	Identificateur d'objet (OID)	Format	Commentaires
Hôte et port	1.3.6.1.4.1.897.4.2.5	TCP#1# <i>nom_hôte</i> <i>numéro_port</i>	Vous pouvez spécifier plusieurs hôtes et ports sous forme d'entrées distinctes, afin d'utiliser CONNECTION_FAILOVER.
Propriété de connexion	1.3.6.1.4.1.897.4.2.10	<i>Prop1=valeur&Prop2=valeur&Prop3=valeur&...</i>	Vous pouvez spécifier plusieurs propriétés de connexion sous forme d'entrées distinctes ou rassemblées sous une même entrée et séparées par le caractère et commercial (&).
Base de données	1.3.6.1.4.1.897.4.2.11	<i>nom_basededonnées</i>	Nom de la base de données à laquelle vous voulez vous connecter. Cette propriété fonctionne comme "/basededonnées" dans l'URL JDBC.
Protocole de connexion	1.3.6.1.4.1.897.4.2.9	Tds	Facultatif, mais doit toujours être "Tds" si vous utilisez un protocole de connexion.

L'exemple ci-dessous montre les informations de connexion entrées pour le serveur de base de données SYBASE11 dans un service de répertoire LDAP :

```
dn: servername=SYBASE11,o=MaSociété,c=FR
servername: SYBASE11
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1266
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1337
1.3.6.1.4.1.897.4.2.5:TCP#1#standby1 4444
1.3.6.1.4.1.897.4.2.10:REPEAT_READ=false&PACKETSIZE=1024
1.3.6.1.4.1.897.4.2.10:CONNECTION_FAILOVER=true
1.3.6.1.4.1.897.4.2.11:pubs2
1.3.6.1.4.1.897.4.2.9:Tds
```

Dans cet exemple, SYBASE11 est accessible par le port 1266 ou 1337 de l'hôte "giotto" ou encore par le port 4444 de l'hôte "standby1". Les deux propriétés de connexion, REPEAT_READ et PACKETSIZE, sont définies dans une seule entrée. La propriété de connexion CONNECTION_FAILOVER est définie dans une entrée distincte. Les applications qui accèdent au serveur SYBASE11 sont initialement connectées à la base de données *pubs2*. Il n'est pas nécessaire de préciser le protocole de connexion. Toutefois, s'il l'est, ce dernier doit être "Tds", et non "TDS".

Propriété de connexion **CONNECTION_FAILOVER**

La propriété de connexion **CONNECTION_FAILOVER** est de type booléen. Elle est utilisée lorsque jConnect passe par JNDI pour obtenir des informations de connexion.

Si **CONNECTION_FAILOVER** a la valeur “true”, jConnect effectue plusieurs tentatives de connexion à un serveur. En cas d'échec d'une première tentative de connexion à un serveur via un hôte et un numéro de port spécifiés, jConnect sollicite JNDI pour l'hôte et le numéro de port suivants associés au serveur, puis il retente la connexion. Les tentatives de connexion sont effectuées dans l'ordre en prenant tous les hôtes et les ports associés au serveur.

Prenons par exemple **CONNECTION_FAILOVER** dotée de la valeur “true” et un serveur de bases de données associé aux hôtes et numéros de port suivants, comme dans l'exemple LDAP plus haut :

```
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1266
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1337
1.3.6.1.4.1.897.4.2.5:TCP#1#standby1 4444
```

Pour établir une connexion au serveur, jConnect tente de se connecter au port 1266 sur l'hôte “giotto”. Si la tentative échoue, jConnect renouvelle l'opération avec le port 1337 de “giotto”. Si vous obtenez le même résultat, jConnect tente alors de se connecter au port 4444 de l'hôte “standby1”.

La valeur par défaut de **CONNECTION_FAILOVER** est “true”.

Si **CONNECTION_FAILOVER** a la valeur “false”, jConnect tente de se connecter à un premier hôte et numéro de port. Si la tentative échoue, jConnect envoie une exception SQL et s'arrête là.

Informations contextuelles destinées à JNDI

L'utilisation de jConnect avec JNDI requiert une bonne connaissance de la spécification JNDI de Sun Microsystems. Rendez-vous sur le site Web à l'adresse suivante :

<http://java.sun.com/products/jndi>

En particulier, il convient de vérifier que les propriétés d'initialisation requises sont définies dans **javax.naming.directory.DirContext**. Ces propriétés sont définissables au niveau système ou à l'exécution.

Il existe deux propriétés principales :

- **Context.INITIAL_CONTEXT_FACTORY**

Cette propriété prend le nom de classe complet du `CONTEXT_FACTORY` initial que JNDI doit utiliser. Elle désigne le pilote JNDI indiqué dans l'URL associé à la propriété `Context.PROVIDER_URL`

- `Context.PROVIDER_URL`

Cette propriété prend l'URL du service de répertoire auquel le pilote (le pilote LDAP par exemple) doit accéder. L'URL doit être une chaîne de caractères, telle que `"ldap://hôteldap:427"`.

L'exemple ci-dessous montre comment définir les propriétés contextuelles au moment de l'exécution et comment établir une connexion à l'aide de JNDI et LDAP. Ici, la propriété contextuelle `INITIAL_CONTEXT_FACTORY` appelle la mise à oeuvre Sun Microsystem d'un fournisseur de service LDAP. La propriété contextuelle `PROVIDER_URL` a pour valeur l'URL d'un service de répertoire LDAP hébergé sur l'hôte `"serveur1_ldap"` au port 983.

```
Properties props = new Properties();

/* Il s'agit à présent d'utiliser LDAP. A cet effet, INITIAL_CONTEXT_FACTORY
 * a pour valeur le nom de classe d'un CONTEXT_FACTORY LDAP. Dans ce cas,
 * le CONTEXT_FACTORY est fourni par la mise en oeuvre Sun d'un pilote
 * de service de répertoire LDAP.
 */
props.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

/* Attribuer à PROVIDER_URL l'URL du serveur LDAP qui
 * doit fournir les informations de répertoire pour la connexion.
 */
props.put(Context.PROVIDER_URL, "ldap://serveur1_ldap:983");

/* Définir des propriétés contextuelles complémentaires. */
props.put("user", "xyz");
props.put("password", "123");

/* Etablir la connexion */
Connection con = DriverManager.getConnection
    ("jdbc:sybase:jndi:ldap://serveur1_ldap:983" +
    "/servername=Sybasell,o=MaSociété,c=FR", props);
```

Notez que la chaîne de connexion transmise à **`getConnection()`** contient des informations spécifiques de LDAP que le développeur doit impérativement fournir.

Lorsque des propriétés JNDI sont définies au moment de l'exécution, comme dans l'exemple précédent, jConnect les transmet à JNDI afin qu'elles soient utilisées lors de l'initialisation d'un serveur (voir ci-dessous) :

```
javax.naming.directory.DirContext ctx =  
    new javax.naming.directory.InitialDirContext(props);
```

jConnect obtient alors de JNDI les informations de connexion nécessaires en appelant **DirContext.getAtributes()**, comme dans l'exemple suivant, où *ctx* est un objet **DirContext** :

```
javax.naming.directory.Attributes attrs =  
    ctx.getAttributes(ldap://serveur1_ldap:983/servername=  
        Sybase11, SYBASE_SERVER_ATTRIBUTES);
```

Ici, SYBASE_SERVER_ATTRIBUTES est une table de chaînes de caractères définies au sein de jConnect. Cette table rassemble les identificateurs d'objet (OID) correspondant aux données de répertoire énumérées au [tableau 2-3](#).

Mise en oeuvre des plug-ins de sockets personnalisés

Cette section décrit comment connecter un plug-in de socket personnalisé à une application, afin de personnaliser la communication entre un client et un serveur. Vous pouvez, par exemple, personnaliser le socket **javax.net.ssl.SSLSocket** pour activer le cryptage des données.

com.sybase.jdbcx.SybSocketFactory est une interface d'extension Sybase incluant la méthode **createSocket(String, int, Properties)** qui renvoie un **java.net.Socket**. Pour qu'un pilote jConnect 4.1 ou version supérieure charge un socket personnalisé, l'application utilisée doit :

- mettre en oeuvre cette interface,
- définir la méthode **createSocket(..)**.

jConnect utilise le nouveau socket pour les opérations d'entrée et de sortie ultérieures. Les classes qui mettent en oeuvre **SybSocketFactory** génèrent des sockets et autorisent l'ajout de fonctionnalités publiques au niveau des sockets.

```
/**
 * Renvoyer un socket connecté à un ServerSocket vers l'hôte,
 * appelé, sur le port spécifié.
 * @param host  hôte serveur
 * @param port  port serveur
 * @param props  Propriétés transmises via la connexion
 * @returns Socket
 * @exception IOException, UnknownHostException
 */
public java.net.Socket createSocket(String host, int port, Properties props)
throws IOException, UnknownHostException;
```

Le transfert de propriétés permet aux instances de **SybSocketFactory** de mettre en oeuvre un socket intelligent à l'aide de propriétés de connexion.

Lors de la mise en oeuvre de **SybSocketFactory** pour générer un socket, le même code d'application peut utiliser différents sockets en transférant les objets factory ou pseudo-factory composant des sockets vers l'application. Les objets factory sont personnalisables à l'aide des paramètres de construction des sockets. Par exemple, vous pouvez personnaliser des objets factory pour renvoyer des sockets associés à des temporisations réseau différentes ou à des paramètres de sécurité déjà configurés. Ces sockets peuvent être des sous-classes **java.net.Socket** grâce auxquelles il est possible d'utiliser directement dans les nouvelles API des fonctionnalités de compression, de sécurité, de marquage d'enregistrement, de collecte de statistiques, d'encapsulation de serveur protégé par firewall (**javax.net.SocketFactory**), etc.

Remarque **SybSocketFactory** est une classe **javax.net.SocketFactory** simplifiée à l'extrême, qui permet aux applications d'utiliser **java.net.*** ou **javax.net.*** si nécessaire.

Pour utiliser un socket personnalisé avec jConnect :

- 1 Spécifiez une classe Java qui met en oeuvre **com.sybase.jdbcx.SybSocketFactory**. Reportez-vous à la section [“Création et configuration d'un socket personnalisé”](#), page 28.
- 2 Définissez la propriété de connexion SYB SOCKET_FACTORY pour que jConnect puisse obtenir un socket.

Propriété de connexion SYB SOCKET_FACTORY

Pour utiliser un socket personnalisé avec jConnect, il convient d'attribuer l'une des valeurs suivantes à la propriété de connexion SYB SOCKET_FACTORY :

- le nom d'une classe qui met en oeuvre **com.sybase.jdbcx.SybSocketFactory**,

ou

- la valeur DEFAULT, qui instancie un nouveau **java.net.Socket()**

Pour plus d'informations sur la configuration de SYB SOCKET_FACTORY, reportez-vous à la section [“Définition des propriétés de connexion”](#), page 12.

Création et configuration d'un socket personnalisé

jConnect utilise le socket personnalisé pour se connecter à un serveur. Le socket doit être entièrement configuré avant d'être exploitable par jConnect.

La présente section explique comment connecter un socket SSL (tel que **javax.net.ssl.SSLSocket**) avec jConnect.

Remarque Actuellement, aucun serveur Sybase ne supporte le protocole SSL.

L'exemple suivant montre comment, après la mise en oeuvre du protocole SSL, il est possible de créer une instance de socket **SSLSocket**, de la configurer et de la renvoyer. Dans cet exemple, la classe **MySSLSocketFactory** met en oeuvre **SybSocketFactory** et étend **javax.net.ssl.SSLSocketFactory** pour mettre en oeuvre le protocole SSL. Elle contient deux méthodes **createSocket**, respectivement pour **SSLSocketFactory** et pour **SybSocketFactory**, qui :

- créent un socket SSL,
- appellent **SSLSocket.setEnabledCipherSuites()** pour spécifier les séries de chiffres disponibles pour le cryptage,
- renvoient le socket qui sera utilisé par jConnect.

Exemple

```
public class MySSLSocketFactory extends SSLSocketFactory
    implements SybSocketFactory
{
    /**
     * Créer un socket, définir les séries de chiffres utilisables
     * et renvoyer le socket.
     * Démontrer comment les séries de chiffres peuvent être codées en
     * dur dans la mise en oeuvre.
     *
     * See javax.net.SSLSocketFactory#createSocket
     */
    public Socket createSocket(String hôte, int port)
        throws IOException, UnknownHostException
    {
        // Préparer une table contenant les séries de chiffres
        // à activer.
        String enableThese[] =
        {
            "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA",
            "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5",
            "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA"
        }
        ;
        Socket s =
            SSLSocketFactory.getDefault().createSocket(hôte, port);
        ((SSLSocket)s).setEnabledCipherSuites(enableThese);
        return s;
    }
    /**
     * Renvoyer un SSLSocket.
```

```
* Indiquer comment définir des séries de chiffres basées sur des
* propriétés de connexion telles que :
* Properties _props = new Properties();
* Définir d'autres propriétés url, de mot de passe, etc.
* _props.put(("CIPHER_SUITES_1",
*         "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA"));
* _props.put("CIPHER_SUITES_2",
*         "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5");
* _props.put("CIPHER_SUITES_3",
*         "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA");
* _conn = _driver.getConnection(url, _props);
*
* Voir com.sybase.jdbcx.SybSocketFactory#createSocket
*/
public Socket createSocket(String hôte, int port)
    Properties props)
    throws IOException, UnknownHostException
{
    // Vérifier si les séries de chiffres sont définies dans
    // les propriétés de connexion
    Vector cipherSuites = new Vector();
    String cipherSuiteVal = null;
    int cipherIndex = 1;
    do
    {
        if((cipherSuiteVal = props.getProperty("CIPHER_SUITES_"
            + cipherIndex++)) == null)
        {
            if(cipherIndex <= 2)
            {
                // Aucune série de chiffres disponible
                // ne renvoie le SSLSocket par défaut reconnu
                // par l'objet, les séries de chiffres étant
                // activées.
                return createSocket(host, port);
            }
            else
            {
                // Une série de chiffres par requête
                // doit être activée pour la connexion
                break;
            }
        }
        else
        {
            // Ajouter à la série de chiffres Vector, pour
            // activer l'ensemble
        }
    }
}
```

```

        cipherSuites.addElement(cipherSuiteVal);
    }
}
while(true);
// Permet de créer une chaîne String[] externe au vecteur créé
String enableThese[] = new String[cipherSuites.size()];
cipherSuites.copyInto(enableThese);
// Activer les séries de chiffres
Socket s =
    SSLSocketFactory.getDefault().createSocket
        (hôte, port);
((SSLSocket)s).setEnabledCipherSuites(enableThese);
// Renvoyer le SSLSocket
return s;
}
// Autres méthodes
}

```

jConnect ne requiert aucune information sur le type de socket. De ce fait, toute configuration doit être achevée avant de renvoyer un socket.

Pour plus d'informations, consultez les fichiers suivants :

- *Encrypt.java* : situé dans les sous-répertoires *sample* (jConnect 4.x) et *sample2* (jConnect 5.x) de votre répertoire jConnect, cet exemple montre comment utiliser l'interface **SybSocketFactory** avec des applications jConnect.
- *MySSLSocketFactory.java* : également situé dans les sous-répertoires *sample* (jConnect 4.x) et *sample2* (jConnect 5.x) de votre répertoire jConnect, il s'agit d'un exemple de mise en oeuvre de l'interface **SybSocketFactory** que vous pouvez utiliser avec votre application.

Gestion de l'internationalisation et de la localisation

Cette section traite des problèmes d'internationalisation et de localisation relatifs à jConnect.

Convertisseurs de jeu de caractères jConnect

jConnect associe des classes spéciales aux différentes conversions de jeu de caractères. La classe de convertisseur détermine le mode de traitement des conversions de jeu de caractères codés sur un seul ou plusieurs octets, ainsi que l'impact des conversions sur les applications en termes de performances.

Il existe deux classes de conversion de jeu de caractères. La classe que jConnect utilise dépend de la version définie (par exemple, `VERSION_4`) et des propriétés de connexion `CHARSET` et `CHARSET_CONVERTER_CLASS`.

- La classe **TruncationConverter** fonctionne uniquement avec des jeux de caractères codés sur un seul octet qui utilise les caractères ASCII, tels que `iso_1` et `cp850`. Elle n'est pas compatible avec les jeux codés sur plusieurs octets ni avec ceux codés sur un seul octet mais qui utilisent des caractères non ASCII.

En utilisant la classe **TruncationConverter**, jConnect 5.x traite les jeux de caractères de la même manière que jConnect version 2.2. La classe **TruncationConverter** est le convertisseur par défaut utilisé avec la `VERSION_2`.

- **PureConverter** est un convertisseur de jeu de caractères codés sur plusieurs octets. jConnect l'utilise avec la version `VERSION_4` ou supérieure, ainsi qu'avec la `VERSION_2` lorsqu'un jeu associé à la propriété de connexion `CHARSET` n'est pas compatible avec la classe **TruncationConverter**.

Bien qu'elle autorise les conversions de jeu de caractères codés sur plusieurs octets, la classe **PureConverter** peut dégrader les performances du pilote jConnect. Dans ce cas, reportez-vous à la section [“Amélioration des performances lors de la conversion d'un jeu de caractères”](#), page 34.

Sélection d'un convertisseur de jeu de caractères

jConnect utilise le paramètre de version défini par la méthode **SybDriver.setVersion()** pour déterminer la classe de convertisseur de jeu de caractères utilisée par défaut. Pour la VERSION_2, la classe par défaut est **TruncationConverter**. Pour la VERSION_4 et plus, il s'agit de la classe **PureConverter**.

Vous pouvez également définir la propriété de connexion **CHARSET_CONVERTER_CLASS** pour spécifier le convertisseur que jConnect doit utiliser. Cela permet de remplacer le convertisseur par défaut.

Par exemple, si après avoir défini la VERSION_4 ou supérieure de jConnect vous préférez néanmoins utiliser la classe **TruncationConverter** au lieu de la classe **PureConverter**, définissez la propriété **CHARSET_CONVERTER_CLASS** :

Pour jConnect 4.1 :

```
...
props.put ( "CHARSET_CONVERTER_CLASS",
            "com.sybase.utils.TruncationConverter" )
```

Pour jConnect 5.x :

```
...
props.put ( "CHARSET_CONVERTER_CLASS",
            "com.sybase.jdbc2.utils.TruncationConverter" )
```

Définition de la propriété de connexion CHARSET

La propriété de connexion **CHARSET** permet de définir le jeu de caractères à utiliser dans une application. Si vous ne définissez pas cette propriété, jConnect prend les jeux de caractères par défaut suivants :

- Pour la VERSION_2, le jeu iso_1.
- Pour les VERSION_3, VERSION_4 et VERSION_5, le jeu de caractères par défaut de la base de données. jConnect convertit alors automatiquement les caractères nécessaires du côté du client.

Vous pouvez également utiliser l'option de ligne de commande **-J charset** pour que l'application **IsqlApp** définisse un jeu de caractères.

Pour connaître les jeux de caractères installés sur votre Adaptive Server, exécutez la requête SQL suivante :

```
select name from syscharsets
go
```

Pour la classe **PureConverter**, si le jeu de caractères CHARSET défini n'est pas compatible avec la VM (Java Virtual Machine) cliente, la connexion échoue et une **SQLException** est générée ; vous devez alors redéfinir la propriété CHARSET de sorte que le jeu de caractères soit pris en charge à la fois par Adaptive Server et par le client.

Lorsque la classe **TruncationConverter** est utilisée, la troncature des caractères s'applique, et ce quel que soit le jeu de caractères défini (jeu ASCII codé sur 7 bits ou autre) par la propriété CHARSET.

Amélioration des performances lors de la conversion d'un jeu de caractères

Si vous utilisez des jeux de caractères codés sur plusieurs octets et que vous devez améliorer les performances du pilote, vous pouvez utiliser la classe **SunloConverter** fournie avec les exemples de jConnect. Pour plus d'informations, reportez-vous à la section "[Conversion d'un jeu de caractères](#)", [page 116](#).

Jeux de caractères supportés

Le [tableau 2-4](#) répertorie les jeux de caractères Sybase supportés dans cette version de jConnect. Vous y trouverez également le convertisseur d'octet JDK correspondant à chacun des jeux supportés.

UCS-2 est supporté par jConnect mais n'est actuellement pris en charge par aucune base de données Sybase ou Open Server.

A l'inverse du convertisseur de code SJIS de JDK, le jeu de caractères *sjis* de Sybase ne comprend pas les extensions JIS d'IBM ou de Microsoft. Par conséquent, la conversion de chaînes Java en une base de données Sybase à l'aide de *sjis* peut produire des valeurs de caractère non supportées par la base Sybase. En revanche, la conversion inverse à partir de *sjis* (d'une base Sybase en chaînes Java) fonctionne normalement.

Le [tableau 2-4](#) répertorie les jeux de caractères actuellement supportés par Sybase.

Tableau 2-4 : Jeux de caractères Sybase supportés

Jeu Sybase	Convertisseur JDK
ascii_7	8859_1
big5	Big5
cp037	Cp037
cp437	Cp437
cp500	Cp500
cp850	Cp850
cp852	Cp852
cp855	Cp855
cp857	Cp857
cp860	Cp860
cp863	Cp863
cp864	Cp864
cp866	Cp866
cp869	Cp869
cp874	Cp874
cp932	Cp932
cp936	Cp936
cp950	Cp950
cp1250	Cp1250
cp1251	Cp1251
cp1252	Cp1252
cp1253	Cp1253
cp1254	Cp1254
cp1255	Cp1255
cp1256	Cp1256
cp1257	Cp1257
cp1258	Cp1258
deckanji	EUCJIS
eucgb	GB2312
eucjis	EUCJIS
eucksc	Cp949
ibm420	Cp420
ibm918	Cp918

Jeu Sybase	Convertisseur JDK
iso_1	8859_1
iso88592	8859-2
iso88595	8859_5
iso88596	8859_6
iso88597	8859_7
iso88598	8859_8
iso88599	8859_9
iso885915	8859_15
koi8	KOI8_R
mac	Macroman
mac_cyr	MacCyrillic
mac_ee	MacCentralEurope
macgreek	MacGreek
macturk	MacTurkish
sjis (voir la remarque)	SJIS
tis620	MS874
utf8	UTF8

Support du symbole monétaire européen

jConnect version 4.1 ou supérieure supporte le nouveau symbole monétaire européen (l'«euro») et sa conversion depuis et vers l'unicode UCS-2.

L'euro a été ajouté dans les jeux de caractères Sybase suivants : cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258, cp874, iso885915 et utf8.

Les jeux de caractères cp1257, cp1258 et iso885915 sont nouveaux.

Pour utiliser le symbole de l'euro :

- Utilisez la classe **PureConverter**, convertisseur Java de jeu de caractères codés sur plusieurs octets. Pour plus d'informations, reportez-vous à la section [“Convertisseurs de jeu de caractères jConnect”](#), page 32.
- Vérifiez que les nouveaux jeux de caractères sont installés sur le serveur.

Actuellement, le symbole de l'euro n'est supporté que par Adaptive Server Enterprise version 11.9.2 ou version supérieure ; Adaptive Server Anywhere ne prend pas en charge ce symbole.

- Sélectionnez le jeu de caractères approprié sur le client. Pour plus d'informations, reportez-vous à la section [“Définition de la propriété de connexion CHARSET”](#), page 33.
- Mettez JDK au niveau de la version 1.1.7 ou de la plate-forme Java™ 2.

Jeux de caractères non supportés

Les jeux de caractères Sybase suivants ne sont pas supportés par jConnect 5.x. En effet, il n'existe aucun convertisseur d'octet JDK correspondant aux jeux de caractères Sybase.

- cp1047
- euccns
- greek8
- roman8
- turkish8

Ces jeux de caractères peuvent être utilisés avec la classe **TruncationConverter** tant que l'application n'utilise que les sous-ensembles ASCII 7 bits de ces caractères.

Utilisation des bases de données

Cette section décrit les problèmes relatifs aux bases de données susceptibles de se poser avec jConnect :

- Mise en oeuvre de la reprise sur le serveur secondaire en mode haute disponibilité
- Appels de procédure à distance interserveur
- Accès aux métadonnées d'une base de données
- Utilisation de curseurs dans des jeux de résultats
- Support pour les mises à jour par batch
- Mise à jour de la base de données à partir du jeu de résultats d'une procédure stockée
- Utilisation des types de données

Mise en oeuvre de la reprise sur le serveur secondaire en mode haute disponibilité

Les versions 4.2 et 5.2 de jConnect supportent la fonctionnalité de reprise sur le serveur secondaire de Sybase disponible dans Adaptive Server Enterprise version 12.0.

Remarque La reprise sur le serveur secondaire de Sybase en mode haute disponibilité ne doit pas être confondue avec la fonctionnalité de “reprise de connexion”. Si vous souhaitez utiliser ces deux fonctions, nous vous conseillons de lire *très attentivement* la présente section.

Présentation

La reprise sur le serveur secondaire de Sybase vous permet de configurer deux Adaptive Server version 12.0 comme compagnons. Si le compagnon primaire tombe en panne, ses devices, ses bases de données et ses connexions peuvent être pris en charge par le compagnon secondaire.

Vous pouvez configurer un système haute disponibilité selon un schéma symétrique ou asymétrique.

Dans une configuration *asymétrique*, deux Adaptive Server, situés chacun sur une machine différente, sont connectés de telle sorte que si l'un d'eux tombe en panne, l'autre reprend sa charge de travail. L'Adaptive Server secondaire joue le rôle de serveur de "reprise automatique" et reste en veille tant qu'aucune reprise n'est déclenchée.

Une configuration *symétrique* comprend également deux Adaptive Server qui s'exécutent sur des machines séparées. Cependant, si l'un d'eux tombe en panne, déclenchant ainsi un processus de reprise, chacun des deux Adaptive Server peut agir en tant que compagnon primaire ou secondaire pour l'autre. Dans cette configuration, chaque Adaptive Server est totalement fonctionnel et possède ses propres devices système, bases de données système et utilisateur et connexions utilisateur.

Quel que soit le schéma, asymétrique ou symétrique, les deux machines sont configurées pour un double accès, ce qui permet à une machine de voir les disques de l'autre et d'y accéder.

Vous pouvez activer la reprise sur le serveur secondaire de Sybase dans jConnect et connecter une application cliente à un Adaptive Server configuré pour ce mode. Si le serveur primaire transfère sa charge de travail sur le serveur secondaire, l'application cliente bascule elle aussi automatiquement sur ce dernier et rétablit les connexions réseau.

Remarque Pour plus d'informations sur le mode reprise sur le serveur secondaire de Sybase, reportez-vous au document *Utilisation de Sybase Failover en environnement haute disponibilité*.

Éléments requis, références et restrictions

- Vous devez posséder deux Adaptive Server version 12.0 configurés pour le mode reprise sur le serveur secondaire.
- Vous devez utiliser jConnect 4.2 ou jConnect 5.2. En effet, les versions de pilote antérieures ne supportent pas cette fonctionnalité.
- Seules les modifications validées dans la base de données avant la reprise sur le serveur secondaire sont conservées lorsque le client transfère ses activités.
- La connexion de l'application cliente doit être effectuée à l'aide de JNDI. Reportez-vous à la section "[Connexion à un serveur à l'aide de JNDI](#)", [page 21](#).

- La notification d'événement jConnect ne fonctionne pas lorsqu'a lieu une reprise sur le serveur secondaire. Reportez-vous à la section "[Utilisation de notifications d'événement](#)", page 66.
- Fermez toutes les instructions qui ne sont plus utiles. En effet, jConnect stocke des informations sur les instructions pour activer la reprise sur le serveur secondaire. Si vous ne fermez pas les instructions inutiles, des fuites de mémoire risquent de se produire.

Mise en oeuvre de la reprise sur le serveur secondaire dans jConnect

Pour mettre en oeuvre dans jConnect le support du mode reprise sur le serveur secondaire, procédez comme suit :

- 1 Configurez les Adaptive Server primaire et secondaire pour le mode reprise sur le serveur secondaire.
- 2 Ajoutez une entrée pour le serveur primaire et une autre, distincte, pour le serveur secondaire, dans le fichier d'informations du service de répertoire requis par JNDI. L'entrée du serveur primaire aura un attribut (HA OID, ou ID d'objet haute disponibilité) qui fera référence à l'entrée du serveur secondaire.

Lorsque vous utilisez LDAP comme fournisseur de service pour JNDI, cet attribut HD peut prendre trois formes :

- *Relative Distinguished Name (RDN)* : cette forme suppose que la base de la recherche (généralement fournie par l'attribut **java.naming.provider.url**) associée à la valeur de cet attribut suffit pour identifier le serveur secondaire. Par exemple, si le serveur primaire se trouve sur l'hôte 4200 et le serveur secondaire sur l'hôte 4202 :

```
dn: servername=haprimary, o=Sybase, c=FR
1.3.6.1.4.1.897.4.2.5: TCP#1#nom_hôte 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary
objectclass: sybaseServer
```

```
dn: servername=hasecondary, o=Sybase, c=FR
1.3.6.1.4.1.897.4.2.5: TCP#1#nom_hôte 4202
objectclass: sybaseServer
```

- *Distinguished Name (DN)* : cette forme suppose que la valeur de l'attribut HA identifie de manière unique le serveur secondaire, et elle peut ou non reproduire des valeurs trouvées dans la base de la recherche. Par exemple :

```
dn: servername=happrimary, o=Sybase, c=FR
1.3.6.1.4.1.897.4.2.5: TCP#1#nom_hôte 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary,
    o=Sybase, c=FR ou=Comptabilité
objectclass: sybaseServer
```

```
dn: servername=hasecondary, o=Sybase, c=FR, ou=Comptabilité
1.3.6.1.4.1.897.4.2.5: TCP#1#nom_hôte 4202
objectclass: sybaseServer
```

Notez que **hasecondary** est situé dans une branche différente de l'arborescence (voir le qualificatif supplémentaire **ou=Comptabilité**).

- *Full LDAP URL* : dans cette forme, le système ne tire aucune conclusion à partir de la base de la recherche. L'attribut HD doit en principe être un URL de LDAP entièrement qualifié, utilisé pour identifier le serveur secondaire (il peut même pointer sur un serveur LDAP secondaire). Par exemple :

```
dn: servername=hafailover, o=Sybase, c=FR
1.3.6.1.4.1.897.4.2.5: TCP#1#nom_hôte 4200
1.3.6.1.4.1.897.4.2.15: ldap://serveur_ldap:386/servername=
secondary,
    o=Sybase, c=US ou=Accounting
objectclass: sybaseServer

dn: servername=secondary, o=Sybase, c=FR, ou=Comptabilité
1.3.6.1.4.1.897.4.2.5: TCP#1#nom_hôte 4202
objectclass: sybaseServer
```

- 3 Dans le fichier d'informations de service de répertoire dont a besoin JNDI, définissez la propriété de connexion REQUEST_HA_SESSION de telle sorte qu'elle active une session de reprise sur le serveur secondaire chaque fois que vous établissez une connexion.

La nouvelle propriété de connexion REQUEST_HA_SESSION sert à indiquer que l'application cliente qui se connecte souhaite démarrer une session de reprise avec l'Adaptive Server 12.0 configuré pour ce mode. Lorsque cette propriété a la valeur "true", jConnect tente d'établir une connexion de reprise sur le serveur secondaire. Si vous ne définissez pas cette propriété, la session de reprise ne démarre pas, même si le serveur est configuré correctement. La valeur par défaut de REQUEST_HA_SESSION est "false".

Cette propriété de connexion se définit comme toutes les autres. Vous n'avez pas besoin de la réinitialiser une fois que la connexion a été établie.

Pour bénéficier d'une plus grande souplesse dans le contrôle des sessions de reprise sur le serveur secondaire, codez l'application cliente pour permettre la définition de REQUEST_HA_SESSION au moment de l'exécution runtime.

L'exemple ci-dessous montre les informations de connexion entrées pour le serveur de bases de données SYBASE11 dans un service de répertoire LDAP :

```
dn: servername=SYBASE11,o=MaSociété,c=FR
1.3.6.1.4.1.897.4.2.5:TCP#1#tahiti 3456
1.3.6.1.4.1.897.4.2.10:REPEAT_READ=false&PACKETSIZE=1024
1.3.6.1.4.1.897.4.2.10:CONNECTION_FAILOVER=false
1.3.6.1.4.1.897.4.2.11:pubs2
1.3.6.1.4.1.897.4.2.9:Tds
1.3.6.1.4.1.897.4.2.15:servername=SECONDARY
1.3.6.1.4.1.897.4.2.10:REQUEST_HA_SESSION=true

dn:servername=SECONDARY, o=MaSociété, c=FR
1.3.6.1.4.1.897.4.2.5:TCP#1#moorea 6000
```

où "tahiti" est le serveur primaire et "moorea" le serveur compagnon secondaire.

4 Demandez une connexion en utilisant JNDI et LDAP.

- jConnect utilise le service de répertoire du serveur LDAP pour déterminer le nom des serveurs primaire et secondaire :

```
/* Etablir la connexion */
Connection con = DriverManager.getConnection
("jdbc:sybase:jndi:ldap://serveur1_ldap:983" +
"/servername=Sybasell,o=MaSociété,c=FR",props);
```


ou

- Spécifiez une base de recherche :

```
props.put(Context.PROVIDER_URL,
           "ldap://serveur1_ldap:983/ o=MaSociété, c=FR");

Connection con=DriverManager.getConnection
("jdbc:sybase:jndi:servername=Sybase11", props);
```

Connexion au serveur primaire

Si un Adaptive Server n'est pas configuré pour la reprise sur le serveur secondaire ou si, pour une raison quelconque, il ne peut pas déclencher une session de reprise, le client ne peut pas se connecter et le message suivant s'affiche :

```
'La demande de session HD a été rejetée par le serveur.
Veuillez reconfigurer la base de données ou travailler
hors session HD.'
```

Reprise sur le serveur secondaire

En cas de reprise sur le serveur secondaire, une exception est générée et le client se reconnecte automatiquement à la base de données secondaire en utilisant JNDI.

Il convient de noter les aspects suivants :

- L'identité de la base de données à laquelle le client était connectée et les transactions déjà validées sont conservées.
- Une partie des jeux de résultats lus, des curseurs et des appels de procédures stockées sont perdus.
- Lorsque la reprise a lieu, votre application peut avoir besoin de redémarrer une procédure ou de revenir à la dernière transaction ou activité exécutée.

Retour au serveur primaire

A un moment donné, le client va repasser du serveur secondaire au serveur primaire. Le moment de ce retour est déterminé par l'administrateur système qui émet la commande **sp_failback** sur le serveur secondaire. Après quoi, les comportements et résultats du serveur sont les mêmes que ceux décrits à la section [“Reprise sur le serveur secondaire”](#), page 43.

Appels de procédure à distance interserveur

Une commande ou une procédure stockée Transact-SQL exécutée sur un serveur peut mettre en oeuvre une autre procédure stockée située sur un autre serveur. Le serveur auquel une application s'est connectée se connecte à son tour à un serveur distant et exécute un appel de procédure à distance interserveur.

Vous pouvez spécifier pour une application un mot de passe "universel" qui sera utilisé pour toutes les connexions interserveur. Une fois la connexion établie, ce mot de passe permet au serveur de se connecter à un serveur distant.

Par défaut, jConnect utilise le mot de passe de la connexion en cours pour toutes les communications interserveur.

Toutefois, s'il existe des mots de passe différents pour un même utilisateur connecté sur deux serveurs et que l'utilisateur exécute des appels de procédure à distance, les mots de passe utilisés pour chacun des serveurs doivent être définis de manière explicite.

jConnect version 4.1 et supérieure inclut une propriété grâce à laquelle vous pouvez définir un mot de passe "à distance" universel ou différents mots de passe sur des serveurs distincts. Il est également possible de définir et de configurer cette propriété à l'aide de la méthode **setRemotePassword()** de la classe **SybDriver** :

```
Properties connectionProps = new Properties();
```

```
public final void setRemotePassword(String serverName,  
    String password, Properties connectionProps)
```

Pour que cette méthode soit utilisable, la classe **SybDriver** doit être importée avant d'être appelée par l'application.

Pour jConnect 4.x :

```
import com.sybase.jdbcx.SybDriver;  
SybDriver sybDriver = (SybDriver)  
    Class.forName("com.sybase.jdbc.SybDriver").newInstance();  
sybDriver.setRemotePassword  
    (serverName, password, connectionProps);
```

Pour jConnect 5.x :

```
import com.sybase.jdbcx.SybDriver;  
SybDriver sybDriver = (SybDriver)  
    Class.forName("com.sybase.jdbc2.jdbc.SybDriver").newInstance();  
sybDriver.setRemotePassword  
    (serverName, password, connectionProps);
```

Remarque Pour définir des mots de passe différents pour plusieurs serveurs, répétez, pour chacun d'eux, l'appel précédent (en fonction de la version de jConnect utilisée).

Cet appel ajoute la paire nom du serveur-mot de passe donnée à l'objet **Properties** spécifié, qui peut être transmise par l'application à **DriverManager** dans **DriverManager.getConnection** (*url_serveur, props*).

Si **serverName** a la valeur NULL, le mot de passe universel défini est **password** pour les connexions ultérieures à tous les serveurs, excepté ceux qui ont été spécifiquement définis par des appels précédents à l'aide de **setRemotePassword()**.

Lorsque la propriété REMOTEPWD est définie par une application, le mot de passe universel n'est plus défini par jConnect.

Accès aux métadonnées d'une base de données

Pour supporter les méthodes **DatabaseMetaData** de JDBC, Sybase fournit un ensemble de procédures stockées que jConnect peut solliciter pour obtenir des métadonnées relatives à une base de données. Ces procédures stockées doivent être installées sur le serveur.

Si ce n'est pas le cas, utilisez l'un des scripts de procédure stockée fournis avec jConnect :

- *sql_server.sql* installe les procédures stockées dans une base de données Adaptive Server version inférieure à 12.0.
- *sql_server12.sql* installe les procédures stockées dans une base de données Adaptive Server version 12.0.
- *sql_anywhere.sql* installe les procédures stockées dans une base de données Adaptive Server Anywhere.

Remarque La dernière version des scripts est compatible avec toutes les versions de jConnect.

Pour plus d'informations sur l'installation de procédures stockées, reportez-vous aux documents *Guide d'installation et Notes de mise à jour*.

En outre, pour utiliser les méthodes de métadonnées, la propriété de connexion `USE_METADATA` doit avoir la valeur “true” (valeur par défaut) lorsque vous établissez une connexion.

Vous ne pouvez pas obtenir de métadonnées relatives à des tables temporaires d'une base de données.

Remarque La méthode `DatabaseMetaData.getPrimaryKeys()` recherche les clés primaires déclarées dans la définition de table (`CREATE TABLE`) ou lors d'une modification de table (`ALTER TABLE ADD CONSTRAINT`). Mais elle ne fait pas appel à `sp_primarykey`.

Installation de métadonnées sur le serveur

Il est possible de mettre en oeuvre le support de métadonnées sur le client (ODBC, JDBC) ou la source de données (procédures stockées serveur). `jdbcConnect` supporte les métadonnées sur le serveur, ce qui permet de :

- conserver la taille réduite de `jdbcConnect` pour rapidement télécharger le pilote depuis Internet ;
- accélérer le temps d'exécution à partir de procédures stockées préchargées sur la source de données ;
- bénéficier d'une plus grande souplesse, `jdbcConnect` pouvant se connecter à diverses bases de données.

Utilisation de curseurs dans des jeux de résultats

`jdbcConnect 5.x` met en oeuvre des méthodes de curseur et de mise à jour JDBC 2.0. Ces méthodes facilitent l'utilisation de curseurs et la mise à jour de lignes dans une table en fonction des valeurs du jeu de résultats.

Remarque Pour bénéficier d'une prise en charge complète de JDBC 2.0, utilisez `jdbcConnect` version 5.x ou supérieure. `jdbcConnect` version 4.x propose des fonctionnalités JDBC 2.0 via des extensions Sybase et l'exemple `ScrollableResultSet.java` situé dans le sous-répertoire *sample* de votre répertoire `jdbcConnect`. Pour plus d'informations javadoc sur ces méthodes, consultez `com.sybase.jdbcx` et les packages

Sous JDBC 2.0, les jeux de résultats **ResultSet** sont caractérisés par leur type et leur synchronisation. Ces paramètres sont intégrés à l'interface **java.sql.ResultSet** et sont décrits dans la documentation javadoc correspondante.

Le [tableau 2-5](#) identifie les caractéristiques de **java.sql.ResultSet** disponibles sous jConnect 5.x.

Tableau 2-5 : Options java.sql.ResultSet disponibles sous jConnect 5.x

Synchronisation	Type		
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
CONCUR_READ_ONLY	Supporté par la version 5.x	Supporté par la version 5.x	Non disponible dans la version 5.x
CONCUR_UPDATABLE	Supporté par la version 5.x	Non disponible dans la version 5.x	Non disponible dans la version 5.x

Cette section traite des sujets suivants :

- [Création d'un curseur](#)
- [Mises à jour et suppressions par position à l'aide des méthodes JDBC 1.x](#)
- [Utilisation d'un curseur associé à une instruction préparée](#)
- [Support pour jeux de résultats SCROLL_INSENSITIVE dans jConnect](#)

Création d'un curseur

Pour créer un curseur à l'aide de jConnect 4.x, vous devez utiliser **SybStatement.setCursorName()** ou **SybStatement.setFetchSize()**. Lorsque vous utilisez **SybStatement.setCursorName()**, vous attribuez explicitement un nom au curseur. La signature de **SybStatement.setCursorName()** est la suivante :

```
void setCursorName(String name) throws SQLException;
```

La méthode **SybStatement.setFetchSize()** sert à créer un curseur et à spécifier le nombre de lignes renvoyées par la base de données à chaque extraction. La signature de **SybStatement.setFetchSize()** est la suivante :

```
void setFetchSize(int rows) throws SQLException;
```

Lorsque vous utilisez **setFetchSize()** pour créer un curseur, le pilote jConnect nomme ce dernier. Pour obtenir le nom du curseur, utilisez **ResultSet.setCursorName()**.

La création de curseur à l'aide de `jdbcConnect` version 5.x s'effectue comme sous la version 4.x. Toutefois, la version 5.x supportant JDBC 2.0, il existe une autre méthode pour les créer. Pour indiquer le type de **ResultSet** renvoyé par l'instruction, utilisez la méthode JDBC 2.0 suivante sur la connexion :

```
Statement createStatement(int resultSetType, int
resultSetConcurrency)throws SQLException
```

Le type et la synchronisation correspondent aux types et synchronisations de l'interface **ResultSet**, répertoriés dans le [tableau 2-5](#). Si l'interface **ResultSet** demandée n'est pas supportée, un avertissement SQL est enchaîné à la connexion. Lorsque l'instruction **Statement** est exécutée, le type de **ResultSet** se rapprochant le plus de celui demandé est renvoyé. Pour plus d'informations sur l'action de cette méthode, reportez-vous aux spécifications de JDBC 2.0.

Si vous n'utilisez pas `createStatement()` ou si vous utilisez `jdbcConnect` version 4.x, les types par défaut de **ResultSet** sont :

- Si vous n'appellez que **Statement.executeQuery()**, le **ResultSet** renvoyé est un résultat **SybResultSet** de type `TYPE_FORWARD_ONLY` et `CONCUR_READ_ONLY`.
- Si vous appelez **setFetchSize()** ou **setCursorName()**, le résultat **ResultSet** renvoyé par **executeQuery()** est un résultat **SybCursorResultSet** de type `TYPE_FORWARD_ONLY` et `CONCUR_UPDATABLE`.

Pour vérifier que le type d'objet **ResultSet** correspond bien à votre demande, l'API JDBC 2.0 pour **ResultSet** a ajouté les deux méthodes :

```
int getConcurrency() throws SQLException;
int getType() throws SQLException;
```

Les procédures élémentaires de création et d'utilisation d'un curseur sont les suivantes :

- 1 Créez le curseur à l'aide de **Statement.setCursorName()** ou de **SybStatement.setFetchSize()**.
- 2 Appelez **Statement.executeQuery()** pour ouvrir le curseur associé à une instruction et renvoyer un jeu de résultats curseur.
- 3 Appelez **ResultSet.next()** pour extraire des lignes et positionner le curseur dans le jeu de résultats.

L'exemple suivant illustre les deux méthodes de création de curseurs et de renvoi d'un jeu de résultats. Il utilise également **ResultSet.setCursorName()** pour obtenir le nom du curseur créé par **SybStatement.setFetchSize()**.

```
// conn étant un objet de connexion, créer
// Statement object et lui attribuer un curseur
// à l'aide de Statement.setCursorName().
Statement stmt = conn.createStatement();
stmt.setCursorName("author_cursor");

// Utiliser l'instruction pour exécuter une
// requête et renvoyer un jeu de résultats curseur.
ResultSet rs = stmt.executeQuery("SELECT au_id,
    au_lname, au_fname FROM authors
    WHERE city = 'Oakland'");
while(rs.next())
{
    ...
}

// Créer un second objet Statement et utiliser
// SybStatement.setFetchSize() pour créer un
// curseur qui renvoie 10 lignes à la fois.
SybStatement syb_stmt = conn.createStatement();
syb_stmt.setFetchSize(10);

// Utiliser syb_stmt pour exécuter une requête et
// renvoyer un jeu de résultats curseur.
SybCursorResultSet rs2 =
    (SybCursorResultSet)syb_stmt.executeQuery
    ("SELECT au_id, au_lname, au_fname FROM authors
    WHERE city = 'Pinole'");
while(rs2.next())
{
    ...
}

// Obtenir le nom du curseur créé par la méthode
// setFetchSize().
String cursor_name = rs2.getCursorName();
...
// Créer pour jConnect 5.x un troisième objet
// Statement à l'aide de la nouvelle méthode sur,
// Connection et obtenir un jeu de résultats,
// SCROLL_INSENSITIVE.
// Remarque : Il n'est plus utile de déclasser les
// instructions Statement ou ResultSet.
Statement stmt = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
```

```

                                ResultSet.CONCUR_READ_ONLY);
ResultSet rs3 = stmt.executeQuery
    ("SELECT ... [whatever]");
//Exécuter l'une des méthodes JDBC 2.0
// appropriée pour les jeux de résultats READ_ONLY.
rs3.next();
rs3.previous();
rs3.relative(3);
rs3.afterLast();
...

```

Mises à jour et suppressions par position à l'aide des méthodes JDBC 1.x

L'exemple suivant montre comment réaliser une mise à jour par position à l'aide de méthodes JDBC 1.x. Il s'agit dans un premier temps de créer deux objets **Statement**, l'un pour la sélection de lignes dans un jeu de résultats curseur, l'autre pour la mise à jour de la base de données conformément à ces lignes.

```

// Créer deux objets Statement et associer un
// curseur au jeu de résultats renvoyé par la
// première instruction, stmt1. Utiliser stmt1
// pour exécuter une requête et renvoyer un
// jeu de résultats curseur.
Statement stmt1 = conn.createStatement();
Statement stmt2 = conn.createStatement();
stmt1.setCursorName("author_cursor");
ResultSet rs = stmt1.executeQuery("SELECT
    au_id, au_lname, au_fname
    FROM authors WHERE city = 'Oakland'
    FOR UPDATE OF au_lname");

// Obtenir le nom du curseur créé pour stmt1 afin
// de l'utiliser avec stmt2.
String cursor = rs.getCursorName();

// Utiliser stmt2 pour mettre à jour la base
// d'après le jeu de résultats renvoyé par stmt1.
String last_name = new String("Smith");
while(rs.next())
{
    if (rs.getString(1).equals("274-80-9391"))
    {
        stmt2.executeUpdate("UPDATE authors " +
            "SET au_lname = "+last_name +
            "WHERE CURRENT OF " + cursor);
    }
}

```


Suppressions dans un jeu de résultats

L'exemple suivant utilise l'objet **Statement** *stmt2*, du code précédent, pour effectuer une suppression par position :

```
stmt2.executeUpdate("DELETE FROM authors  
WHERE CURRENT OF " + cursor);
```

Mises à jour et suppressions par position à l'aide des méthodes JDBC 1.x

Cette section décrit les méthodes JDBC 2.0 qui permettent de mettre à jour des colonnes et une base de données à partir d'un jeu de résultats. Elles sont suivies d'un exemple.

Mise à jour de colonnes dans un jeu de résultats

JDBC 2.0 fournit un certain nombre de méthodes permettant la mise à jour de valeurs de colonne à partir d'un jeu de résultats, sur le client. Les valeurs actualisées peuvent ensuite servir à la mise à jour, l'insertion ou la suppression de données dans la base sous-jacente. L'ensemble de ces méthodes est mis en oeuvre dans la classe **SybCursorResultSet**.

Les méthodes de mise à jour ci-dessous sont des exemples de méthodes JDBC 2.0 disponibles dans *jConnect* :

```
void updateAsciiStream(String columnName, java.io.InputStream x,  
    int length) throws SQLException;  
void updateBoolean(int columnIndex, boolean x) throws  
    SQLException;  
void updateFloat(int columnIndex, float x) throws SQLException;  
void updateInt(String columnName, int x) throws SQLException;  
void updateInt(int columnIndex, int x) throws SQLException;  
void updateObject(String columnName, Object x) throws  
    SQLException;
```

Méthodes de mise à jour de la base de données à partir d'un jeu de résultats

JDBC 2.0 comprend deux nouvelles méthodes pour la mise à jour et la suppression de lignes dans la base de données, qui s'appuient sur les valeurs courantes d'un jeu de résultats. Ces méthodes sont plus simples d'utilisation que la méthode **Statement.executeUpdate()** dans JDBC 1.x et ne nécessitent pas de nom de curseur. Elles sont mises en oeuvre dans **SybCursorResultSet** :

```
void updateRow() throws SQLException;
void deleteRow() throws SQLException;
```

Remarque La synchronisation du jeu de résultats doit être de type **CONCUR_UPDATABLE**, sinon une exception sera renvoyée par les méthodes mentionnées ci-avant. Toutes les colonnes de tables qui requièrent des entrées de type **NON NULL** doivent être spécifiées pour **insertRow()**.

Les méthodes fournies par **DatabaseMetaData** déterminent quand ces modifications sont visibles.

Exemple

L'exemple ci-après crée un objet **Statement** unique qui sert à renvoyer un jeu de résultats curseur. Pour chaque ligne du jeu de résultats, les valeurs de colonne sont mises à jour en mémoire, puis transmises à la base de données.

```
// Créer un objet Statement et définir la taille d'extraction à
// 25. Un curseur est associé à l'objet Statement
// Utiliser l'instruction pour renvoyer un jeu de résultats
// curseur.
SybStatement syb_stmt =
(SybStatement)conn.createStatement();
syb_stmt.setFetchSize(25);
SybCursorResultSet syb_rs =
(SybCursorResultSet)syb_stmt.executeQuery(
    "SELECT * from T1 WHERE ...")

// Mettre à jour chaque ligne du jeu de résultats
// selon le code dans la boucle while suivante.
// jConnect extrait 25 lignes à la fois, le cas échéant.
// La dernière extraction porte sur toutes
// les lignes restantes.
while(syb_rs.next())
{
    // Mettre à jour les colonnes 2 et 3 de chaque
    // ligne, où colonne 2 est une variable (varchar) dans la
    // base de données et colonne 3 est un nombre entier (int).
    syb_rs.updateString(2, "xyz");
    syb_rs.updateInt(3,100);
}
```

```
//Mettre à jour la ligne dans la base de données.
syb_rs.updateRow();
}
// Créer un objet Statement en utilisant la méthode
// JDBC 2.0 mise en oeuvre dans jConnect 5.x
Statement stmt = conn.createStatement
(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
// Utiliser l'objet Statement pour renvoyer un jeu de résultats
// actualisable
ResultSet rs = stmt.executeQuery("SELECT * FROM T1 WHERE...");
// Il n'est pas nécessaire avec jConnect 5.x de déclasser
// SybCursorResultSet. Pour mettre à jour chaque ligne de ResultSet,
// procédez comme indiqué ci-dessus
while (rs.next())
{
rs.updateString(2, "xyz");
rs.updateInt(3,100);
rs.updateRow();
}
```

Suppression d'une ligne d'un jeu de résultats

Pour supprimer une ligne d'un jeu de résultats, utilisez **SybCursorResultSet.deleteRow()** comme suit :

```
while(syb_rs.next())
{
    int col3 = getInt(3);
    if (col3 >100)
    {
        syb_rs.deleteRow();
    }
}
```

Insertion d'une ligne dans un jeu de résultats

L'exemple ci-après montre comment insérer des lignes en utilisant l'API JDBC 2.0, disponible uniquement avec jConnect 5.x. Il n'est pas nécessaire de déclasser un objet vers **SybCursorResultSet**.

```
// Préparer à insérer
rs.moveToInsertRow();
// Spécifier des valeurs de colonne dans
// la nouvelle ligne.
rs.updateString(1, "New entry for col 1");
rs.updateInt(2, 42);
```

```
// Insérer une nouvelle ligne dans db.
rs.insertRow();
// Retourner à la ligne courante du jeu de résultats.
rs.moveToCurrentRow();
```

Utilisation d'un curseur associé à une instruction préparée

Une fois créé, un objet **PreparedStatement** peut être utilisé plusieurs fois avec ou non les mêmes valeurs de paramètre d'entrée. Si vous associez un curseur à un objet **PreparedStatement**, vous devez fermer le curseur après chaque utilisation, puis le rouvrir pour le réutiliser. Un curseur est désactivé lorsque vous fermez son jeu de résultats (**ResultSet.close()**). Il est activé lorsque vous exécutez son instruction préparée (**PreparedStatement.executeQuery()**).

L'exemple ci-après montre comment créer un objet **PreparedStatement**, lui attribuer un curseur et exécuter l'objet **PreparedStatement** à deux reprises, en fermant puis rouvrant le curseur.

```
// Créer un objet d'instruction préparée combiné à
// une requête paramétrée.
PreparedStatement prep_stmt =
conn.prepareStatement(
"SELECT au_id, au_lname, au_fname "+
"FROM authors WHERE city = ? "+
"FOR UPDATE OF au_lname");

// Associer un curseur à l'instruction.
prep_stmt.setCursorName("author_cursor");

// Attribuer une valeur au paramètre de requête.
// Exécuter l'instruction préparée pour renvoyer
// un jeu de résultats.
prep_stmt.setString(1, "Oakland");
ResultSet rs = prep_stmt.executeQuery();

// Traiter le jeu de résultats.
while(rs.next())
{
    ...
}

// Fermer le jeu de résultats, ce qui désactive
// le curseur.
rs.close();

// Exécuter à nouveau l'instruction préparée en
```

```
// changeant la valeur de paramètre.  
prep_stmt.setString(1, "San Francisco");  
rs = prep_stmt.executeQuery();  
// Rouvrir le curseur
```

Support pour jeux de résultats **SCROLL_INSENSITIVE** dans **jConnect**

jConnect version 5.x ne supporte que les jeux de résultats **TYPE_SCROLL_INSENSITIVE**.

jConnect utilise le protocole TDS (Tabular Data Stream) de Sybase pour communiquer avec les serveur de données Sybase. A partir de **jConnect** 5.x, TDS ne supporte pas les curseurs avec défilement. Pour les prendre en charge, **jConnect** met en mémoire cache les données de ligne sur demande, sur le client et sur chaque appel **ResultSet.next()**. Toutefois, lorsque la fin du jeu de résultats est atteinte, tout le jeu de résultats est enregistré dans la mémoire du client. En raison des contraintes liées aux performances, il est recommandé de n'utiliser les jeux de résultats **TYPE_SCROLL_INSENSITIVE** que lorsque le jeu de résultats est de taille réduite.

Remarque Si vous utilisez **ResultSets** **TYPE_SCROLL_INSENSITIVE** avec **jConnect** 5.x, n'appellez la méthode **isLast()** qu'une fois la dernière ligne de **ResultSet** lue, sans quoi une exception **UnimplementedOperationException** sera générée.

Un exemple a été ajouté à **jConnect** 4.x et fournit un **ResultSet** **TYPE_SCROLL_INSENSITIVE** limité avec les interfaces **JDBC 1.0**.

Cette mise en oeuvre utilise des méthodes **JDBC 1.0** standard pour générer un jeu de résultats non modifiable, ne prenant pas en compte les distinctions de curseur avec ou sans défilement. Cette vue statique des données sous-jacentes ne prend en compte les modifications apportées qu'une fois le jeu de résultats fermé. **ExtendedResultSet** met en mémoire cache toutes les lignes de **ResultSet** sur le client. Attention toutefois lorsque vous utilisez cette classe avec des jeux de résultats volumineux.

L'interface **sample.ScrollableResultSet** :

- est une extension de **JDBC 1.0 java.sql.ResultSet** de **JDBC 1.0** ;
- définit des méthodes supplémentaires qui comportent les mêmes signatures que **JDBC 2.0 java.sql.ResultSet** de **JDBC 2.0** ;
- *ne* contient *pas* toutes les méthodes **JDBC 2.0**. Les méthodes manquantes s'appliquent aux modifications de **ResultSet**.

Les méthodes de l'API JDBC 2.0 API définies sont :

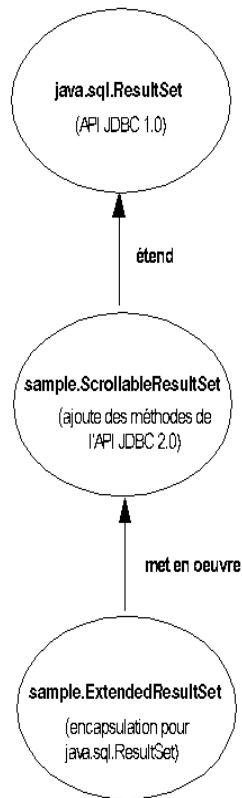
```
boolean previous() throws SQLException;
boolean absolute(int row) throws SQLException;
boolean relative(int rows) throws SQLException;
boolean first() throws SQLException;
boolean last() throws SQLException;
void beforeFirst() throws SQLException;
void afterLast() throws SQLException;
boolean isFirst() throws SQLException;
boolean isLast() throws SQLException;
boolean isBeforeFirst() throws SQLException;
boolean isAfterLast() throws SQLException;
int getFetchSize() throws SQLException;
void setFetchSize(int rows) throws SQLException;
int getFetchDirection() throws SQLException;
void setFetchDirection(int direction) throws SQLException;
int getType() throws SQLException;
int getConcurrency() throws SQLException;
int getRow() throws SQLException;
```

Pour utiliser de nouvelles classes d'exemples, créez **ExtendedResultSet** en utilisant un **java.sql.ResultSet** JDBC 1.0. Les sections de code importantes (dans un environnement Java 1.1) sont les suivantes :

```
// Importer les fichiers exemples.
import sample.*;
// Importer les classes JDBC 1.0.
import java.sql.*;
// Connecter à des db à l'aide d'un pilote ;
// créer une instruction et une requête ;
// Obtenir une référence pour un ResultSet JDBC 1.0.
ResultSet rs = stmt.executeQuery(_query);
// Créer un objet ScrollableResultSet.
ScrollableResultSet srs = new ExtendedResultSet(rs);
// Invoquer des méthodes depuis l'API JDBC 2.0.
srs.beforeFirst();
// ou invoquer des méthodes depuis l'API JDBC 1.0.
if (srs.next())
    String column1 = srs.getString(1);
```

La [figure 2-1](#) schématise les relations entre les nouvelles classes d'exemples et l'API JDBC.

Figure 2-1 : Schéma de classes



Pour plus d'informations sur l'API JDBC 2.0, consultez le site :
<http://java.sun.com/products/jdbc/jdbcse2.html>.

Support pour les mises à jour par batch

Les mises à jour par batch permettent à un objet **Statement** de soumettre à une base de données sous-jacente plusieurs commandes de mise à jour comme une seule unité (ou batch) pour un traitement regroupé.

Remarque Pour utiliser des mises à jour par batch, vous devez rafraîchir les scripts SQL dans le sous-répertoire *sp* du répertoire d'installation jConnect.

Pour obtenir un exemple d'utilisation de mises à jour par batch à l'aide des objets **Statement**, **PreparedStatement** et **CallableStatement**, reportez-vous au fichier *BatchUpdates.java*, situé dans les sous-répertoires *sample* (jConnect 4.x) et *sample2* (jConnect 5.x).

jConnect supporte également dans les batch les objets dynamiques **PreparedStatement**.

Remarques

jConnect met en oeuvre des mises à jour par batch comme spécifié dans l'API JDBC 2.0, sauf restrictions mentionnées ci-dessous.

- Si la norme JDBC 2.0 pour la mise en oeuvre de **BatchUpdateException.getUpdateCounts()** est modifiée ou n'est pas encore disponible, la norme d'origine est mise en oeuvre par jConnect. **BatchUpdateException.getUpdateCounts()** renvoie alors une longueur **int[]** de $M < N$, ce qui signifie que les premières instructions M du batch ont abouti, que l'instruction $M+1$ a échoué et que les instructions $M+2..N$ n'ont pas été exécutées (N correspondant au nombre total d'instructions dans le batch).
- Mises à jour par batch de procédures stockées
Pour appeler des procédures stockées en mode batch (mode non chaîné), vous devez créer la procédure stockée en mode non chaîné. Pour plus d'informations, reportez-vous à la section "[Procédure stockée exécutée en mode non chaîné](#)", page 109.
- Adaptive Server Enterprise 11.5.x ou version supérieure
BatchUpdateException.getUpdateCounts() ne renvoie qu'une longueur **int[]** nulle. En cas d'erreur, toute la transaction est annulée, ce qui génère des lignes de zéros.
- Adaptive Server Enterprise version 11.0.1
Renvoie des lignes de 0 (zéros) affectées aux procédures stockées.
- SQL Anywhere version 5.5.x
 - SQL Anywhere version 5.5.x ne vous permet pas d'obtenir des insertions de décomptes de lignes à partir des procédures stockées qui contiennent des insertions. Par exemple :

```
create proc sp_A as insert tableA values (1,  
'hello A')  
create proc sp_B
```



```
as
insert tableA values (1, 'hello A')
update tableA set coll=2
create proc sp_C
as
update tableA set coll=2
delete tableA
```

L'exécution de **executeBatch** sur la procédure stockée précédente donne respectivement les résultats suivants :

```
0 Rows Affected
1 Rows Affected
2 Rows Affected
```

- Aucun support n'existe pour les objets dynamiques **PreparedStatement** de batch.
- Because SQL Anywhere 5.5.x ne supportant pas les mises à jour par batch conformément à la spécification JDBC 2.0, ces dernières sont effectuées dans une boucle **executeUpdate**.
- Mises à jour par batch dans des bases de données qui ne supportent pas ce type de mise à jour
jConnect effectue les mises à jour par batch dans une boucle **executeUpdate**, même si votre base de données ne supporte pas les mises à jour par batch. Vous pouvez ainsi utiliser le même code par batch, quelle que soit la base de données sélectionnée.

Pour plus d'informations sur les mises à jour par batch, reportez-vous au document *Sun Microsystems, Inc. JDBC™ 2.0 API*.

Mise à jour de la base de données à partir du jeu de résultats d'une procédure stockée

jConnect fournit des méthodes de mise à jour et de suppression qui permettent de placer un curseur dans le jeu de résultats renvoyé par une procédure stockée. Vous pouvez ensuite utiliser la position du curseur pour mettre à jour ou supprimer des lignes dans la table sous-jacente à l'origine du jeu de résultats. Ces méthodes sont intégrées dans **SybCursorResultSet**:

```
void updateRow(String tableName) throws SQLException;
```

```
void deleteRow(String tableName) throws SQLException;
```

Le paramètre *tableName* identifie la table de base de données à l'origine du jeu de résultats.

Pour placer un curseur dans le jeu de résultats renvoyé par une procédure stockée, utilisez soit **SybCallableStatement.setCursorName()** or **SybCallableStatement.setFetchSize()** avant d'exécuter l'instruction qui contient la procédure. L'exemple ci-après montre comment créer un curseur dans le jeu de résultats d'une procédure stockée, mettre à jour les valeurs dans ce jeu de résultats, puis répercuter ces modifications dans la table sous-jacente à l'aide de la méthode **SybCursorResultSet.update()** :

```
// Créer un objet CallableStatement pour l'exécution de
// la procédure stockée.
CallableStatement sproc_stmt =
    conn.prepareCall("{call update_titles}");

// Définir le nombre de lignes à renvoyer depuis la base de données
// données pour chaque extraction. Un curseur est associé
// au jeu de résultats.
(SybCallableStatement)sproc_stmt.setFetchSize(10);

//Exécuter la procédure stockée et générer un jeu de résultats.
SybCursorResultSet sproc_result = (SybCursorResultSet)
    sproc_stmt.executeQuery();

// Parcourir le jeu de résultats, ligne par ligne, mettre à jour
// la ligne courante du curseur et mettre à jour la table des
// titres sous-jacente avec les valeurs modifiées de la ligne.
while(sproc_result.next())
{
    sproc_result.updateString(...);
    sproc_result.updateInt(...);
    ...
    sproc_result.updateRow(titles);
}
```

Utilisation des types de données

Envoi de données *Image*

jConnect comprend une classe **TextPointer** contenant des méthodes **sendData()** qui permettent de mettre à jour une colonne *image* dans une base de données Adaptive Server Enterprise ou Adaptive Server Anywhere. Dans les versions antérieures de jConnect, les données image devaient être envoyées par la méthode **setBinaryStream()** de **java.sql.PreparedStatement**. Les méthodes **TextPointer.sendData()** utilisent **java.io.InputStream** et améliorent très nettement les performances lors de la transmission des données image vers une base Adaptive Server.

Pour obtenir des instances de la classe **TextPointer**, vous pouvez utiliser l'une des deux méthodes **getTextPtr()** de **SybResultSet** :

```
public TextPointer getTextPtr(String columnName)
public TextPointer getTextPtr(int columnIndex)
```

Méthodes publiques de la classe **TextPointer**

Le package **com.sybase.jdbc** inclut la classe **TextPointer**. Son interface de méthode publique est la suivante :

```
public void sendData(InputStream is, boolean log)
    throws SQLException
public void sendData(InputStream is, int length,
    boolean log) throws SQLException
public void sendData(InputStream is, int offset,
    int length, boolean log) throws SQLException
public void sendData(byte[] byteInput, int offset,
    int length, boolean log) throws SQLException
```

sendData(InputStream is, boolean log) : met à jour une colonne image conformément aux données contenues dans le flux d'entrée spécifié.

sendData(InputStream is, int length, boolean log) : met à jour une colonne image conformément aux données contenues dans le flux d'entrée spécifié. *length* correspond au nombre d'octets transmis.

sendData(InputStream is, int offset, int length, boolean log) : met à jour une colonne image conformément aux données contenues dans le flux d'entrée spécifié. La mise à jour commence à la position d'octets indiquée par le paramètre *offset* et couvre le nombre d'octets défini par le paramètre *length*.

sendData(byte[] byteInput, int offset, int length, boolean log) : met à jour une colonne image conformément aux données contenues dans la table d'octets spécifiée par le paramètre *byteInput*. La mise à jour commence à la position d'octet spécifiée par le paramètre *offset* et couvre le nombre d'octets défini par le paramètre *length*.

A chaque méthode correspond un paramètre *log*. Le paramètre *log* indique si les données image doivent être intégralement consignées dans le journal de transactions de la base de données. Si le paramètre *log* a la valeur "true", l'image binaire est entièrement consignée dans le journal de transaction. Si la valeur est "false", la mise à jour est consignée mais pas l'image.

❖ **Mise à jour d'une colonne *image* à l'aide de *TextPointer.sendData()***

Pour intégrer des données image dans une colonne, procédez comme suit :

- 1 Créez un objet **TextPointer** pour la ligne et la colonne à mettre à jour.
- 2 Utilisez **TextPointer.sendData()** pour exécuter la mise à jour.

Les deux sections suivantes illustrent la procédure de mise à jour à l'aide d'un exemple. Dans celui-ci, les données image du fichier *Anne_Ringer.gif* sont envoyées pour mettre à jour la colonne *pic* de la table *au_pix* dans la base de données *pubs2*. La mise à jour concerne la ligne qui présente l'ID d'auteur 899-46-2035.

Création d'un objet
TextPointer

En plus des données text et image, les colonnes correspondantes contiennent des informations distinctes (estampille et emplacement de page). Lorsque vous sélectionnez des données dans une colonne text ou image, ces informations complémentaires sont "cachées" dans le jeu de résultats.

Un objet **TextPointer** destiné à la mise à jour d'une colonne image requiert ces informations cachées, mais pas la partie image des données de la colonne. Pour obtenir ces informations, vous devez sélectionner la colonne dans un objet **ResultSet** puis utiliser la méthode **SybResultSet.getTextPtr()** (voir l'exemple plus bas). **SybResultSet.getTextPtr()** extrait les informations de pointeur de texte, ignore les données image et crée un objet **TextPointer**.

Lorsqu'une colonne contient une quantité importante de données image, il est inutile de sélectionner la colonne d'une ou de plusieurs lignes et d'attendre de récupérer toutes les données. Vous pouvez par conséquent ignorer ce dernier en utilisant la commande **set textsize** pour réduire au minimum la quantité de données renvoyées par paquet. Dans l'exemple de code suivant destiné à créer un objet **TextPointer**, la commande **set textsize** est utilisée à cette fin.

```
/*
 * Définir une chaîne permettant de rechercher dans la colonne
 * pic l'ID d'auteur 899-46-2035.
 */
String getColumnData = "select pic from au_pix where au_id = '899-46-2035'";

/*
 * Utiliser la commande set textsize pour renvoyer uniquement un
 * octet des données de colonne dans un objet Statement. Le
 * paquet contenant les données de colonne comprendra les
 * informations "cachées" nécessaires à la création de l'objet.
 * TextPointer.
 */
Statement stmt= connection.createStatement();
stmt.executeUpdate("set textsize 1");

/*
 * Sélectionner les données de colonne dans un objet ResultSet-
 * -convertir ResultSet en SybResultSet car la méthode getTextPtr
 * se trouve dans SybResultSet, qui est une extension de ResultSet.
 */
SybResultSet rs = (SybResultSet)stmt.executeQuery(getColumnData);

/*
 * Placer le curseur de jeu de résultats sur les données de
 * colonne renvoyées et créer l'objet TextPointer.
 */
rs.next();
TextPointer tp = rs.getTextPtr("pic");

/*
 * En supposant que la mise à jour ne porte que sur une seule
 * ligne, et qu'il n'est pas nécessaire de réduire la valeur du
 * paramètre textsize au minimum pour le prochain renvoi de
 * données du serveur, la valeur par défaut de textsize est
 * rétablie.
 */
stmt.executeUpdate("set textsize 0");
```

Exécution de la mise
à jour à l'aide de
TextPointer.sendData

Le code suivant reprend l'objet **TextPointer** de la section précédente pour intégrer les données image du fichier *Anne_Ringer.gif* dans la colonne *pic*.

```
/*
 * Commencer par définir un flux de données correspondant au fichier.
 */
FileInputStream in = new FileInputStream("Anne_Ringer.gif");

/*
 * Préparer l'envoi du flux de données sans consigner les données
 * image dans le journal de transactions.
 */
boolean log = false;

/*
 * Envoyer les données image de Anne_Ringer.gif pour mettre à jour
 * la colonne pic pour l'ID d'auteur 899-46-2035.
 */
tp.sendData(in, log);
```

Pour plus d'informations, consultez les exemples *TextPointers.java* situés dans les sous-répertoires *sample* (jConnect 4.x) et *sample2* (jConnect 5.x) de votre répertoire d'installation jConnect.

Utilisation des types de données *Date* et *Time*

JDBC utilise trois types de données temporelles : *Time*, *Date* et *Timestamp*. Cependant, Adaptive Server n'en utilise qu'un seul, *datetime*, qui est l'équivalent du type *Timestamp* de JDBC. Le type de données *datetime* d'Adaptive Server supporte une précision qui peut atteindre 1/300e de seconde.

Les trois types de données JDBC sont traités en tant que type *datetime* du côté du serveur. Un type *Timestamp* de JDBC s'apparente fondamentalement au type *datetime* d'un serveur ; par conséquent, aucune conversion ne s'impose. Toutefois, le passage du type de données JDBC *Time* ou *Date* au type *datetime* d'un serveur, ou inversement, requiert une conversion.

- Pour convertir le type *Time* en *datetime*, la date du 1er jan 1970 est ajoutée.
- Pour convertir le type *Date* en *datetime*, "00:00:00" est ajouté.
- Pour convertir un type *datetime* en une variable *Date* ou *Time*, les informations inutiles sont supprimées.

Remarques

- Veillez à bien distinguer le type de données *Timestamp* de JDBC du type *timestamp* d'Adaptive Server. En effet, ce dernier est une valeur binaire variable (*varbinary*) unique qui s'utilise lorsque les mises à jour s'appuient sur une stratégie de "synchronisation optimisée".
- Lorsqu'une valeur est insérée en tant que type de données *Time*, la partie liée à la date n'a plus de raison d'être. Par conséquent, il est préférable d'extraire la valeur à l'aide d'un type *Time*, mais jamais *Date* ou *Timestamp*.
- Si vous utilisez **getObject()** avec une colonne Adaptive Server Anywhere *date* ou *time*, la valeur est renvoyée en tant que type de données *Timestamp* JDBC.

Types de données *Char/Varchar/Text* et *getBytes()*

N'utilisez pas **rs.getBytes()** sur un champ de type *char*, *varchar* ou *text* si les données ne sont pas hexadécimales, octales ou décimales.

Mise en oeuvre de fonctionnalités avancées

Cette section décrit comment utiliser les fonctionnalités avancées de jConnect. Elle aborde les sujets suivants :

- [Utilisation de notifications d'événement](#)
- [Gestion des messages d'erreur](#)
- [Stockage d'objets Java en tant que données de colonne dans une table](#)
- [Chargement de classes dynamique](#)
- [Support des extensions JDBC 2.0 Optional Package](#)

Utilisation de notifications d'événement

La fonctionnalité de notification d'événement de jConnect permet d'informer une application de l'exécution d'une procédure Open Server.

Pour utiliser cette fonctionnalité, vous devez solliciter la classe **SybConnection**, qui hérite de l'interface **Connection**. **SybConnection** contient les méthodes **regWatch()** et **regNoWatch()** qui permettent respectivement d'activer et de désactiver la notification d'événement.

Votre application doit également mettre en oeuvre l'interface **SybEventHandler**. Celle-ci contient une méthode publique, **void event(String proc_name, ResultSet params)**, qui est appelée lorsque l'événement spécifié se produit. Les paramètres de l'événement sont transmis à la méthode **event()**, qui indique à l'application comment répondre.

Pour utiliser la notification d'événement dans votre application, appelez la méthode **SybConnection.regWatch()** afin d'enregistrer l'application dans la liste de notification d'une procédure enregistrée. Utilisez la syntaxe ci-dessous :

```
SybConnection.regWatch(nom_proc,eventHdlr,option)
```

- *nom_proc* est une chaîne correspondant au nom de la procédure enregistrée qui génère la notification.
- *eventHdlr* est une instance de la classe **SybEventHandler** que vous mettez en oeuvre.

- *option* est soit NOTIFY_ONCE, soit NOTIFY_ALWAYS. Utilisez NOTIFY_ONCE pour que l'application soit uniquement informée de la première exécution d'une procédure, ou Use NOTIFY_ALWAYS pour que l'application soit informée de chaque exécution d'une procédure.

Lorsqu'un événement associé au *nom_proc* se produit sur l'Open Server, jConnect appelle **eventHdlr.event()** à partir d'un thread distinct. Les paramètres d'événement sont transmis à **eventHdlr.event()** lorsque celui-ci est exécuté. Le thread étant distinct, la notification d'événement ne bloque pas l'exécution de l'application.

Si *nom_proc* n'est pas une procédure enregistrée, ou si Open Server ne parvient pas à ajouter le client dans la liste de notification, l'appel de **regWatch()** génère une exception SQL.

Pour désactiver la notification d'événement, utilisez l'appel suivant :

```
SybConnection.regNoWatch(nom_proc)
```

Remarque Lorsque vous utilisez des extensions de notification d'événement, l'application doit appeler la méthode **close()** de la connexion pour supprimer un thread fils créé par le premier appel **regWatch()**. Sinon, la Machine Virtuelle risque de suspendre les opérations lorsque vous quittez l'application.

Exemple de notification d'événement

L'exemple suivant montre comment mettre en oeuvre un gestionnaire d'événement, puis enregistrer un événement dans une instance du gestionnaire, lorsqu'une connexion est établie :

```
public class MyEventHandler implements SybEventHandler
{
    // Déclarer les champs et les constructeurs, au besoin.
    ...
    public MyEventHandler(String eventname)
    {
        ...
    }

    // Mettre en oeuvre SybEventHandler.event.
    public void event(String eventName, ResultSet params)
    {
        try
        {
            // Rechercher les messages d'erreur reçus avant
            // la notification d'événement.
```

```
        SQLWarning sqlw = params.getWarnings();
        if sqlw != null
        {
            // Traiter toute erreur.
            ...
        }
        // Traiter les paramètres comme tout jeu de résultats
        // associé à une ligne.
        ResultSetMetaData rsmd = params.getMetaData();
        int numColumns = rsmd.getColumnCount();
        while (params.next())           // facultatif
        {
            for (int i = 1; i <= numColumns; i++)
            {
                System.out.println(rsmd.getColumnName(i) + " = " +
                                   params.getString(i));
            }
            // Prendre les mesures nécessaires par rapport à
            // l'événement. Par exemple, notifier le thread
            // d'application.
            ...
        }
    }
    catch (SQLException sqe)
    {
        // Traiter toute erreur
        ...
    }
}

public class MyProgram
{
    ...
    // Obtenir une connexion et enregistrer un événement dans
    // une instance de MyEventHandler.
    Connection conn = DriverManager.getConnection(...);
    MyEventHandler myHdlr = new MyEventHandler("MY_EVENT");

    // Enregistrer votre gestionnaire d'événement.
    ((SybConnection)conn).regWatch("MY_EVENT", myHdlr,
        SybEventHandler.NOTIFY_ALWAYS);
    ...
    conn.regNoWatch("MY_EVENT");
    conn.close();
}
```

Gestion des messages d'erreur

jConnect fournit deux classes permettant de renvoyer des informations relatives aux erreurs spécifiques de Sybase, **SybSQLException** et **SybSQLWarning**, ainsi qu'une interface **SybMessageHandler** qui vous permet de personnaliser la façon dont jConnect gère les messages d'erreur provenant du serveur.

Extraction d'informations relatives aux erreurs spécifiques de Sybase

jConnect fournit une interface **EedInfo** qui fournit des méthodes d'obtention d'informations relatives aux erreurs spécifiques de Sybase. Cette interface **EedInfo** est intégrée dans **SybSQLException** et **SybSQLWarning**, qui héritent des classes **SQLException** et **SQLWarning**.

SybSQLException et **SybSQLWarning** contiennent les méthodes suivantes :

- **public ResultSet getEedParams();**
Renvoie un jeu de résultats d'une ligne contenant toute valeur de paramètre présente dans le message d'erreur.
- **public int getStatus();**
Renvoie "1" s'il existe des valeurs de paramètre dans le message ou "0" dans le cas contraire.
- **public int getLineNumber();**
Renvoie le numéro de ligne de la procédure stockée ou de la requête à l'origine du message d'erreur.
- **public String getProcedureName();**
Renvoie le nom de la procédure à l'origine du message d'erreur.
- **public String getServerName();**
Renvoie le nom du serveur qui a généré le message.
- **public int getSeverity();**
Renvoie le degré de sévérité du message d'erreur.
- **public int getState();**
Renvoie des informations relatives à la source interne du message d'erreur dans le serveur. L'utilisation de cette méthode est exclusivement réservée au Support Technique de Sybase.
- **public int getTranState();**

Renvoie l'un des états de transaction suivants :

- 0 La connexion se trouve actuellement dans une transaction étendue.
- 1 La validation de la transaction précédente a abouti.
- 3 La transaction précédente a été abandonnée.

Notez que certains messages d'erreur peuvent être de type **SQLException** ou **SQLWarning**, sans pour autant être de type **SybSQLException** ou **SybSQLWarning**. Votre application doit vérifier le type d'exception traité avant d'appeler une méthode **SybSQLException** ou **SybSQLWarning**.

Personnalisation de la gestion des messages d'erreur

L'interface **SybMessageHandler** permet de personnaliser la façon dont jConnect traite les messages d'erreur générés par le serveur. La mise en oeuvre de **SybMessageHandler** dans votre propre classe pour le traitement des messages d'erreur présente les avantages suivants :

- Gestion des erreurs "universelle"

La logique du traitement des erreurs peut être intégrée dans la routine de gestion d'erreurs, plutôt que répétée dans toute une application.

- Consignation des erreurs "universelle"

La routine de gestion d'erreurs peut contenir la logique du traitement de la consignation des erreurs dans sa globalité.

- Reconfiguration du degré de sévérité des messages d'erreur, en fonction des exigences d'une application.

Votre routine de gestion d'erreurs peut contenir la logique permettant de reconnaître des messages d'erreur spécifiques et de modifier leur degré de sévérité en fonction des exigences d'une application plutôt qu'en fonction de l'échelle de sévérité définie par le serveur. Par exemple, le degré de sévérité d'un message signalant l'inexistence d'une ligne peut baisser à la suite d'une opération de nettoyage où les lignes obsolètes sont supprimées. Il peut également être augmenté dans d'autres circonstances.

Remarque Les routines de gestion d'erreurs qui mettent en oeuvre l'interface **SybMessageHandler** reçoivent uniquement les messages générés par les serveurs. Elles ne traitent pas les messages émis par jConnect.

Lorsque `jConnect` reçoit un message d'erreur, il vérifie si une classe **SybMessageHandler** a été enregistrée pour le traitement du message. Dans l'affirmative, `jConnect` appelle la méthode **messageHandler()**. Cette dernière accepte une exception `SQL` en tant qu'argument et `jConnect` traite le message en fonction de la valeur renvoyée par la méthode **messageHandler()**. La routine de gestion d'erreurs peut :

- Renvoyer l'exception `SQL` telle quelle.
- Renvoyer une valeur `NULL`. Dans ce cas, `jConnect` ignore le message.
- Créer un avertissement `SQL` à partir d'une exception `SQL`, puis le renvoyer. L'avertissement est alors ajouté dans la chaîne des messages d'avertissement.
- Si le message d'origine est un avertissement `SQL`, **messageHandler()** peut estimer que l'avertissement est urgent et créer, puis renvoyer une exception `SQL` à transmettre lorsque `jConnect` reprend la main.

Installation d'une routine de gestion d'erreurs

Vous pouvez installer une routine de gestion d'erreurs pour la mise en oeuvre de **SybMessageHandler** en appelant la méthode **setMessageHandler()** depuis **SybDriver**, **SybConnection** ou **SybStatement**. Si vous installez une routine de gestion d'erreurs à partir de **SybDriver**, tous les objets **SybConnection** suivants en héritent. Si vous installez une routine de gestion d'erreurs à partir d'un objet **SybConnection**, tous les objets **SybStatement** créés par ce **SybConnection** en héritent.

Cette hiérarchie s'applique uniquement à partir du moment où la routine de gestion d'erreurs est installée. Par exemple, si vous créez un objet **SybConnection**, *maConnexion* avant d'appeler **SybDriver.setMessageHandler()** pour installer l'objet de routine de gestion d'erreurs, *maConnexion*, ne peut pas utiliser ce dernier.

Pour renvoyer l'objet de routine de gestion d'erreurs courant, utilisez la méthode **getMessageHandler()**.

Exemple de routine de gestion d'erreurs

Dans l'exemple suivant, jConnect version 4.1 est utilisé.

```
import java.io.*;
import java.sql.*;
import com.sybase.jdbcx.SybMessageHandler;
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybStatement;
import java.util.*;

public class MyApp
{
    static SybConnection conn = null;
    static SybStatement stmt = null;
    static ResultSet rs = null;
    static String user = "guest";
    static String password = "sybase";
    static String server = "jdbc:sybase:Tds:192.138.151.39:4444";
    static final int AVOID_SQLE = 20001;

    public MyApp()
    {
        try
        {
            Class.forName("com.sybase.jdbc.SybDriver").newInstance();
            Properties props = new Properties();
            props.put("user", utilisateur);
            props.put("password", mot_de_passe);
            conn = (SybConnection)
                DriverManager.getConnection(server, props);
            conn.setMessageHandler(new NoResultSetHandler());
            stmt =(SybStatement) conn.createStatement();
            stmt.executeUpdate("raiserror 20001 'votre erreur'");

            for (SQLWarning sqw = _stmt.getWarnings();
                sqw != null;
                sqw = sqw.getNextWarning());
            {
                if (sqw.getErrorCode() == AVOID_SQLE);
                {
                    System.out.println("Erreur" +sqw.getErrorCode()+
                        " trouvée dans la liste des avertissements de ;
                        " l'instruction.");
                    break;
                }
            }
        }
        stmt.close();
    }
}
```

```
        conn.close();
    }
    catch(Exception e)
    {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

class NoResultSetHandler implements SybMessageHandler
{

    public SQLException messageHandler(SQLException sqe)
    {
        int code = sqe.getErrorCode();
        if (code == AVOID_SQLE)
        {
            System.out.println("Utilisateur " + _user + " modification de " +
                AVOID_SQLE + " en avertissement");
            sqe = new SQLWarning(sqe.getMessage(),
                sqe.getSQLState(),sqe.getErrorCode());
        }
        return sqe;
    }
}

public static void main(String args[])
{
    new MyApp();
}
```

Stockage d'objets Java en tant que données de colonne dans une table

Certains produits de base de données permettent de stocker des objets Java en tant que données de colonne dans une base de données. Les classes Java sont alors traitées en tant que types de données et vous pouvez déclarer le type de données d'une colonne en tant que classe Java.

jConnect supporte le stockage des objets Java dans une base de données grâce à la mise en oeuvre des méthodes **setObject()** définies dans l'interface **PreparedStatement** et des méthodes **getObject()** définies dans les interfaces **CallableStatement** et **ResultSet**. Cela permet d'utiliser jConnect avec une application utilisant des classes et des méthodes en JDBC natif pour stocker et extraire directement des objets Java en tant que données de colonne.

Remarque our utiliser les méthodes **getObject()** et **setObject()**, vous devez configurer la `VERSION_4` ou supérieure de jConnect. Reportez-vous à la section [“Configuration de la version de jConnect”, page 6](#)

Les sections suivantes décrivent les conditions requises et les procédures nécessaires pour le stockage d'objets dans une table, ainsi que pour leur extraction à l'aide de JDBC combiné à jConnect :

- [Conditions préalables au stockage d'objets Java en tant que données de colonne](#)
- [Envoi d'objets Java à une base de données](#)
- [Réception d'objets Java à partir de la base de données](#)

Remarque Dans Adaptive Server Enterprise version 12.0 et Adaptive Server Anywhere version 6.0.x, il est possible de stocker des objets dans une table, mais avec quelques restrictions. Pour plus d'informations, reportez-vous au document *jConnect for JDBC - Notes de mise à jour*

Conditions préalables au stockage d'objets Java en tant que données de colonne

Pour stocker des objets Java appartenant à une classe Java définie par l'utilisateur dans une colonne, les trois conditions suivantes doivent être remplies :

- La classe doit mettre en oeuvre l'interface **java.io.Serializable**. En effet, jConnect a recours à la sérialisation et la désérialisation en Java natif pour échanger des objets avec une base de données.
- La définition de classe doit être installée dans la base de données cible *ou*, en vous servant de **DynamicClassLoader** (DCL), vous devez charger une classe directement à partir d'un serveur Adaptive Server Anywhere ou Adaptive Server Enterprise et l'utiliser comme si elle figurait dans la CLASSPATH locale. Pour plus d'informations, reportez-vous à la section "[Chargement de classes dynamique](#)", page 79.
- Sur le système client, la définition de classe doit se trouver dans un fichier *.class*, accessible à partir de la variable d'environnement CLASSPATH locale.

Envoi d'objets Java à une base de données

Pour envoyer une instance d'une classe définie par l'utilisateur en tant que données de colonne, utilisez l'une des méthodes **setObject()**, selon le schéma défini dans l'interface **PreparedStatement** :

```
void setObject(int parameterIndex, Object x, int targetSqlType,
    int scale) throws SQLException;
void setObject(int parameterIndex, Object x, int targetSqlType)
    throws SQLException;
void setObject(int parameterIndex, Object x) throws SQLException;
```

L'exemple suivant définit une classe **Address**, montre la définition d'une table *Friends* qui comporte une colonne *Address* ayant pour type de données la classe **Address**, puis insère une ligne dans la table.

```
public class Address implements Serializable
{
    public String streetNumber;
    public String street;
    public String apartmentNumber;
    public String city;
    public int zipCode;
    //Méthodes
    ...
}
```

```
}

/* Ce code crée une table dont la structure est la suivante :
** Create table Friends:
** (firstname varchar(30),
** lastname varchar(30),
** address Address,
** phone varchar(15))
*/

// Etablir la connexion à la base de données contenant la table Friends.
Connection conn =
    DriverManager.getConnection("jdbc:sybase:Tds:localhost:5000",
        "nom_utilisateur", "mot_de_passe");

// Créer un objet Prepared Statement contenant une instruction insert
// pour la mise à jour de la table Friends.
PreparedStatement ps = conn.prepareStatement("INSERT INTO
    Friends values (?, ?, ?, ?)");

// Définir les valeurs dans l'objet d'instruction préparée, ps.
// Définir le prénom en tant que "Jeanne."
ps.setString(1, "Jeanne");

// Définir le nom en tant que "Laporte."
ps.setString(2, "Laporte");

// En supposant que "adresse_Jeanne" est une instance
// d'Address, utiliser setObject(int parameterIndex, Object x) pour
// indiquer "adresse_Jeanne" dans la colonne address.
ps.setObject(3, Joan_address);

// Définir le numéro de téléphone de Jeanne dans la colonne phone.
ps.setString(4, "01-68-52-98-33");

// Exécuter l'insertion.
ps.executeUpdate();
```

Réception d'objets Java à partir de la base de données

Une application JDBC cliente peut recevoir un objet Java à partir de la base de données dans un jeu de résultats ou en tant que valeur d'un paramètre de sortie renvoyé par une procédure stockée.

- Si un jeu de résultats contient un objet Java en tant que données de colonne, utilisez l'une des méthodes **getObject()** suivantes de l'interface **ResultSet** pour récupérer l'objet :

```
Object getObject(int columnIndex) throws SQLException;
Object getObject(String columnName) throws SQLException;
```

- Si un paramètre de sortie issu d'une procédure stockée contient un objet Java, utilisez l'une des méthodes **getObject()** suivantes de l'interface **CallableStatement** pour récupérer l'objet :

```
Object getObject(int parameterIndex) throws SQLException;
```

L'exemple suivant illustre l'utilisation de

ResultSet.getObject(int parameterIndex) pour attribuer à une variable de classe un objet reçu dans un jeu de résultats. L'exemple reprend la classe **Address** et la table *Friends* de la section précédente et présente une application simple qui imprime un nom et une adresse sur une enveloppe.

```
/*
** Cette application prend un prénom et un nom, recherche
** l'adresse d'une personne précise dans la table Friends
** de la base de données et inscrit sur une enveloppe
** le nom et l'adresse récupérés.
*/
public class Envelope
{
    Connection conn = null;
    String firstName = null;
    String lastName = null;
    String street = null;
    String city = null;
    String zip = null;

    public static void main(String[] args)
    {
        if (args.length < 2)
        {
            System.out.println("Syntaxe : Envelope <prénom>
                               <nom>");
            System.exit(1);
        }
        // Créer une enveloppe de 4" x 10"
```

```
Envelope e = new Envelope(4, 10);
try
{
    // Etablir la connexion à la base de données Friends.
    // contenant la table Friends.
    conn = DriverManager.getConnection(
        "jdbc:sybase:Tds:localhost:5000", "nom_utilisateur",
        "mot_de_passe");
    // Rechercher l'adresse de la personne spécifiée.
    firstName = args[0];
    lastName = args[1];
    PreparedStatement ps = conn.prepareStatement(
        "SELECT address FROM friends WHERE " +
        "firstname = ? AND lastname = ?");
    ps.setString(1, firstName);
    ps.setString(2, lastName);
    ResultSet rs = ps.executeQuery();
    if (rs.next())
    {
        Address a = (Address) rs.getObject(1);
        // Définir l'adresse de destination sur l'enveloppe
        e.setAddress(firstName, lastName, a);
    }
    conn.close();
}
catch (SQLException sqe)
{
    sqe.printStackTrace();
    System.exit(2);
}
// Si l'opération a abouti, imprimer l'enveloppe
e.print();
}

private void setAddress(String fname, String lname, Address a)
{
    street = a.streetNumber + " " + a.street + " " +
        a.apartmentNumber;
    city = a.city;
    zip = "" + a.zipCode;
}

private void print()
{
    // Imprimer le nom et l'adresse sur l'enveloppe.
    ...
}
}
```

Les sous-répertoires *sample* (jConnect 4.x) et *sample2* (jConnect 5.x) de votre répertoire jConnect contiennent des exemples plus détaillés de **HandleObject.java**.

Chargement de classes dynamique

Adaptive Server Anywhere version 6.0 et Adaptive Server Enterprise version 12.0 proposent des classes Java en SQL (JCS), ce qui permet de spécifier celles-ci en tant que :

- Types de données de colonnes SQL
- Types de données de variables Transact-SQL
- Valeurs par défaut pour des colonnes SQL

Précédemment, seules les classes qui apparaissaient dans la variable CLASSPATH de jConnec étaient accessibles. Par conséquent, si une application jConnect tentait d'accéder à une instance d'une classe absente de la variable globale CLASSPATH, il en résultait une exception **java.lang.ClassNotFound**.

jConnect version 5.2 met en oeuvre la méthode **DynamicClassLoader** (DCL) pour charger directement une classe depuis un serveur Adaptive Server Anywhere ou Adaptive Server Enterprise et l'utiliser comme si elle figurait dans la variable d'environnement locale CLASSPATH.

Toutes les fonctions de sécurité présentes dans la superclasse sont héritées. Le modèle de délégation du chargeur mis en oeuvre dans Java 2 est appliqué : jConnect tente d'abord de charger une classe demandée depuis CLASSPATH ; en cas d'échec, il utilise **DynamicClassLoader**.

Pour plus d'informations sur JCS et Adaptive Server, reportez-vous au document *Adaptive Server Enterprise Version 12.0 Feature Overview*.

Utilisation de **DynamicClassLoader**

Pour utiliser la fonctionnalité DCL, procédez comme suit :

- 1 Créer et configurer un chargeur de classe. Le code de votre application jConnect doit ressembler à celui-ci :

```
Properties props = new Properties();  
// URL du serveur où résident les classes.  
String classesUrl = "jdbc:sybase:Tds:myase:1200";
```

```
// Propriétés de connexion au serveur ci-dessus.
props.put("user", "grinch");
props.put("password", "meanone");
...

// Demander à SybDriver un nouveau chargeur de classe.
DynamicClassLoader loader = driver.getClassLoader(classesUrl, props);
```

- 2 Utilisez la propriété de connexion `CLASS_LOADER` pour mettre le nouveau chargeur de classe à la disposition de l'instruction qui exécute la requête. Une fois que vous avez créé le chargeur de classe, transférez-lui les connexions suivantes, comme indiqué (il s'agit de la suite de l'exemple de code précédent).

```
// Stocker le chargeur de classe afin que d'autres connexions
// puissent en prendre connaissance.
props.put("CLASS_LOADER", loader);

// Propriétés de connexion supplémentaires
props.put("user", "joeuser");
props.put("password", "joespassword");

// URL du serveur auquel se connecter.
String url = "jdbc:sybase:Tds:jdbc.sybase.com:4446";

// Etablir une connexion.
Connection conn = DriverManager.getConnection(url, props);
```

En supposant la définition de classe Java suivante :

```
class Addr {
    String street;
    String city;
    String state;
}
```

et la définition de table SQL suivante :

```
create table employee (char(100) name, int empid, Addr address)
```

- 3 En l'absence d'une classe **Addr** dans la variable `CLASSPATH` de l'application cliente, utilisez le code côté client décrit ci-après :

```
Statement stmtnt = conn.createStatement();
// Récupérer des lignes dans la table qui contient une classe Java
// comme l'un de ses champs.
ResultSet rs = stmtnt.executeQuery(
    "select * from employee where empid = '19'");
if (rs.next() {
    // Même si la classe ne figure pas dans notre chemin,
    // il faut pouvoir accéder à son instance.
    Object obj = rs.getObject("address");
```

```

// La classe a été chargée à partir du serveur.
// Examiner cette classe.
Class c = obj.getClass();
// Java Reflection possible
// pour accéder aux champs de l'objet.
...
}

```

La propriété de connexion `CLASS_LOADER` permet le partage d'un chargeur de classe par plusieurs connexions.

Toutefois, vous devez vérifier que ce partage n'entraîne pas des conflits de classe. Par exemple, si deux instances différentes et incompatibles de la classe **org.foo.Bar** existent dans deux bases de données distinctes, des problèmes risquent de surgir si le même chargeur sert à accéder aux deux classes. La première classe est chargée lors de l'examen du jeu de résultats renvoyé par la première connexion. Lorsqu'il s'agit ensuite d'examiner un jeu de résultats de la seconde connexion, il apparaît que la classe est déjà chargée. En fait, la seconde classe n'est jamais chargée et JConnect ne dispose d'aucun moyen pour détecter cette situation.

Toutefois, Java comporte un mécanisme intégré qui garantit que les données de version d'une classe correspondent à celle d'un objet désérialisé. De la sorte, Java peut au moins détecter la situation ci-dessus et la signaler.

Il n'est pas nécessaire que les classes et leurs instances résident sur la même base de données ou sur le même serveur, mais il n'y a pas de raison que le chargeur et les connexions suivantes ne puissent pas se référer à la même base ou au même serveur.

Désérialisation

L'exemple suivant montre comment désérialiser un objet d'un fichier local. L'objet sérialisé est une instance d'une classe qui réside sur un serveur et qui n'existe pas dans `CLASSPATH`.

SybResultSet.getObject() utilise **DynamicObjectInputStream**, qui est une sous-classe de **ObjectInputStream** et qui charge une définition de classe à partir de **DynamicClassLoader**, à la place du chargeur de classe système par défaut ("boot").

```

// Créer un flux sur le fichier contenant
//l'objet sérialisé.
FileInputStream fileStream = new FileInputStream("serFile");
// Créer un "désérialiseur" sur celui-ci. Notez que, mis à part
//ce paramètre supplémentaire, tout le reste est identique

```

```
//as ObjectInputStreamDynamicObjectInputStream
stream = new DynamicObjectInputStream(fileStream, loader);
// Lorsque l'objet est désérialisé, sa classe
//est récupérée sur le serveur par le chargeur.
Object obj = stream.readObject();stream.close();
```

Préchargement de fichiers JAR

jConnect version 5.2 inclut une nouvelle propriété de connexion appelée **PRELOAD_JARS**. Définis sous forme de listes délimitées par une virgule de noms de fichiers JAR, les fichiers JAR sont chargés dans leur totalité. Dans ce contexte, “JAR” fait référence au “JARname enregistré” utilisé par le serveur. Ce nom de fichier JAR est spécifié dans le programme Java d’installation, par exemple :

```
install java new jar 'myJarName' from file '/tmp/mystuff.jar'
```

Si vous définissez **PRELOAD_JARS**, les fichiers JAR sont associés au chargeur de classes et il n’est donc pas nécessaire de les précharger à chaque connexion. Il suffit de spécifier **PRELOAD_JARS** pour une connexion. D’autres tentatives pour précharger les mêmes fichiers JAR risquent de nuire aux performances puisque les données JAR sont inutilement récupérées sur le serveur.

Remarque Adaptive Server Anywhere 6.x et les versions supérieures ne peuvent pas renvoyer un fichier JAR en tant qu’entité. jConnect récupère donc chaque classe. Toutefois, Adaptive Server 12.x et versions supérieures récupèrent la totalité du fichier JAR et chaque classe qu’il contient.

Fonctionnalités avancées

DynamicClassLoader proposent plusieurs méthodes publiques. Pour plus de détails, reportez-vous aux informations javadoc sur le site :

JDBC_HOME/docs/en/javadocs

Des fonctionnalités supplémentaires permettent de garder “active” une connexion à la base de données d’un chargeur lorsqu’une série de chargements de classes doit se dérouler et de charger explicitement une classe par son nom.

Les méthodes publiques héritées de **java.lang.ClassLoader** sont également utilisables. Dans **java.lang.Class**, il existe aussi des méthodes qui effectuent le chargement des classes. Toutefois, vous devez les utiliser avec précaution car certaines choisissent elles-même le chargeur de classes à utiliser. En particulier, vous devez utiliser la version à 3 arguments de **Class.forName()**, sinon le chargeur de classe système (“boot”) est utilisée. Pour plus d’informations, reportez-vous à la section [“Gestion des messages d’erreur”](#), page 69.

Support des extensions JDBC 2.0 Optional Package

Le *JDBC 2.0 Optional Package* (anciennement *JDBC 2.0 Standard Extension API*) définit plusieurs fonctionnalités nouvelles que les pilotes JDBC 2.0 peuvent mettre en oeuvre. jConnect version 5.2 supporte les extensions suivantes :

- [JNDI for Naming Databases](#)
(fonctionne avec tous les SGBD Sybase supportés par jConnect)
- [Connection Pooling](#)
(fonctionne avec tous les SGBD Sybase supportés par jConnect)
- [Support de Gestion des transactions distribuées](#)
(fonctionne uniquement avec Adaptive Server Enterprise version 12.0, ou version 11.x avec XA-Server™)

Les fonctionnalités ci-dessus requièrent des classes et/ou des interfaces qui ne se trouvent pas dans les produits Java 2 standard. Vous devez télécharger **javax.sql.*** et **javax.naming.*** pour mettre en oeuvre "JNDI for Naming Databases" et "Connection Pooling", ainsi que **javax.transaction.xa.*** pour le support DTM (Distributed Transaction Management).

Remarque Sybase recommande d'utiliser JNDI 1.2, qui est compatible avec Java 1.1.6 et versions supérieures.

JNDI for Naming Databases

Référence

Document *JDBC 2.0 Optional Package* (anciennement *JDBC 2.0 Standard Extension API*), chapitre 5, “JNDI and the JDBC API.”

Interfaces associées

- **javax.sql.DataSource**
- **javax.naming.Referenceable**
- **javax.naming.spi.ObjectFactory**

Cette fonctionnalité fournit aux clients JDBC une alternative à la méthode standard permettant d’obtenir des connexions à des bases de données. Au lieu d’appeler **Class.forName** (“**com.sybase.jdbc2.jdbc.SybDriver**”), puis de transmettre un URL JDBC à la méthode **getConnection()** du **DriverManager**, les clients accèdent à un serveur de noms JNDI en utilisant un nom logique pour récupérer un objet **javax.sql.DataSource**. Le rôle de cet objet est de charger le pilote et d’établir la connexion à la base de données physique qu’il représente. Le code client est plus simple et réutilisable car les informations spécifiques du fournisseur ont été incorporées dans l’objet **DataSource**.

La mise en oeuvre Sybase de l’objet **DataSource** est **com.sybase.jdbcx.SybDataSource** (pour plus de détails, consultez la documentation javadoc correspondante). Cette mise en oeuvre supporte les propriétés standard ci-dessous, en suivant le modèle de conception des composants JavaBean :

- **databaseName**
- **dataSourceName**
- **description**
- **networkProtocol**
- **password**
- **portNumber**
- **serverName**
- **user**

roleName n’est pas supportée.

jConnect fournit une mise en oeuvre de l'interface **javax.naming.spi.ObjectFactory** afin qu'il soit possible de créer l'objet **DataSource** à partir des attributs d'une entrée de serveur de noms. Avec **javax.naming.Reference** ou **javax.naming.Name** associé à **javax.naming.DirContext**, cet objet factory peut construire des objets **com.sybase.jdbcx.SybDataSource**. Pour l'utiliser, définissez la propriété système **java.naming.object.factory** de façon à y insérer **com.sybase.jdbc2.SybObjectFactory**.

Utilisation

Vous pouvez utiliser **DataSource** de plusieurs façons, dans différentes applications. Toutes les options sont présentées ci-après, avec des exemples de code destinés à vous aider. Pour plus d'informations, reportez-vous au document *JDBC 2.0 Optional Package* (anciennement *JDBC 2.0 Standard Extension API*) et à la documentation JNDI sur le site Web de Sun.

1a. Configuration par l'administrateur : LDAP

jConnect supporte la connectivité LDAP depuis la version 4.0. Par conséquent, la démarche recommandée, qui n'exige aucun logiciel personnalisé, consiste à configurer des **DataSources** comme des entrées LDAP au format LDIF. Par exemple :

```
dn:servername:myASE, o=MaSociété, c=FR
1.3.6.1.4.1.897.4.2.5:TCP#1# mamachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=moi&password=secret
1.3.6.1.4.1.897.4.2.11:bdutilisateur
```

1b. Accès par le client

Il s'agit là de l'application cliente JDBC standard. La seule différence est que vous accédez au serveur de noms pour obtenir une référence à un objet **DataSource** au lieu d'accéder au **DriverManager** et de transmettre un URL JDBC. Une fois que vous avez obtenu la connexion, le code client est identique à tous les autres codes clients JDBC. Ce code est générique et ne fait référence à Sybase que dans la définition de la propriété de l'objet factory, qu'il est possible de définir comme élément d'environnement.

L'installation jConnect contient le programme exemple *sample2/SimpleDataSource.java* qui illustre l'emploi de **DataSource**. Cet exemple n'est proposé que pour référence ; vous ne pouvez pas l'exécuter, à moins de configurer votre environnement et de modifier l'exemple en conséquence. *SimpleDataSource.java* contient le code essentiel suivant :

```
import javax.naming.*;
import javax.sql.*;
import java.sql.*;

// Définir les propriétés JNDI nécessaires pour votre environnement
// (comme ci-dessus)
Properties jndiProps = new Properties();

// Utilisées par JNDI pour créer l'objet SybDataSource
jndiProps.put(Context.OBJECT_FACTORIES,
    "com.sybase.jdbc2.jdbc.SybObjectFactory");

// serveur de noms auquel JNDI doit s'adresser
jndiProps.put(Context.PROVIDER_URL,
    "ldap://serveur_ldap:238/o=MaSociété,c=FR");

// Utilisé par JNDI pour établir le contexte de nommage
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

// Obtenir une connexion à votre serveur de noms
Context ctx = new InitialContext(jndiProps);
DataSource ds = (DataSource) ctx.lookup("servername=myASE");

// Obtenir une connexion au serveur tel qu'il a été configuré plus haut.
// Dans ce cas, le nom utilisateur et le mot de passe par défaut
// seront utilisés
Connection conn = ds.getConnection();

// Créer des méthodes JDBC standard
...
```

Notez que le passage explicite de **Properties** au constructeur **InitialContext** n'est pas obligatoire si les propriétés sont déjà définies dans la machine virtuelle (c'est-à-dire si elles ont été transmises au moment où Java a été appelé), comme suit :

```
java -Djava.naming.object.factory=com.sybase.jdbc2.jdbc.SybObjectFactory
    ou dans le cadre des propriétés de navigateur.
```

Pour plus d'informations sur la définition des propriétés d'environnement, reportez-vous à la documentation sur la machine virtuelle Java.

2a. Configuration par l'administrateur : personnalisé

Cette procédure est en général effectuée par la personne chargée de l'administration système des bases de données ou de l'intégration des applications dans l'entreprise. L'objectif est de définir une source de données puis de la déployer sous un nom logique sur un serveur de noms. Si le serveur doit être reconfiguré (par exemple transféré vers une autre machine, un port, etc.), l'administrateur exécute l'utilitaire de configuration (décrit plus bas) et réattribue le nom logique à la nouvelle configuration de source de données. En conséquence, le code client ne change pas puisqu'il ne connaît que le nom logique.

```
import javax.sql.*;
import com.sybase.jdbcx.*;
.....

// Créer une source de données SybDataSource et la configurer
SybDataSource ds = new com.sybase.jdbc2.jdbc.SybDataSource();
ds.setUser("mon_nomutilisateur");
ds.setPassword("mon_motdepasse");
ds.setDatabaseName("ma_basededonnées");
ds.setServerName("machine_bd");
ds.setPortNumber(4000);
ds.setDescription("This DataSource represents the Adaptive Server
    Enterprise server running on db_machine at port 2638. The default
    username and password have been set to 'me' and 'mine' respectively.
    Upon connection, the user will access the my_favorite_db database on
    this server.");
Properties props = new Properties();
props.put("REPEAT_READ", "false");
props.put("REQUEST_HA_SESSION", "true");
ds.setConnectionProperties(props);
// Stocker l'objet DataSource object. Il faut pour ce faire
// définir des propriétés JNDI spécifiques du type
// de fournisseur de services JNDI que vous utilisez.
// Puis initialiser le contexte et lier l'objet.
Context ctx = new InitialContext();
ctx.bind("jcbcb/myASE", ds);
```

Une fois que vous avez configuré votre objet **DataSource**, vous indiquez où et quand stocker les informations. Pour vous aider, **SybDataSource** est à la fois **java.io.Serializable** et **javax.naming.Referenceable**, mais il appartient encore à l'administrateur de déterminer le mode de stockage des données, selon le fournisseur de services que vous utilisez pour JNDI.

- 2b. Accès par le client Le client récupère l'objet **DataSource** en définissant ses propriétés JNDI, de la même manière que **DataSource** a été déployé. Le client doit avoir un objet **factory** disponible qui peut transformer l'objet tel qu'il est stocké (par exemple, sérialisé) en un objet Java.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource ctx.lookup("jdbc/myASE"));
```

Connection Pooling

Référence

Document *JDBC 2.0 Optional Package* (anciennement *JDBC 2.0 Standard Extension API*), chapitre 6, "Connection Pooling."

Interfaces associées

- **javax.sql.ConnectionPoolDataSource**
- **javax.sql.PooledConnection**

Présentation

Les applications de bases de données traditionnelles créent une connexion à une base que vous utilisez pour chaque session. Or, une application Web de base de données peut avoir besoin d'ouvrir et de fermer une nouvelle connexion plusieurs fois durant une même session. Dans un environnement Web, un moyen efficace gérer les connexions aux bases de données est d'utiliser le "Connection Pooling", qui permet de garder les connexions ouvertes et de gérer leur partage par les diverses requêtes utilisateur afin de maintenir les performances et de réduire le nombre de connexions inactives. A chaque demande de connexion, le pool détermine d'abord s'il n'en contient pas une qui serait inactive. Si c'est le cas, il la renvoie au lieu d'établir une nouvelle connexion à la base de données.

Les fonctionnalités du "Connection Pooling" sont assurées par l'interface **ConnectionPoolDataSource**. Si vous l'utilisez, vous pouvez regrouper des connexions, ce qui est impossible avec l'interface **DataSource**.

Avec l'interface **ConnectionPoolDataSource**, les mises en oeuvre de pool sont à l'écoute de **PooledConnection**. Lorsqu'un utilisateur ferme la connexion ou qu'une erreur détruisant la connexion lui est adressée, la mise en oeuvre est avertie. Elle effectue alors un choix quant au traitement de **PooledConnection**.

Sans le "Connection Pooling", une transaction :

- 1 crée une connexion avec la base de données ;
- 2 envoie la requête à la base de données ;
- 3 récupère le jeu de résultats ;
- 4 affiche le jeu de résultats ;
- 5 détruit la connexion.

Avec le "Connection Pooling", les opérations sont les suivantes :

- 1 la transaction vérifie s'il existe une connexion inutilisée dans le "pool" des connexions ;
- 2 dans l'affirmative, elle l'utilise ; sinon, elle en crée une nouvelle ;
- 3 elle envoie la requête à la base de données ;
- 4 elle récupère le jeu de résultats ;
- 5 elle affiche le jeu de résultats ;
- 6 elle renvoie la connexion dans le "pool".
(L'utilisateur appelle encore la méthode "**close()**", mais la connexion reste ouverte et le pool est informé de la demande de fermeture.)

Il est moins coûteux de réutiliser une connexion que d'en créer une nouvelle chaque fois qu'un client a besoin de se connecter à une base de données.

Pour permettre à un tiers de mettre en oeuvre le pool de connexions, la mise en oeuvre `JConnect` fait en sorte que l'interface **ConnectionPoolDataSource** génère des **PooledConnections**, tout comme l'interface **DataSource** génère **Connections**.

La mise en oeuvre du pool crée des connexions "réelles" à des bases de données, en utilisant les méthodes **getPooledConnection()** de **ConnectionPoolDataSource**. Puis elle s'enregistre elle-même comme récepteur de **PooledConnection**.

Pratiquement, lorsqu'un client demande une connexion, la mise en oeuvre du pool appelle la méthode **getConnection()** sur un objet **PooledConnection** disponible. Lorsque le client a terminé et appelle **close()**, la mise en oeuvre du pool est avertie via l'interface **ConnectionEventListener** que la connexion est libre et donc réutilisable.

L'interface **ConnectionEventListener** informe également la mise en oeuvre du pool si le client a d'une quelconque manière détérioré la connexion à la base de données, afin qu'elle puisse la supprimer du pool.

Pour plus d'informations, reportez-vous à l'Annexe B du document *JDBC 2.0 Optional Package* (anciennement *JDBC 2.0 Standard Extension API*).

Configuration par
l'administrateur :
LDAP

Cette approche est la même qu'à l'opération "[1a. Configuration par l'administrateur : LDAP](#)" décrite dans la section "[JNDI for Naming Databases](#)", sauf que vous ajoutez une ligne dans votre entrée LDIF. Dans l'exemple suivant, la ligne de code ajoutée apparaît mise en gras.

```
dn:servername=myASE, o=MaSociété, c=FR
1.3.6.1.4.1.897.4.2.5:TCP#1# mamachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=moi&password=secret
1.3.6.1.4.1.897.4.2.11:bdutilisateur
1.3.6.1.4.1.897.4.2.18:ConnectionPoolDataSource
```

Accès par des clients
de niveau
intermédiaire

Cette procédure initialise trois propriétés (INITIAL_CONTEXT_FACTORY, PROVIDER_URL et OBJECT_FACTORIES comme indiqué à la page 85) et récupère un objet **ConnectionPoolDataSource**. Pour un exemple de code plus détaillé, reportez-vous à *sample2/SimpleConnectionPool.java*. La différence essentielle est la suivante :

```
...
ConnectionPoolDatabase cpds = (ConnectionPoolDataSource)
    ctx.lookup("servername=myASE");
PooledConnection pconn = cpds.getPooledConnection();
```

Support de Gestion des transactions distribuées

Cette fonctionnalité fournit une API Java standard permettant d'exécuter des transactions distribuées avec Adaptive Server Enterprise version 12.x ou la version 11.x avec XA-Server.

Remarque Cette fonctionnalité est prévue pour une utilisation dans un environnement multiniveau.

Référence

Chapitre 7, "Distributed Transactions", du document *JDBC 2.0 Optional Package* (anciennement *JDBC 2.0 Standard Extension API*).

Interfaces associées

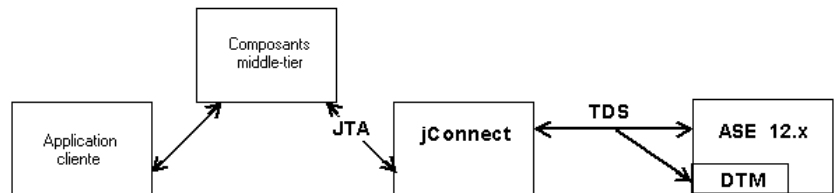
- `javax.sql.XADataSource`
- `javax.sql.XAConnection`
- `javax.transaction.xa.XAResource`

Environnement et configuration système

Pour Adaptive Server Enterprise 12.0

- jConnect communiquant directement avec le gestionnaire de ressources dans Sybase Adaptive Server Enterprise version 12.0, la version installée doit supporter la Gestion des transactions distribuées.
- Les utilisateurs désireux de s'associer à une transaction distribuée doivent posséder le rôle "dtm_tm_role", sinon les transactions échouent.
- Pour utiliser les transactions distribuées, vous devez installer les procédures stockées qui sont dans le répertoire */sp*. Reportez-vous à la section "Installation des procédures stockées" dans le chapitre 1 du document *jConnect for JDBC - Guide d'installation*.

Figure 2-2 : Support de Gestion des transactions distribuées avec la version 12.x

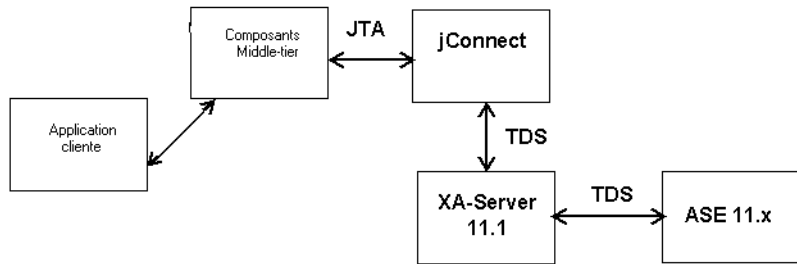


Pour Adaptive Server Enterprise 11.x

jConnect propose également une API Java standard pour exécuter des transactions distribuées avec Adaptive Server Enterprise version 11.x comme serveur de base de données.

- Cette mise en oeuvre ne fonctionne qu'avec Sybase Adaptive Server Enterprise version 11.x et XA-Server 11.1.

Figure 2-3 : Support de Gestion des transactions distribuées avec la version 11.x



- Le login choisi ne peut pas être associé à *master*, *model* ou *sybsystemdb* comme base de données de connexion par défaut. En effet, XA-Server se connecte uniquement lorsque le travail de l'utilisateur est associé à une transaction distribuée ; or celles-ci ne sont pas autorisées sur ces bases de données.
- Aucun accès aux métadonnées n'est possible. Bien qu'il s'agisse d'une restriction au niveau du client, cet accès n'est probablement pas la partie de l'API qui est utilisée dans le cadre des transactions distribuées.

Utilisation d'Adaptive Server Enterprise 12.x

Configuration par
l'administrateur :
LDAP

Cette approche est la même qu'à l'opération "1a. Configuration par l'administrateur : LDAP" décrite dans la section "JNDI for Naming Databases", page 84, sauf que vous ajoutez une ligne à l'entrée LDIF. Dans l'exemple suivant, la ligne de code ajoutée apparaît en gras.

```

dn:servername=myASE, o=MaSociété, c=FR
1.3.6.1.4.1.897.4.2.5:TCP#1# mamachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=moi&password=secret
1.3.6.1.4.1.897.4.2.11:bdutilisateur
1.3.6.1.4.1.897.4.2.18:XADataSource
    
```

Accès par des clients
de niveau
intermédiaire

Cette procédure initialise trois propriétés (INITIAL_CONTEXT_FACTORY, PROVIDER_URL et OBJECT_FACTORIES) et récupère un objet **XADataSource**. Par exemple :

```

...
XADataSource xads = (XADataSource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection();
    
```

ou remplace les valeurs par défaut du nom d'utilisateur et du mot de passe :

```

...
XADataSource xads = (XADataSource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection("my_username", "my_password");
    
```

Utilisation d'Adaptive Server Enterprise 11.x

Configuration par
l'administrateur :
LDAP

Cette approche est la même qu'à l'opération "[1a. Configuration par l'administrateur : LDAP](#)" décrite dans la section "[JNDI for Naming Databases](#)", page 84, à cette différence que vous ajoutez trois lignes dans l'entrée LDIF :

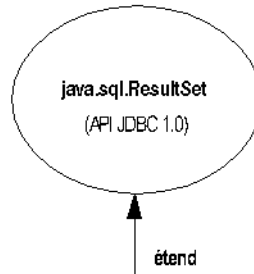
Dans l'exemple ci-dessous, les lignes de code supplémentaires sont affichées en gras.

```
dn:servername:myASE, o=MaSociété, c=FR
1.3.6.1.4.1.897.4.2.5:TCP#1# mamachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=moi&password=secret
1.3.6.1.4.1.897.4.2.11:bdutilisateur
1.3.6.1.4.1.897.4.2.16:userconnection
1.3.6.1.4.1.897:4.2.17:1
1.3.6.1.4.1.897.4.2.18:XADataSource
```

où **...4.2.17:1** indique que jConnect va se connecter à XA-Server et **userconnection** correspond au gestionnaire de ressources logiques (LRM) à utiliser. XA-Server possède un fichier *xa_config* contenant les entrées suivantes :

```
[xa]
lrm=userconnection
server=my_ase_11_server
XAserver=my_xa_server
```

Figure 2-4 : Exemple de configuration pour le support de Gestion des transactions distribuées



Pour plus d'informations sur l'écriture du fichier *xa_config*, reportez-vous à la documentation XA-Server.

Accès par des clients
de niveau
intermédiaire

Cette procédure initialise trois propriétés (INITIAL_CONTEXT_FACTORY, PROVIDER_URL et OBJECT_FACTORIES) et récupère un objet **XADataSource**. Par exemple :

```
...  
XADataSource xads = (XADataSource) ctx.lookup("servername=myASE");  
XAConnection xaconn = xads.getXAConnection();
```

Avec Adaptive Server Enterprise 11.x vous *ne pouvez pas* remplacer le nom d'utilisateur et le mot de passe par défaut; c'est-à-dire que vous ne pouvez pas appeler :

```
xads.getXAConnection("my_username", "my_password");
```

car le gestionnaire de ressources logiques *lrm* est associé à un nom d'utilisateur et un mot de passe spécifiques.

Restrictions, limitations et non-conformités aux normes JDBC

Cette section traite des restrictions et limitations qui s'appliquent à jConnect, y compris les non-conformités de mise en oeuvre de JDBC par jConnect par rapport aux normes JDBC 1.x et 2.0. Les sujets suivants sont traités :

- [Réglage pour multithread](#)
- [Utilisation de `ResultSet.setCursorName\(\)`](#)
- [Utilisation de `setLong\(\)` avec des valeurs de paramètre élevées](#)
- [Utilisation d'instructions `COMPUTE`](#)
- [Exécution de procédures stockées](#)

Réglage pour multithread

Si plusieurs threads appellent simultanément des méthodes sur la même instance **Statement**, **CallableStatement** ou **PreparedStatement** (ce qui n'est pas recommandé), vous devez synchroniser manuellement les appels aux méthodes utilisant **Statement** car jConnect ne les synchronise pas automatiquement.

Par exemple, si deux threads s'exécutent sur la même instance de **Statement** (un thread envoie une requête et l'autre traite les avertissements), vous devez synchroniser les appels vers les méthodes sur **Statement** pour éviter tout risque de conflits.

Utilisation de **ResultSet.setCursorName()**

Certains pilotes JDBC génèrent un nom curseur pour toutes les requêtes SQL, de sorte qu'une chaîne peut toujours être renvoyée. Toutefois, jConnect ne renvoie aucun nom lorsque **ResultSet.setCursorName()** est appelé, à moins que :

- **setFetchSize()** ou **setCursorName()** ne soit appelé sur le **Statement** correspondant, ou
- que la propriété de connexion **SELECT_OPENS_CURSOR** ne soit définie à "true" et que votre requête soit de la forme **SELECT... FOR UPDATE**. Par exemple :

```
select au_id from authors for update
```

Si vous ne procédez pas ainsi, la valeur **NULL** est renvoyée.

Selon l'API JDBC 2.0 (chapitre 11, "Clarifications"), les autres instructions SQL n'ont pas à ouvrir de curseur ni à renvoyer de nom.

Pour plus d'informations sur l'utilisation des curseurs avec jConnect, reportez-vous à la section ["Utilisation de curseurs dans des jeux de résultats"](#), page 46.

Utilisation de **setLong()** avec des valeurs de paramètre élevées

Les mises en oeuvre de la méthode **PreparedStatement.setLong()** associent une valeur de paramètre à un type de données SQL **BIGINT**. La plupart des bases de données Adaptive Server ne disposent pas de type de données **BIGINT** codé sur huit octets. Si une valeur de paramètre requiert plus de 4 octets d'un **BIGINT**, l'utilisation de **setLong()** peut générer une exception d'overflow.

Utilisation d'instructions **COMPUTE**

jConnect ne supporte pas les lignes de résultat. Ainsi, lorsqu'une requête comprend une ligne de résultat, celui-ci est automatiquement annulé. Par exemple, l'instruction suivante est rejetée :

```
SELECT name FROM sysobjects  
WHERE type="S" COMPUTE COUNT(name)
```

Pour éviter ce problème, remplacez l'instruction par le code suivant :

```
SELECT name from sysobjects WHERE type="S"
SELECT COUNT(name) from sysobjects WHERE type="S"
```

Exécution de procédures stockées

- Si vous exécutez une procédure stockée dans un objet **CallableStatement** qui représente des valeurs de paramètre sous forme de points d'interrogation, vous obtenez de meilleures performances que si vous utilisez à la fois des points d'interrogation et des valeurs littérales de paramètre. De plus, la combinaison de littéraux avec des points d'interrogation empêche l'utilisation des paramètres de sortie avec une procédure stockée.

L'exemple ci-dessous crée *sp_stmt* en tant qu'objet **CallableStatement** permettant l'exécution de la procédure stockée **MyProc** :

```
CallableStatement sp_stmt = conn.prepareCall(
    "{call MyProc(?,?)}");
```

Les deux paramètres dans **MyProc** sont représentés sous la forme de points d'interrogation. Vous pouvez enregistrer ces paramètres en tant que paramètres de sortie à l'aide des méthodes **registerOutParameter()** de l'interface **CallableStatement**.

Dans l'exemple ci-dessous, *sp_stmt2* est un objet **CallableStatement** permettant l'exécution de la procédure stockée **MyProc2**.

```
CallableStatement sp_stmt2 = conn.prepareCall(
    "{call MyProc2(?, 'javelin')}");
```

Dans *sp_stmt2*, l'une des valeurs de paramètre est un littéral tandis que l'autre est un point d'interrogation. Vous ne pouvez enregistrer aucun de ces paramètres en tant que paramètre de sortie.

- Pour exécuter des procédures stockées avec des commandes RPC en associant des noms à des paramètres, procédez de l'une des façons suivantes :
 - Utilisez des commandes de langage auxquelles vous transmettez des paramètres d'entrée directement à partir de variables Java via la classe **PreparedStatement**. Ce mécanisme est illustré dans l'extrait de code ci-après :

```
// Préparer l'instruction
System.out.println("Preparing the statement...");
String stmtString = "exec " + procname + " @p3=?, @p1=?";
```

```
PreparedStatement pstmt = con.prepareStatement(stmtString);
```

```
// Définir les valeurs  
pstmt.setString(1, "xyz");  
pstmt.setInt(2, 123);
```

```
// Envoyer la requête  
System.out.println("Executing the query...");  
ResultSet rs = pstmt.executeQuery();
```

- Avec jConnect version 5.2, utilisez l'interface **com.sybase.jdbcx.SybCallableStatement** comme indiqué dans l'exemple suivant :

```
import com.sybase.jdbcx.*;  
....  
// préparer la procédure stockée pour qu'elle s'exécute comme RPC  
String execRPC = "{call " + procName + " (?, ?)}";  
SybCallableStatement scs = (SybCallableStatement)  
con.prepareCall(execRPC);  
  
// Définir les valeurs, nommer les paramètres  
// et (facultatif) enregistrer les paramètres de sortie  
scs.setString(1, "xyz");  
scs.setParameterName(1, "@p3");  
scs.setInt(2, 123);  
scs.setParameterName(2, "@p1");  
  
// Exécuter l'appel RPC  
// may also process the results using getResultSet()  
// and getMoreResults()  
  
// Voir les exemples pour plus de détails sur le traitement  
// des résultats  
ResultSet rs = scs.executeQuery();
```


Le présent chapitre décrit les solutions et les astuces permettant de résoudre les problèmes que vous êtes susceptible de rencontrer lors de l'utilisation de jConnect.

Ce chapitre comprend les sections suivantes :

Nom	Page
Débogage avec jConnect	100
Capture des communications TDS	104
Erreurs lors d'échecs de connexion	106
Utilisation de la mémoire dans les applications jConnect	108
Erreurs de procédure stockée	109
Erreur de mise en oeuvre de socket personnalisé	111

Débogage avec jConnect

jConnect intègre une classe **Debug** contenant un ensemble de fonctions de débogage. Les méthodes **Debug** intègrent toute une gamme de fonctions assert, trace et timer permettant de définir la portée du processus de débogage et la destination des résultats.

L'installation jConnect inclut également un ensemble complet de classes autorisant le débogage. Ces classes sont placées dans le sous-répertoire *devclasses* du répertoire d'installation de jConnect. Pour démarrer le débogage, vous devez rediriger la variable d'environnement CLASSPATH vers les classes d'exécution du mode de débogage (*devclasses* pour jConnect 4.x et *devclasses/jconn2d.jar* pour jConnect 5.x), plutôt que vers le répertoire jConnect standard *classes*. L'autre possibilité consiste à passer explicitement un argument **-classpath** à la commande **java** lorsque vous exécutez un programme Java.

Obtention d'une instance de la classe Debug

Pour pouvoir utiliser la fonction de débogage de jConnect, vous devez importer dans votre application l'interface **Debug** et obtenir une instance de la classe **Debug** en appelant la méthode **getDebug()** pour la classe **SybDriver**.

Pour jConnect 4.x :

```
import com.sybase.jdbcx.Debug
import com.sybase.jdbcx.SybDebug
//
...
SybDriver sybDriver = (SybDriver)
Class.forName("com.sybase.jdbc.SybDriver").newInstance();
Debug sybdebug = sybDriver.getDebug();
...
```

Pour jConnect 5.x :

```
import com.sybase.jdbcx.Debug
import com.sybase.jdbcx.SybDebug
//
...
SybDriver sybDriver = (SybDriver)
Class.forName("com.sybase.jdbc2.jdbc.SybDriver").newInstance();
Debug sybdebug = sybDriver.getDebug();
...
```

Activation du débogage dans votre application

Utilisez la méthode **debug()** sur l'objet **Debug** pour activer le débogage dans votre application et ajoutez l'appel suivant :

```
sybdebug.debug(true, [classes], [printstream]);
```

Le paramètre *classes* est une chaîne qui répertorie les classes à déboguer ; les classes sont séparées par le signe "deux-points". Par exemple :

```
sybdebug.debug(true, "MaClasse")
```

et

```
sybdebug.debug(true, "MaClasse:TaClasse")
```

Si vous incluez "STATIC" dans la chaîne de classe, le débogage est activé pour toutes les méthodes statiques de jConnect, en plus des classes désignées. Par exemple :

```
sybdebug.debug(true, "STATIC:MaClasse")
```

Vous pouvez spécifier "ALL" pour activer le débogage dans toutes les classes. Par exemple :

```
sybdebug.debug(true, "ALL");
```

Le paramètre *printstream* est facultatif. Si vous ne le spécifiez pas, le résultat du débogage est envoyé dans le fichier de résultat désigné par la commande **DriverManager.setLogStream()**.

Désactivation du débogage dans votre application

Pour désactiver le débogage, ajoutez l'appel suivant :

```
sybdebug.debug(false);
```

Définition de CLASSPATH pour le débogage

Avant d'exécuter votre application avec le débogage activé, définissez dans la variable d'environnement CLASSPATH le sous-répertoire */devclasses* du répertoire d'installation de jConnect.

Pour jConnect 4.x :

- Sous UNIX, remplacez *\$JDBC_HOME/classes* par *\$JDBC_HOME/devclasses*.

- Sous Windows, remplacez `%JDBC_HOME%\classes` par `%JDBC_HOME%\devclasses`.

Pour jConnect 5.x :

- Sous UNIX, remplacez `$JDBC_HOME/classes/jconn2.jar` par `$JDBC_HOME/devclasses/jconn2.jar`.
- Sous Windows, remplacez `%JDBC_HOME%\classes\jconn2.jar` par `%JDBC_HOME%\devclasses\jconn2.jar`.

Utilisation des méthodes de débogage

Si vous souhaitez personnaliser le processus de débogage, vous pouvez ajouter des appels dans d'autres méthodes **Debug**.

Dans ces méthodes, le premier paramètre (objet) utilisé pour spécifier l'objet appelant est généralement désigné par *ceci*. Si l'une de ces méthodes est statique, utilisez le paramètre d'objet *null*.

- **println()**

Utilisez cette méthode pour définir le message à imprimer dans le fichier de résultat si le débogage est activé et l'objet inclus dans la liste des classes à déboguer. Le résultat du débogage est envoyé dans le fichier désigné par la commande `sybdebug.debug()`.

La syntaxe est la suivante :

```
sybdebug.println(objet, chaîne du message);
```

Par exemple :

```
sybdebug.println(ceci, "Query: " + requête);
```

génère un message semblable à celui-ci dans le fichier de résultat :

```
myApp(thread[x,y,z]): Query: select * from authors
```

- **assert()**

Utilisez cette méthode pour déclarer une condition et provoquer une exception d'exécution lorsqu'elle n'est pas remplie. Vous pouvez également définir le message à imprimer dans le fichier de résultat si la condition n'est pas remplie. La syntaxe est la suivante :

```
sybdebug.assert(objet, condition booléenne, chaîne  
string);
```

Par exemple :

```
sybdebug.assert(ceci,amount<=buf.length,amount+"
too big!");
```

génère un message semblable à celui-ci dans le fichier de résultat si “amount” dépasse la valeur de *buf.length* :

```
java.lang.RuntimeException:myApp(thread[x,y,z]):
Assertion failed: 513 too big!
at jdbc.sybase.utils.sybdebug.assert(
sybdebug.java:338)
at myApp.myCall(myApp.java:xxx)
at .... more stack:
```

- **startTimer()**
stopTimer()

Utilisez ces méthodes pour démarrer et arrêter un compteur mesurant le nombre de millisecondes qui s'écoulent au cours d'un événement. Cette méthode réserve un compteur pour chaque objet et un autre pour toutes les méthodes statiques. La syntaxe de la commande de démarrage du compteur est la suivante :

```
sybdebug.startTimer(objet);
```

La syntaxe de la commande d'arrêt du compteur est la suivante :

```
sybdebug.stopTimer(objet,chaîne de message);
```

Par exemple :

```
sybdebug.startTimer(ceci);
stmt.executeQuery(requête);
sybdebug.stopTimer(this,"executeQuery");
```

génère un message semblable à celui-ci dans le fichier de résultats :

```
myApp(thread[x,y,z]):executeQuery elapsed time =
25ms
```

Capture des communications TDS

Tabular Data Stream (TDS) est le protocole Sybase qui gère les communications entre une application cliente et Adaptive Server. jConnect intègre une propriété de connexion `PROTOCOL_CAPTURE` qui permet de capturer les paquets TDS bruts et de les consigner dans un fichier.

Si vous rencontrez des problèmes dans une application et que vous ne parvenez pas à les résoudre ni dans l'application, ni sur le serveur, vous pouvez utiliser `PROTOCOL_CAPTURE` pour capturer les échanges entre le client et le serveur dans un fichier. Vous pouvez ensuite envoyer le fichier (contenant des données binaires) au Support Technique de Sybase, afin de le faire analyser.

Remarque Vous pouvez également utiliser **Ribo**, un utilitaire de capture, de conversion et d'affichage du flux de protocoles entre le client et le serveur. Pour plus d'informations sur **Ribo**, consultez le site des utilitaires jConnect sur le Web, à l'adresse :

<http://www.sybase.com/products/internet/jconnect/utilities/>

Propriété de connexion `PROTOCOL_CAPTURE`

La propriété de connexion `PROTOCOL_CAPTURE` permet de consigner dans un fichier les paquets TDS échangés entre une application et un Adaptive Server. Elle prend effet immédiatement : les paquets échangés lors de l'établissement de la connexion sont donc également enregistrés dans le fichier spécifié. Tous les paquets sont copiés dans le fichier jusqu'à ce que la commande **Capture.pause()** soit exécutée ou la session fermée.

L'exemple suivant illustre l'utilisation de `PROTOCOL_CAPTURE` pour envoyer des données TDS dans le fichier `tds_data` :

```
...
props.put("PROTOCOL_CAPTURE", "tds_data")
Connection conn = DriverManager.getConnection(url, props);
```

où *url* désigne l'URL de connexion et *props* un objet **Properties** spécifiant les propriétés de la connexion.

Méthodes `pause()` et `resume()` dans la classe `Capture`

La classe **Capture** se trouve dans le package **com.sybase.jdbcx**. Elle contient deux méthodes publiques :

- **public void pause()**
- **public void resume()**

Capture.pause() arrête la capture des paquets TDS bruts dans un fichier, tandis que **Capture.resume()** le redémarre.

Le fichier de capture TDS de toute une session peut devenir très volumineux. Si vous souhaitez limiter la taille de ce fichier et que vous avez déterminé un intervalle de capture des données TDS, procédez comme suit :

- 1 Immédiatement après avoir établi une connexion, appelez l'objet **Capture** de la connexion et utilisez la méthode **pause()** pour arrêter la collecte des données TDS :

```
Capture cap = ((SybConnection)conn).getCapture();  
cap.pause();
```

- 2 Indiquez **cap.resume()** au moment où vous souhaitez commencer la collecte des données TDS.
- 3 Indiquez **cap.pause()** au moment où vous souhaitez arrêter la collecte des données TDS.

Erreurs lors d'échecs de connexion

Cette section traite des problèmes susceptibles de se produire lorsque vous essayez d'établir une connexion ou de démarrer une passerelle.

Connexion refusée à la passerelle

```
Connexion refusée à la passerelle :  
HTTP/1.0 502 Bad Gateway|Restart Connection
```

Ce message d'erreur indique un problème avec le *nom d'hôte* ou le numéro de *port* utilisé pour vous connecter à votre Adaptive Server. Vérifiez l'entrée [query] de *\$SYBASE/interfaces* (UNIX) ou de *%SYBASE%\ini\sql.ini* (Windows).

Si le problème persiste après que vous avez vérifié le *nom d'hôte* et le *numéro de port*, démarrez le serveur HTTP en mode détaillé afin d'obtenir des informations complémentaires.

Sous Windows, placez-vous à la suite de l'invite DOS et entrez :

```
httpd -Dverbose=1 > nom_fichier
```

Sous UNIX, entrez :

```
sh httpd.sh -Dverbose=1 > nom_fichier &
```

où *nom_fichier* désigne le fichier de résultat dans lequel sont consignés les messages de débogage.

Si vous ne parvenez pas à établir la connexion via la passerelle Cascade HTTP, c'est probablement parce que votre serveur Web ne prend pas en charge cette méthode de connexion. Les applets ne peuvent se connecter qu'à l'hôte depuis lequel elles ont été téléchargées, à moins que vous n'utilisiez la passerelle Cascade HTTP, qui permet d'accéder au serveur de base de données en tant que serveur proxy.

La passerelle Cascade HTTP et votre serveur Web doivent s'exécuter sur la même machine hôte. Dans ce scénario, votre applet peut se connecter à la même machine ou au même hôte via le port contrôlé par la passerelle Cascade HTTP, qui transmet la requête à la base de données appropriée.

Pour plus d'informations sur ce processus, examinez le code source dans les fichiers *Isql.java* et *gateway.html* situés dans le sous-répertoire *sample* (jConnect 4.x) ou *sample2* (jConnect 5.x) du répertoire d'installation jConnect. Recherchez le terme "proxy".

Connexion à un serveur SQL 4.9.2 impossible

jConnect utilise TDS 5.0 (protocole de transfert Sybase). SQL Server 4.9.x utilise TDS 4.6, qui n'est pas compatible avec TDS 5.0.

L'utilisation de jConnect requiert SQL Server 10.0.2 ou version supérieure.

Utilisation de la mémoire dans les applications jConnect

Si vous constatez une augmentation de la mémoire utilisée par les applications jConnect, lisez attentivement les situations et les solutions correspondantes présentées ci-après :

- Dans les applications jConnect, vous devez fermer de manière explicite tous les objets et sous-classes **Statement** (par exemple, **PreparedStatement**, **CallableStatement**) après leur dernière utilisation pour éviter que les instructions ne s'accumulent dans la mémoire. Fermer **ResultSet** ne suffit pas.

Par exemple :

```
ResultSet rs =  
_conn.prepareCall(_requête).execute();  
...  
rs.close();
```

génère des problèmes. Utilisez plutôt :

```
PreparedStatement ps = _conn.prepareCall(_requête);  
ResultSet rs = ps.execute();  
...  
ps.close();  
rs.close();
```

- jConnect utilise le protocole TDS (Tabular Data Stream) de Sybase pour communiquer avec les serveurs de données Sybase. Dans les jConnect 5.0 ou version supérieure, TDS ne supporte pas les curseurs avec défilement. Pour les prendre en charge, jConnect met en mémoire cache les données de ligne sur demande, sur le client et sur chaque appel **ResultSet.next()**. Toutefois, une fois la fin du jeu de résultats atteinte, tout le jeu de résultats est enregistré dans la mémoire du client. En raison des contraintes liées aux performances, il est recommandé de n'utiliser les jeux de résultats **TYPE_SCROLL_INSENSITIVE** que lorsque le jeu de résultats est réduit.

Erreurs de procédure stockée

Cette section traite des problèmes susceptibles de se produire lorsque vous essayez d'utiliser jConnect et des procédures stockées.

Le RPC renvoie moins de paramètres de sortie que prévu

SQLState: JZ0SG - Un RPC a renvoyé moins de paramètres de sortie que prévu pour l'application.

Cette erreur se produit si vous appelez dans

CallableStatement.registerOutParam() plus de paramètres que vous avez déclaré de paramètres "OUTPUT" dans la procédure stockée. Assurez-vous d'avoir déclaré tous les paramètres adéquats comme "OUTPUT."

Reportez-vous à la ligne de code suivante :

```
create procedure yourproc (@p1 int OUTPUT, ...
```

Remarque Si cette erreur survient lorsque vous utilisez Adaptive Server Anywhere (anciennement SQL Anywhere), vous devez installer la version 5.5.04 ou supérieure d'Adaptive Server Anywhere.

Erreur fetch/state en cas de renvoi de paramètres Output par une procédure stockée

Si une requête ne renvoie pas de lignes, elle doit utiliser les méthodes

CallableStatement.executeUpdate() ou **execute()** au lieu de la méthode **executeQuery()**.

Comme l'exige la norme JDBC, jConnect provoque une exception SQL si **executeQuery()** ne renvoie pas de jeu de résultats.

Procédure stockée exécutée en mode non chaîné

Erreur Sybase 7713 - La procédure chaînée ne peut être exécutée qu'en mode non chaîné.

JDBC essaie de placer la connexion en mode **autocommit(true)**. L'application peut activer le mode chaîné à l'aide de **Connection.setAutoCommit(false)** ou en utilisant une commande de langage "**set chained on**". Cette erreur se produit si la procédure stockée n'a pas été créée dans un mode compatible.

Pour résoudre le problème, utilisez la procédure système suivante :

```
sp_procxmode nom_procedure, "anymode"
```

Erreur de mise en oeuvre de socket personnalisé

Si vous recevez une exception identique à l'exception suivante lorsque vous essayez de configurer un socket SSL en appelant

sun.security.ssl.SSLSocketImpl.setEnabledCipherSuites:

```
java.lang.IllegalArgumentException:  
SSL_SH_anon_EXPORT_WITH_RC4_40_MDS
```

vérifiez que les bibliothèques SSL se trouvent dans le chemin d'accès à la bibliothèque système.

Le présent chapitre décrit comment optimiser les performances avec jConnect.

Les sujets suivants sont traités :

Nom	Page
Optimisation des performances de jConnect	114
Optimisation des performances des instructions préparées en SQL dynamique	117
Performances du curseur	124

Optimisation des performances de jConnect

Il existe plusieurs moyens d'optimiser les performances d'une application à l'aide de jConnect. Voici quelques suggestions :

- Utilisez les méthodes **TextPointer.sendData()** pour transmettre des données de type text et image à une base de données Adaptive Server. Reportez-vous à la section [“Envoi de données Image”, page 61](#).
- Créez des objets **PreparedStatement** précompilés pour les instructions SQL dynamiques utilisées fréquemment au cours d'une session. Reportez-vous à la section [“Optimisation des performances des instructions préparées en SQL dynamique”, page 117](#).
- Les mises à jour par batch améliorent les performances en réduisant le trafic réseau. De manière spécifique, toutes les requêtes envoyées au serveur et toutes les réponses transmises au client sont regroupées. Reportez-vous à la section [“Support pour les mises à jour par batch”, page 57](#).
- Pour les sessions qui sont susceptibles de déplacer des données image, des jeux de ligne volumineux et des données texte importantes, définissez la taille de paquet maximale à l'aide de la propriété de connexion **PACKETSIZE**.
- Dans le cas du protocole HTTP encapsulé par TDS, augmentez la taille de paquet TDS au maximum et configurez le serveur Web pour la prise en charge de la fonctionnalité Keep-Alive de HTTP1.1. En outre, attribuez la valeur “true” à l'argument servlet *SkipDoneProc* reportez-vous à [///vi](#)).
- Utilisez des curseurs de protocole (valeur par défaut de la propriété de connexion **LANGUAGE_CURSOR**). Pour plus d'informations, reportez-vous à la section [“Propriété de connexion LANGUAGE_CURSOR”, page 124](#).
- N'utilisez des jeux de résultats **TYPE_SCROLL_INSENSITIVE** que si le jeu de résultats est de taille raisonnable. Pour plus d'informations, reportez-vous à la section [“Support pour jeux de résultats SCROLL_INSENSITIVE dans jConnect”, page 55](#).

Vous trouverez ci-après d'autres points à prendre en compte pour l'optimisation des performances.

Remise à l'échelle de BigDecimal

Avec JDBC 1.0, vous devez associer un facteur d'échelle à `getBigDecimal()`. Lorsqu'un objet **BigDecimal** est ensuite renvoyé du serveur, il doit être remis à l'échelle conformément à la valeur d'origine de `getBigDecimal()`.

Pour ne pas avoir à traiter ce processus, utilisez la méthode `getBigDecimal()` de JDBC 2.0 que jConnect met en oeuvre dans la classe **SybResultSet** et qui ne nécessite pas de valeur d'échelle :

```
public BigDecimal getBigDecimal(int columnIndex)
    throws SQLException
```

Par exemple :

```
SybResultSet rs =
    (SybResultSet)stmt.executeQuery("SELECT
    numeric_column from T1");
while (rs.next())
{
    BigDecimal bd rs.getBigDecimal(
        "numeric_column");
    ...
}
```

Propriété de connexion REPEAT_READ

Pour améliorer les performances lors de l'extraction d'un jeu de résultats de la base de données, vous pouvez désactiver la propriété de connexion `REPEAT_READ`. Toutefois, lorsque `REPEAT_READ` a la valeur "false" :

- Vous devez lire les valeurs de colonne dans l'ordre des index de colonne. Cela peut être problématique si vous voulez accéder aux colonnes par leur nom plutôt que par leur numéro.
- Vous ne pouvez pas effectuer de lectures successives d'une valeur de colonne.

Conversion d'un jeu de caractères

Si vous utilisez des jeux de caractères codés sur plusieurs octets et que vous devez améliorer les performances du pilote, vous pouvez utiliser la classe **SunloConverter** fournie avec les exemples de jConnect. Ce convertisseur s'appuie sur les classes **sun.io** de Java Software Division de Sun Microsystems, Inc.

La classe **SunloConverter** n'est pas une mise en oeuvre purement Java de la fonctionnalité de convertisseur de jeu de caractères. C'est la raison pour laquelle il n'est pas intégré dans le produit jConnect standard. Il vous est néanmoins fourni pour que vous puissiez améliorer les performances lors du processus de conversion, si nécessaire.

Remarque Des tests réalisés par Sybase sur plusieurs machines virtuelles (VM) ont révélé que la classe **SunloConverter** améliore systématiquement les performances. Toutefois, Java Software Division of Sun Microsystems, Inc. se réserve le droit de supprimer ou de modifier les classes **sun.io** dans les versions ultérieures de JDK, dans les versions ultérieures de JDK, ce qui risque de rendre la classe **SunloConverter** incompatible avec les futures versions de JDK

Pour utiliser la classe **SunloConverter**, vous devez installer les applications exemple jConnect. Pour plus d'informations sur l'installation de jConnect et de ses composants, y compris les exemples, reportez-vous au document *Sybase jConnect for JDBC - Guide d'installation*. Une fois les exemples installés, vous pouvez, par l'intermédiaire de la propriété de connexion `CHARSET_CONVERTER_CLASS`, définir la classe **SunloConverter** dans le sous-répertoire *sample* (jConnect 4.x) ou *sample2* (jConnect 5.x) de votre répertoire d'installation jConnect.

Optimisation des performances des instructions préparées en SQL dynamique

En SQL intégré, les instructions dynamiques sont des instructions SQL compilées de façon dynamique (au moment de l'exécution) plutôt que de façon statique. Bien que cela ne soit pas obligatoire, les instructions dynamiques contiennent généralement des paramètres d'entrée. En SQL, la commande **prepare** permet de précompiler une instruction dynamique et de l'enregistrer de sorte qu'elle puisse être réutilisée à plusieurs reprises dans une même session sans être recompilée à chaque fois.

Lorsqu'une instruction est appelée à être utilisée plusieurs fois dans une même session, ce mécanisme de précompilation est plus efficace que celui qui consiste à envoyer l'instruction à la base de données pour être compilée à chaque utilisation. Les performances seront d'autant améliorées que l'instruction est complexe.

Toutefois, ce mécanisme implique une surcharge liée à la précompilation même, à l'enregistrement et à la libération d'une instruction dans la base de données. Il peut donc être inefficace si vous n'utilisez une instruction qu'occasionnellement.

La précompilation d'une instruction SQL dynamique et son enregistrement en mémoire prennent du temps et monopolisent des ressources. Toutefois, si vous ne comptez pas utiliser une instruction plusieurs fois lors d'une même session, il peut être plus rentable d'abandonner ce mécanisme. En outre, notez qu'une fois préparée dans la base de données, une instruction SQL s'apparente à une procédure stockée. Ainsi, il peut être préférable de créer des procédures stockées hébergées sur le serveur, au lieu de définir des instructions préparées dans l'application. Vous trouverez plus d'informations à ce sujet à la section [“Instructions préparées et procédures stockées”, page 118](#).

Pour optimiser les performances d'instructions SQL dynamiques dans une base de données Sybase avec jConnect, procédez comme suit :

- Créez des objets **PreparedStatement** contenant des instructions précompilées (les instructions seront utilisées plusieurs fois dans une même session).
- Créez des objets **PreparedStatement** contenant des instructions SQL non compilées (les instructions ne seront utilisées qu'occasionnellement dans une même session).

Comme il est décrit dans les sections suivantes, le moyen le plus efficace de définir la propriété de connexion `DYNAMIC_PREPARE` et de créer des objets **PreparedStatement** dépend généralement de la portée de votre application : doit-elle être portable entre différents pilotes JDBC ou autorise-t-elle l'utilisation d'extensions JDBC spécifiques de `jConnect` ?

Les versions 4.1 et supérieures de `jConnect` offrent des fonctionnalités qui améliorent les performances des instructions SQL dynamiques.

Instructions préparées et procédures stockées

Une fois compilée dans la base de données, une instruction SQL dynamique appartenant à un objet **PreparedStatement** devient une procédure stockée conservée en mémoire et liée à la structure de données relative à votre session. Avant d'opter pour le maintien des procédures stockées dans la base de données ou la création d'objets **PreparedStatement** contenant des instructions SQL compilées dans votre application, tenez compte des exigences en termes de ressources système et de maintenance de base de données et d'application.

- Une fois compilée, une procédure stockée est généralement disponible sur l'ensemble des connexions. A l'inverse, une instruction SQL dynamique contenue dans un objet **PreparedStatement** doit être compilée et libérée dans chaque session qui l'utilise.
- Si votre application accède à plusieurs bases de données et qu'elle utilise des procédures stockées, ceci implique ces procédures sont disponibles sur toutes les bases cibles. Cela peut alourdir le processus de maintenance des bases de données. Vous éviterez cet inconvénient en utilisant des objets **PreparedStatement** qui contiennent des instructions SQL dynamiques.
- Si votre application crée des objets **CallableStatement** pour appeler des procédures stockées, vous pouvez encapsuler du code SQL et des références aux tables dans les procédures stockées. Vous pouvez alors modifier la base de données sous-jacente ou le code SQL sans changer d'application.

Instructions préparées dans des applications portables

Pour que votre application s'exécute sur des bases de données d'éditeurs différents et que certains objets **PreparedStatement** contiennent des instructions précompilées ou non compilées, suivez la procédure ci-dessous :

- Lorsque vous accédez à une base de données Sybase, vérifiez que la propriété de connexion **DYNAMIC_PREPARE** a la valeur "true".
- Pour renvoyer des objets **PreparedStatement** contenant des instructions précompilées, utilisez **Connection.prepareStatement()** normalement :

```
PreparedStatement ps_precomp =
    Connection.prepareStatement(chaîne_sql);
```

- Pour renvoyer des objets **PreparedStatement** contenant des instructions non compilées, utilisez **Connection.prepareCall()**.

Connection.prepareCall() renvoie un objet **CallableStatement**, mais celui-ci étant une sous-classe de **CallableStatement** de **PreparedStatement**, vous pouvez surclasser un objet **CallableStatement** en objet **PreparedStatement**, comme indiqué dans l'exemple ci-dessous :

```
PreparedStatement ps_uncomp =
    Connection.prepareCall(chaîne_sql);
```

L'objet **PreparedStatement** *ps_uncomp* est assuré de contenir une instruction non compilée, puisque seule la méthode **Connection.prepareStatement()** est configurée pour renvoyer des objets **PreparedStatement** contenant des instructions précompilées.

Instructions préparées dans des applications dotées d'extensions jConnect

Si la portabilité entre pilotes ne vous intéresse pas, vous pouvez dans le code demander à ce que la méthode **SybConnection.prepareStatement()** définisse le contenu d'un objet **PreparedStatement**, à savoir des instructions précompilées ou non compilées. Dans ce cas, le type de codage des instructions préparées dépend de la fréquence d'exécution des instructions dynamiques au cours d'une même session.

Si la plupart des instructions dynamiques ne sont exécutées qu'occasionnellement (une fois ou deux tout au plus)

Dans ce cas, procédez comme suit :

- Désactivez la propriété de connexion `DYNAMIC_PREPARE` en lui attribuant la valeur “false”.
- Pour renvoyer des objets **PreparedStatement** contenant des instructions non compilées, utilisez **Connection.prepareStatement()** normalement :

```
PreparedStatement ps_uncomp =  
    Connection.prepareStatement(chaîne_sql) ;
```

- Pour renvoyer des objets **PreparedStatement** contenant des instructions précompilées, utilisez la méthode **SybConnection.prepareStatement()** en spécifiant la valeur “true” pour le paramètre *dynamic* :

```
PreparedStatement ps_precomp =  
    (SybConnection)conn.prepareStatement(chaîne_sql, true) ;
```

Si la plupart des instructions dynamiques sont exécutées plusieurs fois dans une session

Dans ce cas, procédez comme suit :

- Activez la propriété de connexion `DYNAMIC_PREPARE` en lui attribuant la valeur “true”.
- Pour renvoyer des objets **PreparedStatement** contenant des instructions précompilées, utilisez **Connection.prepareStatement()** normalement :

```
PreparedStatement ps_precomp =  
    Connection.prepareStatement(chaîne_sql) ;
```

- Pour renvoyer des objets **PreparedStatement** contenant des instructions non compilées, vous pouvez utiliser **Connection.prepareCall()** (voir la troisième puce de la section [Instructions préparées dans des applications portables](#)) ou **SybConnection.prepareStatement()** en spécifiant “false” pour le paramètre *dynamic* :

```
PreparedStatement ps_uncomp =  
    (SybConnection)conn.prepareStatement(chaîne_sql, false) ;
```

```
PreparedStatement ps_uncomp =  
    Connection.prepareCall(chaîne_sql) ;
```

Connection.prepareStatement()

jConnect met en oeuvre la méthode **Connection.prepareStatement()** pour qu'elle renvoie les instructions SQL précompilées ou non compilées dans les objets **PreparedStatement**. Si **Connection.prepareStatement()** est configurée pour renvoyer des instructions SQL précompilées dans les objets **PreparedStatement**, elle envoie des instructions SQL dynamiques à la base de données, où elles sont précompilées et enregistrées. Vous obtenez le même résultat qu'avec la commande **prepare**. Si **Connection.prepareStatement()** est configurée pour renvoyer des instructions SQL non compilées, elle les renvoie dans des objets **PreparedStatement** sans passer par la base de données.

Le type d'instruction SQL renvoyé par **Connection.prepareStatement()** est déterminé par la propriété de connexion **DYNAMIC_PREPARE** ; il s'applique pendant toute la durée d'une session.

Pour les applications Sybase, jConnect 5.0 fournit une méthode **prepareStatement()** dans la classe **SybConnection**.

SybConnection.prepareStatement() vous permet de préciser si une instruction SQL dynamique individuelle doit être précompilée, indépendamment du paramétrage de la propriété **DYNAMIC_PREPARE** au niveau session.

Propriété de connexion DYNAMIC_PREPARE

DYNAMIC_PREPARE est une propriété de connexion de type booléen qui permet d'activer les instructions SQL dynamiques préparées :

- Si **DYNAMIC_PREPARE** a la valeur "true", chaque appel de la méthode **Connection.prepareStatement()** lors d'une session provoque le renvoi d'une instruction précompilée dans un objet **PreparedStatement**.

Dans ce cas, lorsqu'un objet **PreparedStatement** est exécuté, l'instruction qu'il contient est déjà précompilée dans la base de données, avec des espaces réservés pour les valeurs affectées de façon dynamique ; il ne reste plus qu'à exécuter l'instruction.

- Si **DYNAMIC_PREPARE** a la valeur "false" pour une connexion, l'objet **PreparedStatement** renvoyé par **Connection.prepareStatement()** ne contient pas d'instruction précompilée.

Dans ce cas, dès qu'un objet **PreparedStatement** est exécuté, l'instruction SQL dynamique qu'il contient doit être envoyée à la base de données pour y être compilée et exécutée.

La valeur par défaut de `DYNAMIC_PREPARE` is “false”.

Dans l'exemple suivant, `DYNAMIC_PREPARE` a la valeur “true” pour permettre la précompilation des instructions SQL dynamiques. Ici, **props** est un objet **Properties** qui définit les propriétés de connexion.

```
...
props.put("DYNAMIC_PREPARE", "true")
Connection conn = DriverManager.getConnection(url, props);
```

Lorsque `DYNAMIC_PREPARE` a la valeur “true”, il convient de noter les points suivants :

- Toutes les instructions dynamiques ne peuvent pas être précompilées avec la commande **prepare**. En effet, la norme SQL-92 impose des restrictions quant aux instructions compatibles avec la commande **prepare** ; en outre, il peut en être de même pour certains éditeurs tiers de bases de données.
- Si la base de données génère une erreur lors de la précompilation et l'enregistrement d'une instruction qui lui a été envoyée par **Connection.prepareStatement()**, `JConnect` intercepte l'erreur et renvoie un objet **PreparedStatement** contenant une instruction SQL dynamique non compilée. A chaque exécution de l'objet **PreparedStatement**, l'instruction est renvoyée à la base de données pour y être compilée et exécutée.
- Une instruction précompilée reste en mémoire dans la base de données et persiste jusqu'à la fin de la session en cours ou jusqu'à ce que son objet **PreparedStatement** soit explicitement fermé. L'identification des incohérences sur un objet **PreparedStatement** ne supprime pas l'instruction préparée de la base de données.

D'une manière générale, il est recommandé de fermer tous les objets **PreparedStatement** explicitement après leurs dernières utilisations. Vous éviterez ainsi l'accumulation d'instructions préparées en mémoire lors d'une session, ainsi que la dégradation des performances.

SybConnection.prepareStatement()

Si votre application admet les extensions JDBC spécifiques de jConnect, vous pouvez utiliser la méthode d'extension **SybConnection.prepareStatement()** pour renvoyer des instructions SQL dynamiques dans des objets

PreparedStatement :

```
PreparedStatement SybConnection.prepareStatement(String sql_stmt,
    boolean dynamic) throws SQLException
```

SybConnection.prepareStatement() peut renvoyer des objets **PreparedStatement** contenant des instructions SQL précompilées ou non compilées, selon la valeur du paramètre *dynamic*. Si *dynamic* a la valeur "true", **SybConnection.prepareStatement()** renvoie un objet **PreparedStatement** contenant une instruction SQL précompilée. Si *dynamic* a la valeur "false", la méthode renvoie un objet **PreparedStatement** contenant une instruction SQL non compilée.

L'exemple suivant illustre l'utilisation de

SybConnection.prepareStatement() pour renvoyer un objet **PreparedStatement** contenant une instruction précompilée :

```
PreparedStatement precomp_stmt =
    ((SybConnection) conn).prepareStatement( "SELECT * FROM
    authors WHERE au_fname LIKE ?", true);
```

Ici, l'objet de connexion *conn* est déclassé en objet **SybConnection** pour permettre l'utilisation de **SybConnection.prepareStatement()**. Notez que la chaîne SQL transmise à **SybConnection.prepareStatement()** sera précompilée dans la base de données, même si la propriété de connexion **DYNAMIC_PREPARE** est désactivée ("false").

Si la base de données génère une erreur de précompilation d'une instruction qui lui a été transmise par la méthode **SybConnection.prepareStatement()**, jConnect émet une **SQLException** et l'appel ne renvoie aucun objet **PreparedStatement**. Cette méthode produit donc un résultat tout à fait différent de la méthode **Connection.prepareStatement()**, qui intercepte les erreurs SQL et qui, en cas d'erreur, renvoie un objet **PreparedStatement** contenant une instruction non compilée.

Performances du curseur

Lorsque vous utilisez la méthode **Statement.setCursorName()** ou **setFetchSize()** dans la classe **SybCursorResultSet**, jConnect crée un curseur dans la base de données. D'autres méthodes conduisent jConnect à ouvrir, extraire et mettre à jour un curseur.

Dans les versions de jConnect inférieures à la 4.0, vous pouvez uniquement créer et manipuler des curseurs en envoyant à la base de données des instructions SQL dotées de commandes de curseur explicites, en vue de l'analyse et de la compilation.

Dans jConnect version 4.0 ou supérieure, vous pouvez soit envoyer des instructions SQL à la base de données, soit coder les commandes de curseur sous la forme de jetons au sein du protocole de communication TDS. Les curseurs du premier type sont des curseurs de langue : ceux du second type appartiennent à la catégorie des curseurs de protocoles.

Les curseurs de protocole offrent de meilleures performances que les curseurs de langue. En outre, certaines bases de données ne supportent pas les curseurs de langue, comme c'est le cas des bases Adaptive Server Anywhere.

Dans jConnect, par défaut, tous les curseurs sont de type protocole. Toutefois, la propriété de connexion **LANGUAGE_CURSOR** vous permet de donner la préférence aux protocoles langue.

Propriété de connexion **LANGUAGE_CURSOR**

LANGUAGE_CURSOR est une propriété de connexion à valeur booléenne qui vous permet de déterminer le type des protocoles créés (langue ou protocole :

- Lorsque la propriété **LANGUAGE_CURSOR** a la valeur "false", tous les curseurs créés lors d'une même session sont de type protocole, ce qui optimise les performances : pour créer et manipuler des curseurs, jConnect envoie des commandes de curseur sous la forme de jetons au protocole TDS.

Par défaut, **LANGUAGE_CURSOR** a la valeur "false".

- Lorsque la propriété **LANGUAGE_CURSOR** a la valeur "true", les curseurs créés lors d'une même session sont de type langue : pour créer et manipuler des curseurs, jConnect envoie des instructions SQL à la base de données en vue de l'analyse et de la compilation.

L'activation de la propriété `LANGUAGE_CURSOR` (“true”) ne présente pas d'avantage particulier. Elle peut néanmoins permettre de restaurer le fonctionnement normal d'une application qui présenterait des signes de dysfonctionnement lorsque la propriété est désactivée (“false”).

Migration des applications jConnect

Ce chapitre explique comment faire migrer des applications qui utilisent des extensions Sybase des versions 4.0 ou inférieures de jConnect aux versions 4.1 et supérieures.

Les sujets suivants sont traités :

Nom	Page
Migration des applications vers jConnect 4.1	128
Migration des applications vers jConnect 5.x	128
Migration des applications vers jConnect 4.2 et 5.2	128
Modifications apportées aux extensions Sybase	131

Migration des applications jConnect

Migration des applications vers jConnect 4.1

jConnect 4.1 offre une compatibilité descendante avec les versions précédentes de jConnect. Toutes les applications existantes devraient continuer à fonctionner sans recompilation.

Pour développer de nouvelles applications qui utilisent des extensions Sybase, utilisez les interfaces situées dans **com.sybase.jdbcx**.

Cette interface commune vous permet d'effectuer la mise à niveau des applications vers jConnect version 4.1 et supérieures avec un minimum de changements. Pour connaître les modifications apportées aux extensions Sybase, reportez-vous à la section "[Modifications apportées aux extensions Sybase](#)", page 131.

Migration des applications vers jConnect 5.x

A la différence des versions précédentes de jConnect, le package et les classes destinées au gestionnaire de pilote jConnect 5.x sont désignés par *com.sybase.jdbc2.jdbc.SybDriver*. Il convient de changer le code source de chargement de SybDriver des applications et de le recompiler avec la plate-forme Java™ 2.

Vous trouverez ci-après un exemple de chargement de pilote sous jConnect 5.0 :

```
Driver d =  
(Driver)Class.forName("com.sybase.jdbc2.jdbc.SybDriver").newInstance();  
DriverManager.registerDriver(d);
```

Si vos applications utilisent des extensions Sybase de l'API JDBC, remplacez **com.sybase.jdbc** et/ou **com.sybase.utils** par **com.sybase.jdbcx**. Pour plus d'informations sur les modifications apportées aux extensions Sybase, reportez-vous à la section "[Modifications apportées aux extensions Sybase](#)", page 131.

Pour plus d'informations sur l'utilisation des extensions Sybase, reportez-vous aux exemples fournis avec jConnect.

Migration des applications vers jConnect 4.2 et 5.2

Si vous effectuez une mise à niveau vers jConnect 4.2 ou 5.2, reportez-vous au tableau suivant qui indique les chemins à modifier et à quel endroit vous devez recompiler le code source.

Légende :

- A** Il est conseillé de modifier le chemin du package **com.sybase.jdbcx**
- B** Modifier la variable CLASSPATH en fonction de la nouvelle structure d'installation
- C** Recompiler pour utiliser le nouveau pilote jConnect 5.x

Reportez-vous aux paragraphes suivants pour plus de détails.

Mise à niveau à partir de	vers jConnect version			
jConnect version	4.1	4.2	5.0	5.2
4.0 & inférieures	A	AB	AC	ABC
4.1	-	AB	AC	ABC
4.2	-	-	chemin non supporté	AC
5.0	-	-	-	B

❖ A. Utilisation des nouvelles extensions Sybase.

- 1 Remplacez le chemin d'importations du package

```
import com.sybase.jdbc.*

par

import com.sybase.jdbcx.*;
```

- 2 Utilisez les nouvelles API d'extensions Sybase. Reportez-vous à la section [“Modifications apportées aux extensions Sybase”](#), page 131.

❖ B. Modifiez CLASSPATH en fonction de la nouvelle structure d'installation newJDBC_HOME.

Définissez JDBC_HOME comme répertoire principal du pilote jConnect installé.

Par exemple :

- Pour jConnect 4,2 :
JDBC_HOME=jConnect-4_2
- Pour jConnect 5.0 :

`JDBC_HOME=<répertoire d'installation jConnect>`

Pour plus d'informations sur la définition de JDBC_HOME, reportez-vous à la section "Définition des variable d'environnement", dans le chapitre 1 du document *jConnect for JDBC - Guide d'installation*.

Changement de		CLASSPATH inclut
version		
De	4.1	<i>JDBC_HOME/classes</i>
Vers	5.2	<i>JDBC_HOME/jconn2.jar</i>
De	4.1	<i>JDBC_HOME/classes</i>
Vers	4.2	<i>JDBC_HOME/classes</i>
De	5.0	<i>JDBC_HOME/classes/jconn2.jar</i>
Vers	5.2	<i>JDBC_HOME/classes/jconn2.jar</i>

❖ **C. Recompilez pour pouvoir utiliser le nouveau pilote jConnect 5.x**

- Modifiez dans le code source l'emplacement à partir duquel le pilote est chargé et remplacez :

```
Class.forName("com.sybase.jdbc.SybDriver");
```

par

```
Class.forName("com.sybase.jdbc2.jdbc.SybDriver");
```


Modifications apportées aux extensions Sybase

Les versions C.1, 4.2 et 5.x de jConnect incluent désormais un nouveau package, **com.sybase.jdbcx**, qui contient toutes les extensions Sybase pour JDBC. Dans les versions précédentes de jConnect, ces extensions étaient intégrées aux packages **com.sybase.jdbc** et **com.sybase.utils**.

com.sybase.jdbcx constitue une interface cohérente pour les différentes versions de jConnect. Toutes les extensions Sybase sont définies en tant qu'interfaces Java, ce qui permet de changer des mises en oeuvre sous-jacentes indépendamment des applications créées à l'aide des applications de ces interfaces.

Pour développer de nouvelles applications qui utilisent des extensions Sybase, servez-vous de **com.sybase.jdbcx**. Les interfaces de ce package vous permettent d'effectuer la mise à niveau des applications à niveau vers les versions jConnect supérieures à la 4.0, avec un minimum de changements.

Remarque Les applications précédemment créées à l'aide des extensions Sybase vers l'API JDBC et qui étaient disponibles dans **com.sybase.jdbc** et **com.sybase.utils** sont supportées par la version 4.x de jConnect ; cependant, toutes les extensions Sybase de **com.sybase.jdbc** et **com.sybase.utils** ont été marquée obsolètes.

Certaines extensions Sybase ont été modifiées en fonction de la nouvelle interface **com.sybase.jdbcx**.

Exemple

Si une application utilise **SybMessageHandler**, les différences de code seront les suivantes :

- code **jConnect 4.0** :

```
import com.sybase.jdbc.SybConnection;
import com.sybase.jdbc.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setMessageHandler(new ConnectionMsgHandler());
```

- code **jConnect 4.1** et versions supérieures :

```
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setSybMessageHandler(new ConnectionMsgHandler());
```

Pour plus d'informations sur l'utilisation des extensions Sybase, reportez-vous aux exemples fournis avec jConnect.

Noms de méthodes modifiés

Le tableau suivant répertorie les nouveaux noms de méthodes dans la nouvelle interface :

Classe	Ancien nom	Nouveau nom
SybConnection	getCapture()	createCapture()
SybConnection	setMessageHandler()	setSybMessageHandler()
SybConnection	getMessageHandler()	getSybMessageHandler()
SybStatement	setMessageHandler()	setSybMessageHandler()
SybStatement	getMessageHandler()	getSybMessageHandler()

Classe Debug

Les références statiques directes à la classe **Debug** ne sont plus supportées, mais elles existent sous forme obsolète dans le package **com.sybase.utils**. Pour utiliser les fonctions de débogage jConnect, utilisez la méthode **getDebug()** de la classe **SybDriver** afin d'obtenir une référence à la classe **Debug**. Par exemple :

```
import com.sybase.jdbcx.SybDriver;
import com.sybase.jdbcx.Debug;
.
.
.
SybDriver sybDriver =
    SybDriver.Class.forName
        ("com.sybase.jdbc2.jdbc.SybDriver") newInstance();
Debug sybDebug = sybDriver.getDebug();
sybDebug.debug(true, "ALL", System.out);
```

Pour obtenir une liste complète des extensions Sybase, reportez-vous à la documentation javadoc de jConnect, située dans le répertoire *docs/* du répertoire d'installation jConnect.

Le présent chapitre décrit les passerelles de serveur Web et explique comment les utiliser avec jConnect.

Il se compose des sections suivantes :

Nom	Page
Introduction	136
Utilisation de la passerelle Cascade	139
Utilisation du servlet d'encapsulation par TDS	145

Introduction

L'utilisation d'une passerelle en tant que proxy permet de préciser le chemin d'accès à un serveur de bases de données. Elle s'impose si la base de données et le serveur Web sont hébergés sur des hôtes différents, ou si vous développez des applications Internet qui, pour accéder à un serveur de bases de données sécurisé, doivent passer par un firewall.

Pour permettre la connexion à des serveurs à l'aide du protocole SSL (Secure Sockets Layer), jConnect fournit un servlet Java que vous pouvez installer sur tout serveur Web prenant en charge les interfaces **javax.servlet**. Ce servlet permet à jConnect de supporter le cryptage en utilisant le serveur Web comme passerelle.

Remarque jConnect inclut également le protocole SSL sur le système client. Pour plus d'informations à ce sujet, reportez-vous à la section [“Mise en oeuvre des plug-ins de sockets personnalisés”](#), page 27.

jConnect comprend une passerelle Cascade, version légèrement modifiée du serveur Cascade Web. Cette passerelle supporte la méthode CONNECT HTTP, qui permet l'encapsulation de données TDS (Tabular Data Stream) par HTTP. Vous pouvez utiliser la passerelle Cascade pour vous connecter à un serveur de bases de données qui est hébergé sur un hôte différent de celui du serveur Web, sans passer par un firewall. Reportez-vous à la section [“Utilisation de la passerelle Cascade”](#), page 139.

Encapsulation par TDS

jConnect utilise TDS pour communiquer avec les serveurs de base de données. L'encapsulation de données TDS par HTTP est utile pour renvoyer des requêtes. Le flux TDS émis d'un client vers un serveur principal via la passerelle est intégré dans le corps de la requête. L'en-tête de la requête indique la longueur du flux TDS qui est inclus dans le paquet de requête.

A l'inverse de HTTP, TDS est un protocole orienté connexion. Afin de supporter des fonctionnalités de sécurité telles que le cryptage pour applications Internet, jConnect utilise un servlet d'encapsulation par TDS qui lui permet de maintenir une connexion logique sur l'ensemble des requêtes HTTP. Le servlet génère un ID de session lors de la demande de connexion initiale. Cet ID est ensuite inclus dans l'en-tête de chaque requête suivante. L'utilisation d'ID de session permet d'identifier les sessions actives, ainsi que de reprendre une session (tant que le servlet dispose d'une connexion ouverte).

La connexion logique fournie par le servlet d'encapsulation par TDS permet à jConnect de supporter une communication cryptée entre deux systèmes (par exemple, un client jConnect pour lequel la propriété de connexion `CONNECT_PROTOCOL` a la valeur "https" qui se connecte à un serveur Web sur lequel est exécuté un servlet d'encapsulation par TDS.

CONFIGURATION DE JCONNECT ET DE PASSERELLES

Il existe plusieurs façons de configurer les serveurs Web et Adaptive Server. Vous trouverez ci-dessous une description de quatre configurations courantes. Reportez-vous aux exemples suivants pour savoir où installer le pilote jConnect et quand utiliser la passerelle Cascade ou une passerelle utilisant le servlet d'encapsulation par TDS.

Serveur Web et Adaptive Server sur un même hôte

Dans cette configuration à deux niveaux, les serveurs Web et Adaptive Server sont tous deux installés sur le même hôte.

- Installez jConnect sur l'hôte du serveur Web.
- Aucune passerelle n'est requise.

Serveur Web JDBC dédié et Adaptive Server sur un même hôte

Dans cette configuration, le serveur Web principal est hébergé sur un hôte distinct. Un serveur Web (destiné à l'accès à Adaptive Server) et l'Adaptive Server partagent un second hôte. Des liens provenant du serveur principal acheminent au serveur Web dédié les requêtes nécessitant un accès à SQL.

- Installez jConnect sur le second hôte (Adaptive Server).
- Aucune passerelle n'est requise.

Serveur Web et Adaptive Server sur des hôtes distincts

Dans cette configuration à trois niveaux, l'Adaptive Server et le serveur Web résident sur des hôtes différents. jConnect requiert une passerelle qu'il utilisera en tant que proxy pour l'Adaptive Server.

- Installez jConnect sur l'hôte du serveur Web.
- Ceci requiert soit un servlet d'encapsulation par TDS, soit une passerelle Cascade.

Connexion à un serveur via un firewall

Pour établir une connexion à un serveur protégé par firewall, vous devez utiliser un serveur Web conjointement au servlet d'encapsulation par TDS ; cela permet d'assurer la transmission par Internet des réponses aux requêtes de base de données.

- Installez jConnect sur l'hôte du serveur Web.
- Ceci requiert un serveur Web qui supporte les interfaces **javax.servlet**.

Utilisation de la passerelle Cascade

L'utilisation d'une passerelle en tant que proxy permet de préciser le chemin d'accès à un serveur de bases de données. Vous devez y avoir recours si la base de données et le serveur Web résident sur des hôtes différents, même s'il n'existe pas de firewall.

jConnect comprend une passerelle Cascade, version légèrement modifiée du serveur Cascade Web écrit en Java par David Wilkerson (e-mail : davidw@cascade.org.uk ; site Web : <http://www.cascade.org.uk/>).

Cette passerelle reçoit et transmet des paquets, passe du protocole HTTP à TDS lors de l'envoi d'une requête à l'Adaptive Server et de TDS à HTTP, lors du renvoi de la réponse.

Remarque La passerelle Cascade ne prend pas en charge le cryptage. Elle ne convient donc pas aux applications Internet qui se connectent à un serveur principal via un firewall.

Conditions d'utilisation requis

- Si vous n'avez pas installé jConnect dans le répertoire d'installation par défaut, vous devez remplacer, dans *www.dos* (DOS) ou *www.template* (UNIX), toutes les références au répertoire par défaut par le chemin d'installation complet de jConnect.
- La passerelle Cascade et votre serveur Web doivent s'exécuter sur la même machine hôte. Cela permet aux applets de se connecter au même hôte que le serveur Web, mais au port contrôlé par la passerelle Cascade. La passerelle achemine la requête vers la base de données appropriée. Pour plus d'informations sur ce processus, examinez le code source dans les fichiers *Isql.java* et *gateway.html* situés dans le sous-répertoire *sample* (jConnect 4.x) ou *sample2* (jConnect 5.x) du répertoire d'installation jConnect. Recherchez le terme "proxy".

Si votre serveur de bases de données réside sur le même hôte que votre serveur Web, l'utilisation de la passerelle Cascade n'est pas nécessaire.

Installation de la passerelle Cascade

La passerelle Cascade est installée lorsque vous effectuez une installation complète de jConnect, soit à l'aide du programme d'installation de jConnect, soit à partir des fichiers *install.bat* ou *install.sh*. Au besoin, vous pouvez également utiliser le programme d'installation de jConnect pour n'installer que les fichiers de la passerelle Cascade. Reportez-vous aux documents *Guide d'installation* et *Notes de mise à jour*.

Démarrage de la passerelle Cascade

Pour tester la passerelle Cascade, suivez les instructions ci-dessous relatives à votre plate-forme.

Windows NT et Windows 95

- 1 A l'invite DOS, placez-vous dans le répertoire d'installation jConnect.
Définissez ce répertoire dans la variable d'environnement JDBC_HOME.

- 2 Démarrez la passerelle en tapant :

```
httpd
```

Si la commande aboutit, l'instruction suivante s'affiche :

```
HTTPDServer www.dos
```

UNIX

Placez-vous dans le répertoire d'installation de jConnect (défini par JDBC_HOME). Tapez la commande suivante :

```
sh httpd.sh &
```

Résolution des incidents

- Si aucun message n'apparaît après avoir entré la commande **httpd**, cela signifie que le serveur ne fonctionne pas. Relancez la commande en mode détaillé.

Sous Windows, placez-vous après l'invite DOS et entrez :

```
httpd -Dverbose=1 > nom_fichier
```

Sous UNIX, entrez :

```
sh httpd.sh -Dverbose=1 > nom_fichier &
```

où *nom_fichier* est le fichier de résultats des messages de débogage.

- Si le message d'erreur suivant s'affiche :

```
HTTPServer: IOException: getRequest() Address  
already in use
```

Cette erreur indique qu'un autre processus est en cours sur le port spécifié dans le fichier *www.dos* (DOS) ou *www.template* (UNIX), situé dans le répertoire JDBC_HOME. Elle se produit au démarrage de la passerelle.

Vous pouvez :

- Arrêter le processus en cours sur le port spécifié. Après vous être assuré que le processus a pris fin, essayez de relancer la passerelle.

ou
- Modifier le numéro de port dans le fichier *www.dos* ou *www.template*, puis remplacer dans le fichier *gateway.html*, situé dans le sous-répertoire *sample* (jConnect 4.x) ou *sample2* (jConnect 5.x) du répertoire JDBC_HOME, le paramètre *proxy* par "*localhost:nouveau_port*".

Si votre machine hôte n'est pas le "*localhost*" (c'est le cas si le serveur Cascade HTTP et l'explorateur se trouvent sur des hôtes différents, vérifiez que le paramètre *proxy* utilise le nom de l'hôte au lieu de "*localhost*".

Test de la passerelle Cascade

Pour vérifier la configuration et tester la passerelle Cascade, vous pouvez exécuter un programme de validation qui établit une connexion à la base de données de démonstration Sybase.

Remarque Ce programme utilise "*localhost:8000*" pour tester votre passerelle.

A partir de l'invite DOS (Windows NT ou Windows 95) ou d'une invite UNIX, placez-vous dans le répertoire JDBC_HOME.

Pour jConnect 4.x, tapez :

```
java sample.SybSample Validate
```

Pour jConnect 5.x, tapez :

```
java sample2.SybSample Validate
```

Si la validation aboutit, le numéro de version de jConnect apparaît et un message de connexion s'affiche dans la fenêtre de résultats.

Résolution des incidents

Si le message d'erreur "Commande ou nom de fichier incorrect" (Windows 95) ou "Le nom spécifié n'est pas reconnu comme étant une commande interne ou externe" (Windows NT) apparaît, vérifiez que le chemin comprend le sous-répertoire *\bin* dans le répertoire d'installation de JDK.

Examen du fichier *index.html*

Utilisez votre explorateur Web pour afficher le fichier *index.html* situé dans le répertoire d'installation de jConnect. Ce fichier *index.html* contient des liens à la documentation et au code exemple de jConnect.

Remarque Si vous utilisez sur la même machine que celle sur laquelle jConnect est installé, vérifiez que l'explorateur n'a pas accès à votre variable d'environnement CLASSPATH. Reportez-vous à la section "Restrictions relatives à la définition de CLASSPATH lors de l'utilisation de Netscape" dans le chapitre 1 du document *Sybase jConnect for JDBC - Guide d'installation*.

- 1 Ouvrez l'explorateur Web.
- 2 Entrez l'URL correspondant à votre configuration. Par exemple, si l'explorateur et la passerelle Cascade sont hébergés sur le même hôte, entrez :

```
http://localhost:8000/index.html
```

Si l'explorateur et la passerelle Cascade résident sur des hôtes différents, entrez :

```
http://hôte:port/index.html
```

où *hôte* est le nom de l'hôte sur lequel réside la passerelle Cascade *port* est le port récepteur.

Résolution des incidents

Si après avoir entré le nom d'hôte, le numéro de port et le nom de fichier corrects, l'explorateur ne parvient pas à ouvrir ce lien, c'est que la passerelle Cascade n'est pas active. Reportez-vous à la section "[Démarriage de la passerelle Cascade](#)", page 140.

Exécution de l'applet Isql exemple

Après avoir chargé le fichier *index.html* dans l'explorateur :

- 1 Cliquez sur "Run Sample JDBC Applets" (Exécuter des applets JDBC exemple).

Vous accédez à la page de programmes exemple de jConnect.

- 2 Faites défiler cette page vers le bas jusqu'à la mention "Executable Samples" (Programmes exécutables).
- 3 Recherchez "Isql.java" dans le tableau et cliquez sur "Run" (Exécuter) à l'extrémité de la ligne.

L'applet **Isql.java** vous invite à adresser une requête dans une base exemple et affiche le résultat. L'applet indique un nom d'hôte Adaptive Server, un numéro de port, un nom d'utilisateur (*guest*), un mot de passe (*sybase*), une base de données et une requête par défaut. Elle établit alors la connexion à la base de données de démonstration Sybase et renvoie le résultat lorsque vous cliquez sur Go.

Résolution des incidents

Sous UNIX, si l'aspect de l'applet est inattendu, modifiez les dimensions de l'écran comme suit :

- 1 Utilisez un éditeur de texte pour ouvrir le fichier suivant :

Pour jConnect 4.x

\$JDBC_HOME/sample/gateway.html

Pour jConnect 5.x

\$JDBC_HOME/sample/gateway.html

- 2 Modifiez la valeur du paramètre de hauteur (ligne 7 à 650). Essayez différentes valeurs.
- 3 Rechargez la page Web dans l'explorateur.

Définition d'une connexion à la passerelle Cascade

Pour définir une connexion dans l'application qui utilise la passerelle Cascade, indiquez dans l'URL le nom de l'hôte qui héberge cette passerelle :

hôte:port

où *hôte* est le nom de l'hôte sur lequel réside la passerelle Cascade et *port* est le port récepteur.

Utilisation du servlet d'encapsulation par TDS

Pour pouvoir utiliser le servlet d'encapsulation par TDS, vous devez installer un serveur Web qui supporte les interfaces **javax.servlet**, tel que Java™ Web de Sun Microsystems, Inc. puis ajouter le servlet jConnect dans la liste des servlets actifs. Vous pouvez également définir des paramètres pour que le servlet définisse des délais de déconnexion et une taille de paquet minimale.

Lorsque vous utilisez le servlet d'encapsulation par TDS, les requêtes adressées par un client au serveur principal via la passerelle, comprennent une commande GET ou POST, l'ID de session TDS (après la requête initiale), l'adresse du serveur principal et l'état de la requête.

Le flux TDS est intégré dans le corps de la requête. L'en-tête de la requête comprend deux champs, l'un destiné à la longueur du flux TDS, l'autre réservé à l'ID de session attribué par la passerelle.

Lorsque le client envoie une requête, le champ Content-Length indique la taille du flux TDS, et la commande de requête est POST. Si la requête ne contient aucune donnée TDS, soit parce que le client récupère la portion suivante de la réponse du serveur, soit parce qu'il ferme la connexion, la commande de requête devient GET.

L'exemple suivant montre comment les informations sont transmises entre le client et une passerelle HTTPS à l'aide du protocole HTTPS encapsulé par TDS ; il illustre une connexion à un serveur principal "serveurbd" doté du numéro de port "1234".

Tableau 6-1 : Demande de connexion adressée du client vers la passerelle. Aucun ID de session.

<i>Requête</i>	POST/tds?ServerHost=serveurbd&ServerPort=1234&Operation=more HTTP/1.0
<i>En-tête</i>	Content-Length : 605
<i>Contenu (TDS)</i>	Demande de connexion

Tableau 6-2 : De la passerelle au client. L'en-tête contient un ID de session attribué par le servlet TDS.

<i>Requête</i>	200 SUCCESS HTTP/1.0
<i>En-tête</i>	Content-Length : 210 TDS-Session: TDS00245817298274292
<i>Contenu (TDS)</i>	Accusé de connexion EED

Tableau 6-3 : Du client à la passerelle. L'en-tête des requêtes suivantes contient l'ID de session.

<i>Requête</i>	POST/tds?TDS-Session=TDS00245817298274292&Operation=more HTTP/1.0
<i>En-tête</i>	Content-Length : 32
<i>Contenu (TDS)</i>	Requête 'SELECT * from authors'

Tableau 6-4 : De la passerelle au client. L'en-tête des réponses suivantes contient l'ID de session.

<i>Requête</i>	200 SUCCESS HTTP/1.0
<i>En-tête</i>	Content-Length : 2048 TDS-Session: TDS00245817298274292
<i>Contenu (TDS)</i>	Format des lignes et certaines lignes issues de la réponse à la requête.

Configuration système requise pour le servlet d'encapsulation par TDS

Pour pouvoir utiliser le servlet jConnect pour le protocole HTTP encapsulé par TDS, vous devez disposer des éléments suivants :

- Un serveur Web qui supporte les interfaces **javax.servlet**. Pour l'installer, suivez les instructions qui l'accompagnent.
- Un explorateur Web qui supporte JDK 1.1, tel que Netscape 4.0, Internet Explorer 4.0 ou HotJava.

Installation du servlet

Une fois installé, jConnect comprend un sous-répertoire *gateway* (jConnect 4.x) ou *gateway2* (jConnect 5.x) dans le répertoire *classes*. Ce sous-répertoire contient les fichiers nécessaires au servlet d'encapsulation par TDS.

Copiez le package **gateway** de jConnect dans un sous-répertoire *gateway* (jConnect 4.x) ou *gateway2* (jConnect 5.x) du répertoire de servlets de votre serveur Web, puis suivez les instructions relatives à votre serveur Web pour activer les servlets.

Paramétrage des arguments de servlet

Lorsque vous ajoutez le servlet dans votre serveur Web, vous pouvez entrer des arguments de servlet facultatifs afin de personnaliser les performances de ce dernier :

- *SkipDoneProc* [*true/false*] : les bases de données Sybase renvoient souvent des informations de décompte de lignes en cours d'exécution d'une requête. Les applications clientes ignorent généralement ces données. Si vous affectez la valeur "true" à *SkipDoneProc*, le servlet supprime ces informations des réponses "à la volée", ce qui réduit l'utilisation du réseau et les traitements requis sur le client. Cet argument s'avère particulièrement utile avec HTTPS/SSL, car il évite le cryptage et le décryptage des données indésirables avant que ces dernières ne soient ignorées.
- *TdsResponseSize* : définit la taille de paquet TDS maximale pour le protocole HTTPS encapsulé. Pour optimiser les performances, la valeur de cet argument doit être fonction du nombre d'utilisateurs et du volume de données échangées : augmentez-la si seuls quelques utilisateurs échangent un grand volume de données, et réduisez-la si un nombre important d'utilisateurs effectuent de plus petites transactions.
- *TdsSessionIdleTimeout* : définit le délai (en millisecondes) d'inactivité d'une connexion serveur au terme duquel la connexion est fermée automatiquement. La valeur par défaut de *TdsSessionIdleTimeout* est 600 000 (10 minutes).

Si vous disposez de programmes clients interactifs susceptibles de rester inactifs pendant des périodes prolongées et que vous ne voulez pas que les connexions soient interrompues, augmentez la valeur de *TdsSessionIdleTimeout*.

Vous pouvez également définir le délai de déconnexion à partir du client jConnect en utilisant la propriété de connexion `SESSION_TIMEOUT`, ce notamment pour les applications susceptibles de rester inactives pendant des périodes prolongées. Dans ce cas, il est préférable d'augmenter le temps imparti au niveau de la connexion plutôt qu'à celui du servlet.

- *Debug* : active le débogage. Reportez-vous à la section "[Débogage avec jConnect](#)", page 100.

Entrez arguments de servlet dans une chaîne, en les séparant par une virgule. Par exemple :

```
TdsResponseSize=[taille],TdsSessionIdleTimeout=[délai],Debug=true
```

Pour plus d'informations sur l'entrée d'arguments de servlet, reportez-vous à la documentation qui accompagne votre serveur Web.

Appel du servlet

Pour déterminer à quel moment utiliser la passerelle où le servlet d'encapsulation par TDS est installé, jConnect s'appuie sur l'extension de chemin associée à la propriété de connexion *proxy*. jConnect reconnaît l'extension du chemin servlet au *proxy* et appelle le servlet sur la passerelle désignée.

Pour définir l'URL de connexion, utilisez le format ci-dessous :

```
http://hôte:port/chemin_servlet_TDS
```

jConnect appelle le servlet d'encapsulation par TDS sur le serveur Web afin d'encapsuler TDS par HTTP. Le chemin du servlet doit correspondre à celui défini dans la liste des alias de servlet sur votre serveur Web.

Suivi des sessions TDS actives

Vous pouvez afficher des informations relatives aux sessions TDS, y compris les connexions serveur pour chaque session. Pour ouvrir l'URL administrative, utilisez votre explorateur Web :

```
http://hôte:port/chemin_servlet_TDS?Operation=list
```

Par exemple, si votre serveur se nomme MONSERVEUR et que le chemin au servlet TDS est */tds*, entrez :

```
http://monserveur:8080/tds?Operation=list
```

Cela permet d'afficher une liste de sessions TDS actives. Pour obtenir plus d'informations sur une session (y compris sur la connexion serveur), cliquez dessus.

Interruption d'une session TDS

Pour interrompre toute session TDS active, vous pouvez utiliser l'URL décrit ci-dessus. Dans la liste de sessions de la première page, cliquez sur une session active, puis choisissez *Terminate This Session* (Interrompre la session).

Reprise d'une session TDS

Vous pouvez définir la propriété de connexion `SESSION_ID` afin d'autoriser la reprise d'une connexion ouverte existante, si nécessaire. Lorsque vous définissez un `SESSION_ID`, `jdbcConnect` ignore la phase de connexion du protocole et reprend la connexion à la passerelle en utilisant l'ID de session désigné. Si celui-ci n'existe pas sur le servlet, `jdbcConnect` envoie une exception `SQLException` lorsque vous essayez d'utiliser la connexion pour la première fois.

Encapsulation TDS et Netscape Enterprise Server 3.5.1 sous Solaris

Netscape Enterprise Server 3.5.1 ne supporte pas les méthodes `javax.servlet.ServletConfig.getInitParameters()` ou `javax.servlet.ServletConfig.getInitParameterNames()`. Pour fournir les valeurs de paramètre nécessaires, vous devez remplacer les appels des méthodes `getInitParameter()` et `getInitParameterNames()` par des valeurs de paramètre codées en dur dans `TDSTunnelServlet.java`.

Pour entrer les valeurs de paramètre nécessaires dans `TDSTunnelServlet.java` et utiliser l'encapsulatoir par TDS avec Netscape Enterprise Server 3.5.1 sous Solaris, procédez comme suit :

- 1 Codez en dur les valeurs de paramètre dans `TDSTunnelServlet.java`.
- 2 Créez les fichiers `.class` à partir des déclarations de classe dans `TDSTunnelServlet.java`.

Les fichiers suivants sont créés :

- `TDSTunnelServlet.class`
- `TdsSession.class`
- `TdsSessionManager.class`

- 3 Créez le répertoire de destination des fichiers `.class` dans le répertoire d'installation de Netscape Enterprise Server 3.5.1 (NSE_3.5.1). Pour ce faire, procédez comme suit :

```
mkdir rép_install_NSE_3.5.1/plugins/java/servlets/gateway
```

- 4 Copiez les fichiers `.class` dérivés de `TDSTunnelServlet.java` dans le répertoire que vous avez créé à l'étape précédente.
- 5 Copiez les classes du répertoire `$JDBC_HOME/classes/com/sybase` vers `rép_install_NSE_3.5.1/docs/com/sybase`.

Pour ce faire, vous pouvez copier de manière récursive tous les fichiers du répertoire `$JDBC_HOME/classes` dans le répertoire `rép_install_NSE_3.5.1/docs`, comme suit :

```
cp -r $JDBC_HOME/classes rép_install_NSE_3.5.1/docs
```

Un certain nombre de fichiers et de répertoires qui n'appartiennent pas au répertoire `$JDBC_HOME/classes/com/sybase` sont copiés. Ces fichiers et répertoires supplémentaires sont anodins, mais occupent de l'espace disque. Supprimez-les.

- 6 Indiquez l'URL du servlet d'encapsulation par TDS dans le paramètre d'URL *proxy*.

Par exemple, dans `$JDBC_HOME/sample/gateway.html`, modifiez le paramètre *proxy* comme suit :

```
<param name=proxy value="http://nom_hôte/servlet/  
nom_passerelle.nom_servlet_encaps_TDS">
```

Messages d'exception et d'avertissement SQL

Le tableau ci-dessous répertorie les messages d'exception et d'avertissement que vous êtes susceptible de rencontrer lors de l'utilisation de jConnect.

Etat SQL	Message/Description/Action
010DP	<p>La propriété de connexion dupliquée a été ignorée.</p> <p><i>Description :</i> Une propriété de connexion a été définie deux fois. Elle peut apparaître en double dans la liste des propriétés de connexion d'un pilote, la première fois en minuscules ("mot_de_passe") et la seconde fois en majuscules ("MOT_DE_PASSE), par exemple. Or, aucune distinction de casse n'est effectuée dans les propriétés de connexion. Pour jConnect, ces propriétés portent donc le même nom.</p> <p>Il peut également arriver que la propriété de connexion soit définie à la fois dans la liste des propriétés de connexion et dans l'URL. Dans ce cas, c'est la valeur définie dans la liste des propriétés de connexion qui prévaut.</p> <p><i>Action :</i> Vérifiez que l'application ne définit la propriété de connexion qu'une seule fois. Néanmoins, si la propriété est définie à la fois dans la liste des propriétés de connexion et dans l'URL et que ce qui en découle vous convient, vous pouvez ignorer ce message d'avertissement.</p>
010HA	<p>La demande de session HD a été rejetée par le serveur. Veuillez reconfigurer la base de données ou travailler hors session HD.</p> <p><i>Description :</i> La propriété de connexion REQUEST_HA_SESSION n'a pas la valeur "true" et le serveur auquel jConnect a tenté de se connecter n'a pas autorisé la connexion.</p> <p><i>Action :</i> Reconfigurez le serveur afin qu'il supporte le mode reprise sur le serveur secondaire, ou n'activez pas ce mode et dans ce cas ne définissez pas la propriété REQUEST_HA_SESSION à "true".</p>
010HD	<p>Le mode de reprise sur le serveur secondaire HD de Sybase n'est pas supporté par ce type de serveur de base de données.</p> <p><i>Description :</i> La base de données à laquelle jConnect a tenté de se connecter ne supporte pas le mode reprise sur le serveur secondaire.</p> <p><i>Action :</i> Connectez-vous uniquement à des serveurs de bases de données qui gèrent le mode reprise sur le serveur secondaire.</p>
010MX	<p>Les informations d'accès aux métadonnées sont introuvables dans cette base de données. Please install the required tables as mentioned in the jConnect documentation. L'erreur suivante a été détectée lors de la tentative d'extraction de métadonnées : _____</p> <p><i>Description :</i> Il se peut que le serveur ne dispose pas des procédures stockées nécessaires au renvoi de métadonnées.</p> <p><i>Action :</i> Vérifiez que les procédures stockées correspondantes sont installées sur le serveur. Reportez-vous à la section "Installation de procédures stockées" dans le chapitre 1 du document <i>jConnect for JDBC - Guide d'installation</i>.</p>

Etat SQL	Message/Description/Action
010P4	<p>Un paramètre de sortie reçu a été ignoré.</p> <p><i>Description :</i> La requête exécutée a renvoyé un paramètre de sortie, mais celui-ci n'a pas été pris en compte dans le code de traitement du résultat de l'application.</p> <p><i>Action :</i> Si votre application requiert les données de ce paramètre de sortie, vous devez réécrire l'application en conséquence. Cela peut impliquer l'utilisation d'un CallableStatement pour exécuter la requête et l'ajout d'appels dans les paramètres registerOutputParameter() et getXXX().</p>
010PF	<p>Un ou plusieurs fichiers jar spécifiés dans la propriété de connexion PRELOAD_JARS n'ont pas pu être chargés.</p> <p><i>Description :</i> Cette erreur se produit lorsque vous utilisez DynamicClassLoader avec la propriété de connexion PRELOAD_JARS contenant une liste de noms de fichiers JAR séparés par une virgule. Lorsque DynamicClassLoader ouvre sa connexion au serveur à partir duquel les classes sont chargées, il tente de précharger tous les fichiers JAR mentionnés dans la propriété de connexion. Si un ou plusieurs noms de fichiers JAR n'existent pas sur le serveur, vous obtenez le message d'erreur ci-dessus.</p> <p><i>Action :</i> Vérifiez que chaque fichier JAR mentionné dans la propriété de connexion PRELOAD_JARS de votre application existe bien et qu'il est accessible sur le serveur.</p>
010RC	<p>La synchronisation et le type de ResultSet demandés ne sont pas supportés. Ils ont été convertis.</p> <p><i>Description :</i> Vous avez demandé pour ResultSet une combinaison synchronisation/type qui n'est pas supportée. Les valeurs demandées ont dû être converties.</p> <p><i>Action :</i> Choisissez pour ResultSet une combinaison synchronisation/type qui est supportée.</p>
010SJ	<p>Les informations d'accès aux métadonnées sont introuvables dans cette base de données. Veuillez installer les tables requises comme indiqué dans la documentation jConnect.</p> <p><i>Description :</i> Aucune information relative aux métadonnées n'est configurée sur le serveur.</p> <p><i>Action :</i> Si votre application requiert des métadonnées, installez les procédures stockées nécessaires au renvoi de métadonnées, fournies avec jConnect (reportez-vous à la section "Installation de procédures stockées" dans le chapitre 1 du document <i>jConnect for JDBC - Guide d'installation</i>). Si les métadonnées vous sont inutiles, attribuez la valeur "false" à la propriété USE_METADATA.</p>

Etat SQL	Message/Description/Action
010SK	<p>La base de données ne peut pas définir l'option de connexion.</p> <p><i>Description :</i> Votre application a tenté une opération non supportée par la base de données à laquelle vous êtes connecté.</p> <p><i>Action :</i> Il peut être nécessaire de mettre à niveau votre base de données ou de vérifier que la dernière version de métadonnées y est installée.</p>
010SN	<p>Autorisation en écriture refusée sur ce fichier. Fichier : ____.</p> <p>Message d'erreur : ____</p> <p><i>Description :</i> L'autorisation en écriture sur le fichier définie par la propriété de connexion PROTOCOL_CAPTURE est refusée, en raison d'une violation de sécurité survenue sur le VM. Cela peut se produire si un applet tente d'écrire sur ce fichier.</p> <p><i>Action :</i> Si vous essayez d'écrire sur le fichier à partir d'un applet, vérifiez que ce dernier a accès au système de fichiers cible.</p>
010SP	<p>Ce fichier ne peut pas être ouvert en écriture. Fichier : ____.</p> <p>Message d'erreur : ____</p> <p><i>Action :</i> Vérifiez que le nom du fichier est correct et que ce dernier est accessible en écriture.</p>
010TP	<p>Le jeu de caractères initial, appliqué à la connexion, ____, n'a pas pu être converti par le serveur. Le jeu de caractères proposé par le serveur, ____, va être utilisé ainsi que les conversions effectuées par jConnect.</p> <p><i>Description :</i> Le serveur qui ne peut pas utiliser le jeu de caractères initialement demandé par jConnect, emploie un jeu différent. jConnect accepte la modification et effectuera les conversions nécessaires.</p> <p>Le message est donné à titre indicatif et n'appelle aucune intervention.</p> <p><i>Action :</i> Pour éviter ce message, définissez la propriété de connexion CHARSET selon un jeu de caractères supporté par le serveur.</p>

Etat SQL	Message/Description/Action
010UF	<p>Echec de la tentative d'exécution de la commande <code>use database</code>. Message d'erreur :</p> <p><i>Description</i> : jConnect n'a pas pu établir de connexion à la base de données spécifiée dans l'URL de connexion. Il existe deux causes possibles à cela :</p> <ul style="list-style-type: none"> • Le nom entré dans l'URL est incorrect. • La propriété <code>USE_METADATA</code> a la valeur "true" (par défaut), mais aucune procédure stockée nécessaire au renvoi de métadonnées n'a été installée. Par conséquent, jConnect tente d'exécuter la commande <code>use database</code> sur la base de données indiquée dans l'URL, mais en vain. La base de données à laquelle vous avez voulu accéder peut être de type SQL Anywhere. Or, les bases de ce type ne supportent pas la commande <code>use database</code>. <p><i>Action</i> : Vérifiez le nom de la base de données dans l'URL. Assurez-vous que les procédures stockées nécessaires au renvoi de métadonnées sont installées sur le serveur (reportez-vous à la section "Installation de procédures stockées" dans le chapitre 1 du document <i>jConnect for JDBC - Guide d'installation</i>). Pour accéder à une base de données SQL Anywhere, supprimez son nom de l'URL ou attribuez la valeur "false" à la propriété <code>USE_METADATA</code>.</p>
010UP	<p>La propriété de connexion est inconnue et a été ignorée.</p> <p><i>Description</i> : Vous avez essayé de définir une propriété de connexion dans l'URL, que jConnect ne reconnaît pas. jConnect ignore cette propriété.</p> <p><i>Action</i> : Vérifiez dans la définition de l'URL de votre application que les propriétés de connexion référencées du pilote jConnect sont correctes.</p>
0100V	<p>La version du protocole TDS utilisé est trop ancienne. Version : _____</p> <p><i>Description</i> : Un serveur ne supporte pas la version requise du protocole TDS. jConnect requiert la version 5.0 ou supérieure.</p> <p><i>Action</i> : Utilisez un serveur qui support la version requise de TDS. Pour plus d'informations, reportez-vous à la section relative à la configuration système dans le guide d'installation de jConnect.</p>
JW010	<p>I/O layer : thread operation failed. (Couche E/S : échec lors de l'exécution du thread).</p> <p><i>Description</i> : Une erreur interne s'est produite dans un flux d'E/S synchronisé.</p> <p><i>Action</i> : Fermez, puis rouvrez la connexion.</p>
JZ001	<p>La propriété user name est trop longue. Taille maximale admise : 30.</p> <p><i>Action</i> : La propriété ne doit pas dépasser 30 octets. Corrigez l'erreur.</p>

Etat SQL	Message/Description/Action
JZ002	<p>La propriété password est trop longue. Taille maximale admise : 30.</p> <p><i>Action</i> : La propriété ne doit pas dépasser 30 octets. Corrigez l'erreur.</p>
JZ003	<p>Le format de l'URL est incorrect. URL : _____</p> <p><i>Action</i> : Vérifiez le format URL. Reportez-vous à la section "Valeurs de propriété de connexion dans un URL", page 19.</p> <p>Si le format de la propriété de connexion PROXY utilisée est incorrect, il peut se produire une erreur d'exception de type JZ003.</p> <p>Le format de la propriété PROXY du proxy Cascade est le suivant :</p> <p><i>adresse_ip:numéro_port</i></p> <p>Le format de la propriété PROXY du servlet d'encapsulation par TDS est le suivant :</p> <p><i>http[s]://hôte:port/alias_servlet_encapsulation</i></p>
JZ004	<p>La propriété user name est introuvable dans DriverManager.getConnection(..., Properties).</p> <p><i>Action</i> : Corrigez l'erreur.</p>
JZ006	<p>IOException détectée : _____</p> <p><i>Description</i> : Une erreur d'E/S inattendue a été détectée dans une couche inférieure. Les exceptions de ce type sont renvoyées en tant qu'exceptions SQL par l'instruction sqlstate JZ006 ERR_IO_EXCEPTION. Ces erreurs résultent fréquemment de problèmes de communication réseau.</p> <p><i>Action</i> : Essayez d'augmenter la taille du cache de l'instruction.</p>
JZ008	<p>La valeur d'index de colonne est incorrecte.</p> <p><i>Description</i> : L'index de colonne demandé est inférieur à 1 ou supérieur à l'index de colonne disponible le plus élevé.</p> <p><i>Action</i> : Vérifiez l'appel de la méthode getXXX() et le texte de la requête initiale, ou appelez rs.next().</p>

Etat SQL	Message/Description/Action
JZ009	<p>L'erreur suivante a été détectée pendant la conversion. Message d'erreur : _____</p> <p><i>Description :</i> Les causes possibles sont nombreuses :</p> <ul style="list-style-type: none"> • Conversion entre deux types de données incompatibles, par exemple une date est convertie en int (nombre entier). • Tentative de conversion d'une chaîne contenant un caractère non numérique en un type numérique. • Erreur de formatage, telle que dans une chaîne de date et heure. <p><i>Action :</i> Vérifiez que la spécification JDBC supporte la conversion de type demandée. Assurez-vous que les chaînes sont formatées correctement. Si une chaîne contient des caractères non numériques, évitez de la convertir en une chaîne de type numérique.</p>
JZ00B	<p>Dépassement numérique.</p> <p><i>Description :</i> Vous avez essayé d'envoyer un BigInteger en tant que valeur numérique TDS, mais la valeur était trop élevée, ou vous avez essayé d'envoyer une instruction Java long en tant que int (nombre entier) et la valeur était trop élevée.</p> <p><i>Action :</i> Ces valeurs ne peuvent pas être stockées dans Sybase. Pour l'instruction long, essayez d'utiliser une valeur numérique Sybase. Il n'existe aucune solution pour Bignum.</p>
JZ00E	<p>Attempt to call execute() or executeUpdate() for a statement where setCursorName() has been called. (Ni execute() ni executeUpdate() ne peut être appelée sur une instruction comprenant setCursorName().)</p> <p><i>Action :</i> Evitez d'appeler execute ou executeUpdate sur une instruction qui définit un nom de curseur. Pour supprimer ou modifier un curseur, utilisez une instruction distincte. Pour plus d'informations, reportez-vous à la section "Utilisation de curseurs dans des jeux de résultats", page 46.</p>
JZ00F	<p>Cursor name has already been set by setCursorName(). (Nom de curseur est déjà défini par setCursorName().)</p> <p><i>Action :</i> Le nom de curseur ne doit être défini qu'une seule fois par instruction. Fermez le jeu de résultats associé à l'instruction de curseur courante.</p>
JZ00G	<p>Aucune valeur de colonne n'a été passée pour la mise à jour de cette ligne.</p> <p><i>Description :</i> Vous essayez d'actualiser une ligne qui n'a pas été modifiée.</p> <p><i>Action :</i> Pour redéfinir des valeurs de colonne dans une ligne, appelez les méthodes updateXX() avant updateRow().</p>

Etat SQL	Message/Description/Action
JZ00H	<p>Le jeu de résultat ne peut pas être mis à jour. Utilisez <code>Statement.setResultSetConcurrencyType()</code>.</p> <p><i>Action</i> : Pour rendre modifiable un jeu de résultats en lecture seule, utilisez la méthode <code>Statement.setResultSetConcurrencyType()</code> ou ajoutez une clause <code>for update</code> dans l'instruction SQL <code>select</code>.</p>
JZ00L	<p>Echec de la connexion. Examinez les <code>SQLWarnings</code> associés à cette exception pour la ou les raisons suivantes :</p> <p><i>Action</i> : Suivez les instructions du message et prenez les mesures correctives nécessaires selon l'origine de l'échec de la connexion.</p>
JZ010	<p>Impossible de désérialiser une valeur d'objet. Message d'erreur : _____</p> <p><i>Action</i> : Vérifiez que l'objet Java de la base de données met en oeuvre l'interface sérialisable et qu'il est défini dans la variable système <code>CLASSPATH</code> locale.</p>
JZ011	<p>Exception de format détectée durant l'analyse des propriétés de connexion _____. <i>Description</i> : La valeur d'une propriété de connexion numérique est différente d'un nombre entier. <i>Action</i> : Corrigez l'erreur.</p>
JZ012	<p>Erreur interne. Contactez le Support Technique de Sybase. Le type d'accès n'est pas admis par la propriété de connexion. <i>Action</i> : Prenez contact avec le Support Technique de Sybase.</p>
JZ013	<p>Erreur lors de la recherche de l'entrée JNDI : _____ <i>Action</i> : Corrigez l'URL JNDI ou insérez une nouvelle entrée dans le service de répertoire.</p>
JZ0BD	<p>Valeur hors intervalle ou non admise utilisée comme paramètre de méthode <i>Action</i> : Vérifiez la valeur de paramètre dans la méthode.</p>
JZ0BE	<p><code>BatchUpdateException</code> : une erreur est survenue lors de l'exécution de l'instruction par batch : _____.</p>
JZ0BP	<p>Les paramètres de sortie ne sont pas admis dans les instructions <code>Batch Update</code>. <i>Action</i> :</p>

Etat SQL	Message/Description/Action
JZ0BR	<p>Le curseur ne se trouve pas sur une ligne qui supporte la méthode _____.</p> <p><i>Action :</i> Vous avez tenté d'appeler une méthode ResultSet qui est incorrecte pour la position de ligne courante (par exemple, si vous avez appelé insertRow() alors que le curseur ne se trouve pas sur une ligne d'insertion).</p> <p><i>Action :</i> Appelez une méthode ResultSet appropriée à la position de ligne courante.</p>
JZ0BS	<p>Instructions par batch non supportées.</p>
JZ0BT	<p>La méthode _____ n'est pas supportée pour les ResultSets du type _____.</p> <p><i>Description :</i> Vous avez tenté d'appeler une méthode ResultSet incorrecte pour le type de ResultSet.</p> <p><i>Action :</i> Appelez une méthode ResultSet appropriée au type de ResultSet.</p>
JZ0C0	<p>La connexion a déjà été arrêtée.</p> <p><i>Description :</i> L'application a déjà appelé la méthode Connection.close() sur cet objet de connexion. Celui-ci ne peut plus être utilisé.</p> <p><i>Action :</i> Corrigez le code de sorte que les références à l'objet de connexion soient annulées dès la fermeture d'une connexion.</p>
JZ0D0	<p>Cette installation jConnect n'a pas encore été enregistrée. Vous devez installer les classes SybDriverKey appropriées.</p> <p><i>Action :</i> Accédez au site Web de jConnect pour enregistrer le logiciel jConnect : http://www.sybase.com/products/internet/jconnect/.</p> <p>Une fois l'enregistrement terminé, vous pouvez télécharger les classes SybDriverKey nécessaires à l'activation du pilote jConnect.</p>
JZ0D2	<p>Votre licence Sybase JDBC a expiré le _____. Veuillez acquérir</p> <p><i>Action :</i> Prenez contact avec Sybase afin d'acquérir une nouvelle licence pour votre pilote jConnect.</p>
JZ0D3	<p>Votre licence Sybase JDBC va expirer le _____. Veuillez acquérir Veuillez acquérir une nouvelle licence.</p> <p><i>Action :</i> Prenez contact avec Sybase afin d'acquérir une nouvelle licence pour votre pilote jConnect.</p>

Etat SQL	Message/Description/Action
JZ0D4	<p>Protocole non reconnu dans l'URL JDBC de Sybase :</p> <p><i>Description</i> : L'URL de connexion spécifié utilise un autre protocole que TDS, qui est le seul protocole actuellement pris en charge par jConnect.</p> <p><i>Action</i> : Vérifiez la définition de l'URL. Si TDS y apparaît comme sous-protocole, vérifiez que la casse et le format suivants sont utilisés :</p> <p>jdbc:sybase:Tds:hôte:port</p> <p>Si JNDI y apparaît comme sous-protocole, vérifiez que l'URL commence par :</p> <p>jdbc:sybase:jndi:</p>
JZ0D5	<p>Erreur lors du chargement du protocole.</p> <p><i>Action</i> : Vérifiez le paramétrage de la variable système CLASSPATH.</p>
JZ0D6	<p>Le numéro de version indiqué dans setVersion n'est pas reconnu. Sélectionnez l'une des valeurs SybDriver.VERSION_* et assurez-vous que la version de jConnect que vous utilisez a bien le niveau de version spécifié ou un niveau supérieur.</p> <p><i>Action</i> : Reportez-vous aux instructions du message.</p>
JZ0D7	<p>Erreur lors de l'appel du fournisseur de l'URL. Message d'erreur : _____</p> <p><i>Action</i> : Vérifiez que l'URL JNDI est correct.</p>
JZ0D8	<p>Erreur lors de l'initialisation du fournisseur de l'URL : _____</p> <p><i>Action</i> : Vérifiez que l'URL JNDI est correct.</p>
JZ0EM	<p>Fin des données.</p> <p><i>Action</i> : Signalez cette erreur au Support Technique de Sybase.</p>
JZ0H0	<p>Impossible de démarrer un thread pour le gestionnaire d'événement ; nom de l'événement :</p> <p><i>Action</i> : Signalez cette erreur au Support Technique de Sybase.</p>
JZ0H1	<p>Une notification d'événement a été reçue mais le gestionnaire d'événement est introuvable ; nom de l'événement :</p> <p><i>Action</i> : Signalez cette erreur au Support Technique de Sybase.</p>
JZ0HC	<p>Un caractère incorrect '_____' a été détecté pendant la conversion de l'hexadécimal.</p> <p><i>Description</i> : Un des caractères de la chaîne censée représenter une valeur binaire ne fait pas partie des caractères alphanumériques admis (0 à 9, A à F) dans une valeur hexadécimale.</p> <p><i>Action</i> : Vérifiez le contenu de la chaîne et corrigez l'erreur.</p>

Etat SQL	Message/Description/Action
JZ011	<p>I/O Layer: Error reading stream. (Couche E/S : erreur lors de la lecture du flux.)</p> <p><i>Description :</i> La connexion n'a pas pu écrire le résultat demandé. Elle a dû être interrompue au terme du temps imparti.</p> <p><i>Action :</i> Augmentez le temps imparti pour l'instruction.</p>
JZ012	<p>I/O layer: Error writing stream. (Couche E/S : erreur lors de l'écriture du flux.)</p> <p><i>Description :</i> La connexion n'a pas pu écrire le résultat demandé. Elle a dû être interrompue au terme du temps imparti.</p> <p><i>Action :</i> Augmentez le temps imparti pour l'instruction.</p>
JZ013	<p>Propriété inconnue. Ce message signale un problème interne. Signalez l'erreur au Support Technique de Sybase.</p> <p><i>Action :</i> Ce message signale un problème interne. Signalez cette erreur au Support Technique de Sybase.</p>
JZ015	<p>Une propriété CHARSET inconnue a été spécifiée : _____.</p> <p><i>Description :</i> Le code de jeu de caractères associé à la propriété de connexion CHARSET n'est pas admis.</p> <p><i>Action :</i> Corrigez l'erreur. Reportez-vous à la section "Convertisseurs de jeu de caractères jConnect", page 32.</p>
JZ016	<p>Une erreur s'est produite durant la conversion d'UNICODE en un jeu de caractères du serveur. Message d'erreur : _____</p> <p><i>Action :</i> Remplacez le code de jeu de caractères associé à la propriété de connexion CHARSET sur le client jConnect. Vous devez être en mesure d'envoyer tous les caractères voulus au serveur. Il peut également être nécessaire d'installer un autre jeu de caractères sur le serveur.</p>
JZ017	<p>Aucune réponse de la gateway proxy.</p> <p><i>Description :</i> La passerelle Cascade ou de sécurité ne répond pas.</p> <p><i>Action :</i> Vérifiez que la passerelle est installée correctement et qu'elle est active.</p>
JZ018	<p>Connexion refusée par la gateway proxy. Message de la gateway : _____</p> <p><i>Description :</i> Le serveur ou la passerelle Web spécifiée par la propriété de connexion PROXY a rejeté votre demande de connexion.</p> <p><i>Action :</i> Vérifiez les journaux d'accès et d'erreurs sur le proxy pour connaître la raison du rejet. Vérifiez que le proxy est une passerelle JDBC.</p>

Etat SQL	Message/Description/Action
JZ0I9	<p>Cet InputStream était fermé.</p> <p><i>Description :</i> Vous avez essayé de lire un InputStream issu d'un getAsciiStream(), getUnicodeStream(), or getBinaryStream(), mais l'InputStream était déjà fermé. Le flux a pu être fermé par un changement de colonne ou l'annulation du jeu de résultats alors que les ressources disponibles sont insuffisantes pour mettre les données en cache.</p> <p><i>Action :</i> Augmentez la taille du cache ou parcourez les colonnes dans l'ordre.</p>
JZ0IA	<p>Erreur de troncature lors de la tentative d'envoi.</p> <p><i>Description :</i> Une erreur de troncature s'est produite lors de la conversion du jeu de caractères avant l'envoi d'une chaîne. La chaîne convertie dépasse la longueur qui lui a été allouée.</p> <p><i>Action :</i> Remplacez le code de jeu de caractères associé à la propriété de connexion CHARSET sur le client jConnect. Vous devez être en mesure d'envoyer tous les caractères voulus au serveur. Il peut également être nécessaire d'installer un autre jeu de caractères sur le serveur.</p>
JZ0IS	<p>Si vous lisez cet Input Stream, vous ne pourrez plus l'utiliser pour mettre à jour la base de données.</p> <p><i>Description :</i> Vous avez, après avoir mis à jour une colonne dans un jeu de résultats, essayé de lire la valeur de colonne modifiée à l'aide de l'une des méthodes SybResultSet : getAsciiStream(), getUnicodeStream(), getBinaryStream(). Or, cela n'est pas autorisé.</p> <p><i>Action :</i> Evitez d'extraire des flux d'entrée issus de colonnes modifiées.</p>
JZ0J0	<p>Les valeurs d'offset et/ou de longueur excèdent la longueur text/image courante.</p> <p><i>Action :</i> Vérifiez les valeurs d'offset et/ou de longueur.</p>
JZ0NC	<p>Appel de wasNull alors qu'aucun nom de colonne n'a été associé.</p> <p><i>Description :</i> Vous pouvez uniquement appeler wasNull() après avoir émis un appel d'obtention de colonne, tel que getInt() ou getBinaryStream().</p> <p><i>Action :</i> Déplacez l'appel wasNull() dans le code.</p>
JZ0NE	<p>Le format de l'URL est incorrect. URL : _____. Message d'erreur : _____</p> <p><i>Action :</i> Vérifiez le format de l'URL. Vérifiez que le numéro de port ne comporte que de caractères numériques.</p>

Etat SQL	Message/Description/Action
JZ0NF	<p>Impossible de charger SybSocketFactory. Vérifiez l'orthographe du nom de classe, assurez-vous que le package est bien spécifié, que la classe se trouve dans l'arborescence des classes et qu'il contient un public zero-argument constructor.</p> <p><i>Action :</i> Reportez-vous aux instructions du message.</p>
JZ0P1	<p>Type de résultat inattendu.</p> <p><i>Description :</i> La base de données a renvoyé un résultat que l'instruction ne peut pas transmettre à l'application ou que celle-ci n'attend pas à ce stade. L'application utilise JDBC de manière incorrecte pour exécuter la requête ou la procédure stockée. Si l'application JDBC est connectée à une application Open Server, l'erreur peut provenir de l'application Open Server qui envoie des résultats inattendus.</p> <p><i>Action :</i> Utilisez les outils de débogage com.sybase.utils.Debug(true, "ALL") afin d'en savoir plus sur ce type de résultat et sa cause.</p>
JZ0P4	<p>Erreur de protocole. Ce message signale un problème interne. Signalez l'erreur au Support Technique de Sybase.</p> <p><i>Action :</i> Reportez-vous aux instructions du message.</p>
JZ0P7	<p>La colonne n'a pas été mise en cache ; utilisez la propriété RE-READABLE_COLUMNS.</p> <p><i>Description :</i> Ce message signale une tentative de relecture d'une colonne ou de lecture d'une colonne dans le désordre bien que la propriété de connexion REPEAT_READ ait la valeur "false".</p> <p>Lorsque REPEAT_READ a la valeur "false", les colonnes ne peuvent être lues que dans l'ordre ascendant des index et chaque ligne de données d'une colonne ne peut être lue qu'une seule fois. Par exemple, vous ne pouvez ni lire la colonne 2 d'une ligne après avoir lu la colonne 3, ni lire la valeur de celle-ci une seconde fois.</p> <p><i>Action :</i> attribuez la valeur "true" à REPEAT_READ ou évitez de relire une valeur de colonne et lisez les colonnes dans l'ordre.</p>
JZ0P8	<p>Le nom du type de colonne RSMDA demandé est inconnu.</p> <p><i>Description :</i> jConnect n'a pas pu déterminer quel était le nom d'un type de colonne dans la méthode ResultSetMetaData.getColumnTypeName().</p> <p><i>Action :</i> Vérifiez que votre base de données dispose des procédures stockées les plus récentes pour le renvoi de métadonnées.</p>

Etat SQL	Message/Description/Action
JZ0P9	<p>Détection d'une requête COMPUTE BY. Ce type de résultat n'est pas supporté ; la requête a été annulée.</p> <p><i>Description</i> : La requête exécutée a renvoyé des résultats de type COMPUTE, ce que jConnect ne prend pas en charge.</p> <p><i>Action</i> : Supprimez l'instruction COMPUTE BY de votre requête ou procédure stockée.</p>
JZ0PA	<p>La requête a été annulée et la réponse a été supprimée.</p> <p><i>Description</i> : L'annulation a probablement été émise par une autre instruction sur cette connexion.</p> <p><i>Action</i> : Vérifiez la chaîne des exceptions et avertissements SQL sur cette instruction, ainsi que sur d'autres instructions pour en déterminer l'origine.</p>
JZ0PB	<p>Le serveur ne supporte pas l'opération demandée.</p> <p><i>Description</i> : Lorsque jConnect établit une connexion à un serveur, il soumet à ce dernier la liste des fonctionnalités qu'il souhaite voir prendre en charge et le serveur, à son tour, indique à jConnect les fonctionnalités qu'il supporte. Ce message d'erreur est envoyé lorsqu'une application fait appel à une fonction déclarée comme n'étant pas supportée lors de la phase de dialogue initiale.</p> <p>Par exemple, jConnect génère ce message si la base de données ne supporte pas la précompilation d'instructions SQL dynamiques, que le code appelle l'instruction <code>SybConnection.prepareStatement(sql_stmt, dynamic)</code>, et que <i>dynamic</i> a la valeur "true".</p> <p><i>Action</i> : Modifiez le code pour éviter qu'il fasse appel à des fonctionnalités non supportées.</p>
JZ0R0	<p>ResultSet a déjà été fermé.</p> <p><i>Description</i> : La méthode <code>ResultSet.close()</code> déjà été appelée sur l'objet de jeu de résultats ; vous ne pouvez plus utiliser ce dernier.</p> <p><i>Action</i> : Corrigez le code de sorte que les références à l'objet <code>ResultSet</code> soient annulées dès la fermeture d'un jeu de résultats.</p>
JZ0R1	<p>Le jeu de résultats est IDLE car vous ne tentez pas d'accéder à une ligne.</p> <p><i>Description</i> : L'application a appelé l'une des méthodes d'extraction de données de colonne <code>ResultSet.getXXX</code> mais aucune ligne n'est active ; l'application n'a pas appelé la méthode <code>ResultSet.next()</code>, ou <code>ResultSet.next()</code> renvoyé la valeur "false" pour indiquer qu'il n'existe aucune donnée.</p> <p><i>Action</i> : Vérifiez que <code>rs.next()</code> a la valeur "true" avant d'appeler <code>rs.getXXX</code>.</p>

Etat SQL	Message/Description/Action
JZOR2	<p>Aucun jeu de résultats pour cette requête.</p> <p><i>Description :</i> Vous avez utilisé <code>Statement.executeQuery()</code>, mais l'instruction n'a renvoyé aucune ligne.</p> <p><i>Action :</i> Exécutez <code>executeUpdate</code> lorsque les instructions ne renvoient aucune ligne.</p>
JZOR3	<p>La colonne est DEAD. Erreur interne. Contactez le Support Technique de Sybase.</p> <p><i>Action :</i> Reportez-vous aux instructions du message.</p>
JZOR4	<p>La colonne ne possède pas de pointeur de texte. Soit il ne s'agit pas d'une colonne de type <code>text/image</code>, soit la colonne a la valeur NULL.</p> <p><i>Description :</i> Vous ne pouvez pas mettre à jour une colonne <code>text/image</code> de valeur NULL. En effet, les colonnes de ce type ne comprennent aucun pointeur de texte.</p> <p><i>Action :</i> Vérifiez que vous n'êtes pas en train de modifier ou de récupérer un pointeur de texte dans une colonne qui ne supporte pas les données de type <code>text</code> ou <code>image</code>. Assurez-vous également que vous ne modifiez pas une colonne de type <code>text/image</code> de valeur NULL. Insérez les données avant de les modifier.</p>
JZORM	<p>La dernière mise à jour ou suppression opérée sur cette ligne a été effectuée. Impossible de rafraîchir la ligne à partir de la base de données.</p> <p><i>Description :</i> Après avoir mis à jour une ligne de la base de données à l'aide de la méthode <code>SybCursorResult.updateRow()</code>, ou l'avoir supprimé à l'aide de <code>SybCursorResult.deleteRow()</code>, vous avez utilisé <code>SybCursorResult.refreshRow()</code> pour la rafraîchir.</p> <p><i>Action :</i> Ne cherchez pas à rafraîchir une ligne après l'avoir mise à jour ou supprimée de la base de données.</p>
JZOS0	<p>Etat du moteur : BUSY.</p> <p><i>Description :</i> Cette erreur n'est générée que par la méthode <code>Statement.setCursorname()</code> lorsque l'application tente de définir le nom du curseur alors que l'instruction est déjà en cours d'utilisation et qu'elle a produit un jeu de résultats non-curseur qui doit être lu.</p> <p><i>Action :</i> Définissez le nom du curseur dans une instruction avant d'exécuter des requêtes sur cette dernière, ou appelez <code>Statement.cancel()</code> avant de définir le nom du curseur ; vous serez ainsi assuré de la disponibilité de l'instruction.</p>
JZOS1	<p>Etat du moteur : tentative d'exécution de FETCH sur un IDLE.</p> <p><i>Description :</i> Une erreur interne s'est produite dans l'instruction.</p> <p><i>Action :</i> Fermez l'instruction et ouvrez-en une autre.</p>

Etat SQL	Message/Description/Action
JZ0S2	<p>L'objet Statement a déjà été fermé.</p> <p><i>Description</i> : La méthode Statement.close() déjà été appelée sur l'objet d'instruction ; vous ne pouvez plus utiliser ce dernier.</p> <p><i>Action</i> : Corrigez le code de sorte que les références à l'objet d'instruction soient annulées dès la fermeture d'une instruction.</p>
JZ0S3	<p>La méthode récupérée ne peut pas être réutilisée dans cette sous-classe.</p> <p><i>Description</i> : PreparedStatement ne supporte pas executeQuery(String), executeUpdate(String), ni execute(String).</p> <p><i>Action</i> : Pour transmettre une chaîne de requête, utilisez Statement, pas PreparedStatement.</p>
JZ0S4	<p>Impossible d'exécuter une requête entièrement vide.</p> <p><i>Action</i> : N'exécutez pas de requête vide ("").</p>
JZ0S8	<p>Séquence de caractères d'échappement incorrecte dans une requête SQL : '_____'. _____</p> <p><i>Description</i> : Cette erreur résulte d'une erreur de syntaxe dans la séquence d'échappement.</p> <p><i>Action</i> : Recherchez la syntaxe correcte dans la documentation de JDBC.</p>
JZ0S9	<p>Impossible d'exécuter une requête entièrement vide.</p> <p><i>Action</i> : N'exécutez pas de requête vide ("").</p>
JZ0SA	<p>Instruction préparée : les paramètres d'entrée ne sont pas configurés, index : _____.</p> <p><i>Action</i> : Vérifiez que chaque paramètre d'entrée a une valeur.</p>
JZ0SB	<p>L'index des paramètres est en dehors de l'intervalle admis : _____.</p> <p><i>Description</i> : Vous avez essayer d'obtenir, de définir ou d'enregistrer un paramètre dont l'index dépasse le nombre maximal de paramètres admis.</p> <p><i>Action</i> : Vérifiez le nombre de paramètres contenus dans votre requête.</p>
JZ0SC	<p>CallableStatement : vous avez tenté de transformer un état renvoyé en paramètre d'entrée.</p> <p><i>Description</i> : Vous avez préparé un appel de procédure stockée qui renvoie un état, mais le paramètre défini (1) correspond à l'état renvoyé.</p> <p><i>Action</i> : Les paramètres admis avec ce type d'appel commencent à 2.</p>

Etat SQL	Message/Description/Action
JZ0SD	<p>Aucun paramètre enregistré n'a été détecté parmi les paramètres de sortie.</p> <p><i>Description :</i> Il s'agit d'une erreur logique d'application. Vous avez essayé d'appeler getXXX() ou wasNull() sur un paramètre, mais vous n'avez encore lu aucun paramètre ou il n'existe aucun paramètre de sortie.</p> <p><i>Action :</i> Vérifiez que l'application a enregistré des paramètres de sortie sur CallableStatement, que l'instruction a été exécutée et que les paramètres de sortie ont été lus.</p>
JZ0SE	<p>Type d'objet incorrect (ou objet null) spécifié pour setObject().</p> <p><i>Description :</i> Un argument de type incorrect a été transmis à PreparedStatement.setObject.</p> <p><i>Action :</i> Reportez-vous à la documentation de JDBC. L'argument doit être une constante issue de java.sql.Types.</p>
JZ0SF	<p>Aucun paramètre attendu. Vérifiez que la requête a été émise.</p> <p><i>Description :</i> Vous avez essayé de définir un paramètre dans une instruction qui n'admet aucun paramètre.</p> <p><i>Action :</i> Vérifiez que la requête a été envoyée avant de définir les paramètres.</p>
JZ0SG	<p>Callable Statement a renvoyé moins de paramètres de sortie que prévu pour l'application.</p> <p><i>Description :</i> Cette erreur se produit si vous appelez CallableStatement.registerOutParam() davantage de paramètres que vous avez déclaré de paramètres "OUTPUT" dans la procédure stockée. Pour plus d'informations, reportez-vous à la section "Le RPC renvoie moins de paramètres de sortie que prévu", page 109 .</p> <p><i>Action :</i> Vérifiez vos procédures stockées et les appels registerOutParameter. Assurez-vous d'avoir déclaré tous les paramètres adéquats comme "OUTPUT". Reportez-vous à la ligne de code suivante :</p> <pre>create procedure yourproc (@p1 int OUTPUT, ...</pre> <p>Remarque Si cette erreur survient lorsque vous utilisez Adaptive Server Anywhere (anciennement SQL Anywhere), vous devez installer la version 5.5.04 d'Adaptive Server Anywhere.</p>
JZ0SH	<p>La fonction statique escape a été utilisée mais les informations d'accès aux métadonnées sont introuvables sur ce serveur.</p> <p><i>Action :</i> Installez les informations d'accès aux métadonnées avant d'utiliser des séquences d'échappement de fonction statique.</p>

Etat SQL	Message/Description/Action
JZ0SI	<p>La fonction statique escape utilisée n'est pas supportée par ce serveur.</p> <p><i>Action</i> : N'utilisez pas cette séquence d'échappement.</p>
JZ0SJ	<p>Les informations d'accès aux métadonnées sont introuvables dans cette base de données.</p> <p><i>Action</i> : Installez les informations sur les métadonnées avant d'effectuer des appels de métadonnées.</p>
JZ0SM	<p>Type SQL non supporté :</p> <p><i>Action</i> : N'utilisez pas Types.NULL, Types.OTHER, ni PreparedStatement.setObject(null).</p>
JZ0SN	<p>setMaxFieldSize : la taille de ce champ ne peut pas être négative.</p> <p><i>Action</i> : Utilisez une valeur positive ou zéro (0 - infinie) lorsque vous appelez setMaxFieldSize.</p>
JZ0T2	<p>Erreur de lecture du thread du service récepteur. Contrôlez les communications réseau.</p> <p><i>Action</i> : Suivez les instructions du message d'erreur.</p>
JZ0T3	<p>L'opération de lecture a dépassé le temps imparti.</p> <p><i>Description</i> : Le temps imparti pour la lecture d'une réponse à une requête a été dépassé.</p> <p><i>Action</i> : Augmentez ce délai en appelant Statement.setQueryTimeout().</p>
JZ0T4	<p>L'opération d'écriture a dépassé le temps imparti. Temps imparti (en ms.) ____.</p> <p><i>Description</i> : Le temps imparti pour l'émission d'une requête a été dépassé.</p> <p><i>Action</i> : Augmentez ce délai en appelant Statement.setQueryTimeout().</p>
JZ0T5	<p>Le cache est plein. Utilisez la valeur par défaut ou une valeur supérieure pour STREAM_CACHE_SIZE.</p> <p><i>Action</i> : Suivez les instructions du message d'erreur.</p>
JZ0T6	<p>Erreur lors de la lecture de l'URL TDS encapsulé.</p> <p><i>Description</i> : Un erreur s'est produite dans le protocole encapsulé lors de la lecture de l'en-tête de l'URL.</p> <p><i>Action</i> : Vérifiez l'URL que vous avez défini pour la connexion.</p>
JZ0T7	<p>Erreur de lecture du thread du service récepteur : ThreadDeath détecté. Contrôlez les communications réseau.</p> <p><i>Action</i> : Suivez les instructions du message d'erreur puis redémarrez l'application. Si l'incident persiste, prenez contact avec le Support Technique de Sybase.</p>

Etat SQL	Message/Description/Action
JZ0T9	<p>La requête à émettre n'a pas été synchronisée. Contactez le Support Technique de Sybase.</p> <p><i>Action :</i> Reportez-vous aux instructions du message.</p>
JZ0TC	<p>Tentative de conversion non admise entre deux types de données.</p> <p><i>Description :</i> La conversion entre les types Java et SQL a échoué.</p> <p><i>Action :</i> Vérifiez que la conversion de type demandée est supportée par JDBC.</p>
JZ0TE	<p>Tentative de conversion non admise entre deux types de données. Valid database types are: '_____.'</p> <p><i>Description :</i> Le type de données de colonne de la base de données et le type demandé dans l'appelResultSet.getXXX() ne sont pas implicitement convertibles.</p> <p><i>Action :</i> Utilisez un des types de données autorisés cités dans le message d'erreur.</p>
JZ0US	<p>La propriété de connexion SYB SOCKET_FACTORY a été définie et la propriété de connexion PROXY a été paramétrée sur l'URL d'un servlet. Le pilote jConnect ne supporte pas cette combinaison. Si vous voulez sécuriser vos envois d'adresses HTTP à partir d'applets exécutés depuis un explorateur, utilisez un URL proxy commençant par "https://".</p> <p><i>Action :</i> Reportez-vous aux instructions du message.</p>
JZ0CX	<p>_____ is an unrecognized transaction coordinator type.</p> <p><i>Description :</i> Les informations de métadonnées indiquent que le serveur support les transactions distribuées mais jConnect ne supporte pas le protocole utilisé.</p> <p><i>Action :</i> Vérifiez que vous avez installé les scripts de métadonnées les plus récents. Si l'erreur persiste, prenez contact avec le Support Technique de Sybase.</p>
JZ0XS	<p>The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.</p> <p><i>Description :</i> Le serveur auquel jConnect a tenté de se connecter ne supporte pas les transactions distribuées.</p> <p><i>Action :</i> N'utilisez pas XADataSource avec ce serveur, ou mettez à niveau/configurez le serveur pour la gestion des transactions distribués.</p>

Etat SQL	Message/Description/Action
JZ0XU	<p>L'utilisateur courant n'est pas habilité à exécuter des transactions de type XA. Veillez à ce que l'utilisateur dispose du droit _____ .</p> <p><i>Description :</i> L'utilisateur connecté à la base de données n'est pas autorisé à exécuter des transactions distribuées. Il est probable qu'il ne possède pas le rôle adéquat (indiqué dans le message).</p> <p><i>Action :</i> Accordez à l'utilisateur le rôle indiqué dans le message d'erreur ou confiez l'exécution de la transaction à un autre utilisateur qui possède ce rôle.</p>
S0022	<p>'_____' est un nom de colonne incorrect.</p> <p><i>Description :</i> Le nom de colonne indiqué n'existe pas.</p> <p><i>Action :</i> Vérifiez l'orthographe de ce nom.</p>
ZZ00A	<p>La méthode ne s'est pas achevée et n'aurait pas dû être appelée.</p> <p><i>Description :</i> Vous avez tenté d'utiliser une méthode qui n'est pas mise en oeuvre.</p> <p><i>Action :</i> Reportez-vous aux notes de mise à jour fournies avec votre version de jConnect pour obtenir plus d'informations. Vous pouvez également vérifier sur la page Web de jConnect (http://www.sybase.com) s'il n'existe pas une version plus récente de jConnect qui met en oeuvre la méthode. Dans la négative, n'utilisez pas la méthode.</p>

Programmes exemple jConnect

La présente annexe décrit les programmes exemple proposés avec jConnect de Sybase jConnect.

Elle se compose des sections suivantes :

Nom	Page
Exécution d'IsqlApp	172
Exécution de programmes et codes exemple de jConnect	174

Exécution d'IsqlApp

IsqlApp permet d'émettre des commandes **isql** à partir de la ligne de commande.

La syntaxe d'**IsqlApp** est la suivante :

```
IsqlApp [-U nom_utilisateur] [-P mot_de_passe]
        [-S nom_serveur]
        [-G passerelle]
        [-p {http|https}]
        [-D liste_classe_débogage]
        [-v]
        [-I fichier_commandes_entrée]
        [-c délimiteur_fin_commande]
        [-C jeu_car] [-L langue]
        [-T ID_session]
        [-V <version {2,3,4,5}>]
```

Paramètre	Description
-U	ID de connexion avec lequel vous souhaitez vous connecter au serveur.
-P	Mot de passe associé à l'ID de connexion.
-S	Nom du serveur auquel vous souhaitez vous connecter.
-G	Adresse de passerelle. Pour le protocole HTTP, l'URL est le suivant : <code>http://hôte:port</code> . Pour utiliser le protocole HTTPS qui supporte le cryptage, l'URL est <code>https://hôte:port/alias_servlet</code> .
-p	Indique si vous souhaitez utiliser le protocole HTTP ou HTTPS qui supporte le cryptage.
-D	Active le débogage de toutes les classes ou de celles spécifiées, séparées par une virgule. Par exemple : <i>-D ALL</i> affiche les résultats de débogage de toutes les classes. <i>-D SybConnection, Tds</i> n'affiche que les résultats de débogage des classes SybConnection et Tds.
-v	Active le mode d'affichage ou d'impression détaillé.
-I	Indique à IsqlApp de prendre en compte les commandes d'un fichier plutôt que celles entrées au clavier. Spécifiez à la suite de ce paramètre le nom du fichier à utiliser pour l'entrée IsqlApp. Le fichier doit contenir des délimiteurs de fin de commande (par défaut, file "go").
-c	Permet de préciser un mot-clé (par exemple, "go") qui, lorsqu'il apparaît seul sur une ligne, met fin à la commande. Vous pouvez ainsi entrer des commandes sur plusieurs lignes avant d'utiliser le mot-clé de fin de commande. Si vous ne précisez pas de délimiteur particulier, chaque nouvelle ligne met fin à une commande.

Paramètre	Description
-C	Définit le jeu de caractères des chaînes acheminées par TDS. Si vous ne spécifiez aucun jeu de caractères, IsqlApp utilise le jeu de caractères par défaut du serveur.
-L	Langue d’affichage des messages d’erreur renvoyés par le serveur et des messages jConnect.
-T	Lorsque ce paramètre est défini, jConnect suppose qu’une application tente de reprendre la communication sur une session TDS existante, maintenue ouverte par la passerelle d’encapsulation par TDS. jConnect passe alors outre la phase initiale de la connexion et fait suivre toutes les requêtes adressées par une application à l’ID de session spécifié.
-V	Active les caractéristiques spécifiques à la version utilisée. Reportez-vous à la section “Propriété de connexion JCONNECT_VERSION” , page 9.

Remarque Vous devez insérer un espace entre chaque option.

Pour obtenir une description détaillée des options de ligne de commande, entrez :

```
java IsqlApp -help
```

L’exemple ci-après montre comment se connecter à une base de données sur un hôte “monserveur” via le port “3756” et exécuter un script **isql** appelé “monscript” :

```
java IsqlApp -U sa -P sapassword  
-S jdbc:sybase:Tds:monserveur:3756  
-I $JDBC_HOME/sp/monscript -c run
```

Remarque Un applet qui fournit un accès à une interface utilisateur graphique pour les commandes **isql** est également disponible :

Pour jConnect 4.x :

\$JDBC_HOME/sample/gateway.html (UNIX)

%JDBC_HOME%\sample\gateway.html (Windows)

Pour jConnect 5.x :

\$JDBC_HOME/sample2/gateway.html (UNIX)

%JDBC_HOME%\sample2\gateway.html (Windows)

Exécution de programmes et codes exemple de jConnect

jConnect est fourni avec plusieurs programmes exemple qui illustrent les différents sujets abordés dans ce chapitre et peuvent vous aider à mieux comprendre l'interaction entre jConnect et des classes et des méthodes JDBC. Vous trouverez également dans cette section un extrait de code exemple.

Applications exemple

Vous pouvez installer des programmes exemple en même temps que jConnect. Ces exemples comprennent le code source ; vous pourrez ainsi examiner comment jConnect met en oeuvre différentes classes et méthodes JDBC. Pour plus d'informations sur l'installation des programmes exemple, reportez-vous au document *jConnect for JDBC - Guide d'installation*.

Remarque Les programmes exemple de jConnect sont uniquement des programmes de démonstration.

Une fois installés, ces programmes se trouvent dans le sous-répertoire *sample* (jConnect 4.x) ou *sample2* (jConnect 5.x) du répertoire d'installation jConnect. Vous trouverez dans l'un de ces sous-répertoire le fichier *index.html* qui présente une liste exhaustive des exemples disponibles, accompagnés d'une description. En outre, ce fichier vous permet d'afficher et d'exécuter ces programmes exemple sous forme d'applets.

Exécution des applets exemple

Pour exécuter certains des programmes exemple en tant qu'applets, vous pouvez utiliser un explorateur Web. Cela vous permet d'afficher le code source tout en visualisant les résultats.

Pour cela, il convient néanmoins de démarrer la passerelle Cascade. Reportez-vous à la section [“Utilisation de la passerelle Cascade”](#), page 139.

Pour ouvrir le fichier *index.html*, utilisez votre explorateur Web :

Pour jConnect 4.x, entrez :

<http://localhost:8000/sample/index.html>

Pour jConnect 5.x, entrez :

http://localhost:8000/sample2/index.html

Exécution des programmes exemple avec Adaptive Server Anywhere

Tous les programmes exemple sont compatibles avec Adaptive Server, ce qui n'est pas le cas avec Adaptive Server Anywhere. Pour connaître la liste des programmes exemple compatibles avec Adaptive Server Anywhere, consultez le fichier *index.html*, situé dans le sous-répertoire *sample* ou *sample2*.

Pour exécuter les programmes exemple compatibles avec Adaptive Server Anywhere, vous devez installer le script *pubs2_any.sql* sur le serveur Adaptive Server Anywhere. Ce script se trouve dans le sous-répertoire *sample* (jConnect 4.1) ou *sample2* (jConnect 5.0).

Sous Windows, placez-vous sur l'invite DOS et entrez :

```
java IsqlApp -U dba -P mot_de_passe
-S jdbc:sybase:Tds:[nom_hôte]:[port]
-I %JDBC_HOME%\sample\pubs2_any.sql -c go
```

Sous UNIX, entrez :

```
java IsqlApp -U dba -P mot_de_passe
-S jdbc:sybase:Tds:[nom_hôte]:[port]
-I $JDBC_HOME/sample/pubs2_any.sql -c go
```

Code exemple

Le code exemple ci-après illustre l'appel du pilote jConnect, l'établissement d'une connexion, l'émission d'une instruction SQL et le traitement des résultats.

```
import java.io.*;
import java.sql.*;

public class SampleCode
{
    public static void main(String args[])
    {
        try
        {
            /*
             * Ouvrir la connexion. Possibilité d'envoyer une SQLException.
             */
            Connection con = DriverManager.getConnection(
                "jdbc:sybase:Tds:monserveur:3767", "sa", "");
            /*
```

```
    * Créer un objet Statement object, conteneur de l'instruction
    * SQL. Possibilité d'envoyer une SQLException.
    */
    Statement stmt = con.createStatement();
/*
    * Créer un objet de jeu de résultats en exécutant la requête.
    * Possibilité d'envoyer une SQLException.
    */
    ResultSet rs = stmt.executeQuery("Select 1");
/*
    * Traiter le jeu de résultats.
    */

    if (rs.next())
    {
        int value = rs.getInt(1);
        System.out.println("Valeur extraite " + value);
    }
}
/*
    * Traiter les exceptions.
    */
catch (SQLException sqe)
{
    System.out.println("Exception inattendue : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
    System.exit(1);
}
System.exit(0);
}
```

Index

A

- Adaptive Server
 - connexion à 18
 - exemple de connexion 19
- Adaptive Server Anywhere 16, 19
 - stockage et récupération d'un objet Java 74
 - symbole euro 36
- API native
 - partiellement écrite en Java 2
- appel de procédure à distance interserveur 44
- applet 139, 140
 - isql, exemple 143
- application
 - migration vers jConnect 4.1 128
 - migration vers jConnect 4.2 et 5.2 129
 - migration vers jConnect 5.x 128
- argument
 - Debug 147
- avertissement
 - messages SQL 151

B

- base de données
 - JNDI for naming 84
 - stockage des objets Java comme données de colonne 74

C

- capture
 - communications TDS 104
- chargement de classe dynamique (DCL) 79
- classe
 - convertisseur de jeux de caractères 32
 - convertisseur de jeux de caractères PureConverter 32

- convertisseur de jeux de caractères
 - TruncationConverter 32
- convertisseur de jeux de caractères, sélection 33
- Debug 100
- PureConverter 32
- TruncationConverter 32, 37
- colonne
 - mise à jour dans des jeux de résultats curseur 51
 - suppression dans des jeux de résultats curseur 50
- connexion
 - à un serveur avec JNDI 21
 - à une passerelle Cascade 144
 - erreurs 107
 - passerelle refusée 106
 - pooling 88
- conventions xii
 - syntaxiques xii
 - typographiques xii
- conversion de jeu de caractères
 - amélioration des performances 116
- curseur 46
 - création 47
 - jeu de résultats 50
 - mise à jour et suppression par position avec des méthodes JDBC 1.x 50
 - performances 124
 - utilisation avec PreparedStatement 54

D

- DCL, chargement de classe dynamique 79
- débogage
 - activation 101
 - défini dans la variable d'environnement CLASSPATH 101
 - désactivation 101
 - jConnect 100
 - méthodes 102
 - obtention d'une instance de la classe de débogage

- 100
- désérialisation 81
- documentation
 - jConnect ix
- données
 - image 61, 62, 63
 - image et méthodes publiques de la classe TextPointer 61

E

- encapsulation
 - argument du servlet d' 147
 - configuration système pour le servlet d'encapsulation 146
 - installation de servlets 146
 - reprise des sessions TDS 149
 - suivi
 - des sessions 148
- encapsulation par TDS 4, 136
 - capture des communications 104
- erreur
 - de connexion 106, 107
 - procédure stockée 109
- euro
 - symbole 36
- exception
 - messages SQL 151
- extension Sybase
 - modifications 131

F

- fichier
 - préchargement de fichiers JAR 82
- fonctionnalités avancées 66

H

- HD (haute disponibilité) 38

I

- instruction
 - Compute 96
 - PreparedStatement et utilisation de curseurs 54
- interface
 - JDBC 2
 - SybEventHandler 66
 - SybMessageHandler 70
- internationalisation 32

J

- jConnect
 - amélioration des performances 114
 - appel 10
 - configuration 6
 - définition des propriétés de connexion 12
 - présentation 4
 - problèmes de mémoire dans les applications 108
 - résolution des incidents 100
 - utilisation des curseurs 46
- jConnect 4.x
 - jeu de résultats SCROLL_INSENSITIVE 55
- JDBC
 - interfaces 2
 - présentation 2
 - restrictions, limitations et non-conformités par
 - rapport aux normes 95
 - types de pilotes 2
- JDBC 2.0
 - extensions standard 83
 - Optional Package, support des extensions 83
- jeu de caractères
 - amélioration des performances lors de la conversion 34
 - définition 33
 - jeux multi-octets supportés 34
 - jeux supportés 34
- jeu de caractères multi-octets
 - classes de convertisseur 32
- jeu de résultats
 - et limitations TYPE_SCROLL_INSENSITIVE 55
 - SCROLL_INSENSITIVE dans jConnect 4.x 55
- jeu de résultats curseur
 - méthodes de mise à jour de la base de données 51

mise à jour de colonnes 51
 mise à jour et suppression par position avec des
 méthodes JDBC 2.0 51
 mise à jour par position 50
 suppression d'une ligne 53
 suppression par position 50
JNDI
 informations contextuelles 24
 utilisation 21
JNDI for naming databases 84

L

Lightweight Directory Access Protocol (LDAP) 22
 ligne
 insertion 53
 suppression dans un jeu de résultats curseur 53
 localisation 32

M

manuel
 lecteurs de ce manuel viii
 mémoire
 problèmes d'utilisation 108
 message d'erreur
 exceptions et avertissements SQL 151
 exemple de routine de gestion 72
 gestion 69
 gestion personnalisée 70
 intallation d'une routine de gestion 71
 spécifique de Sybase 69
 messages
 erreur SQL 151
 métadonnées
 accès 45
 mise en oeuvre sur le serveur 46
 USE_METADATA 17
 méthode
 rs.getBytes() 65
 setRemotePassword() 44
 Statement.cancel() 9
 mise à jour
 à partir du jeu de résultats d'une procédure stockée

 60
 par position avec des méthodes JDBC 1.x 50
 par position avec des méthodes JDBC 2.0 51
 mise à jour batch 59
 procédures stockées 58
 mise à jour de la base à partir du jeu de résultats 60
 multithread
 réglage 95

N

notification d'événement 66
 exemple 67

O

objet Java
 stockage comme données de colonne 74
 stockage de données de colonne dans 74
 objet java
 conditions préalables au stockage comme données de
 colonne 75

P

passerelle
 Cascade 139, 140
 configuration 137
 connexion refusée 106
 encapsulation 136
 Open Server Gateway 20
 passerelle Cascade
 configuration 137
 définition d'une connexion à 144
 démarrage 140
 installation (Cascade) 140
 test 141
 performances
 amélioration 114
 du curseur 124
 pilote
 JDBC 2, 3
 pont JDBC-ODBC 2

- propriétés 12
- protocole réseau/tout Java 3
- types de pilotes JDBC 2
- pont
 - JDBC-ODBC 2
- pooling
 - connexions 88
- procédure stockée 60
 - erreurs 109
 - exécution 97
- programme exemple
 - jConnect 174
- programme utilitaire
 - IsqlApp 172
- propriété
 - téléchargement 12
- propriété de connexion
 - APPLICATIONNAME 13
 - CANCEL_ALL 6, 9, 13
 - CHARSET 6, 13
 - CHARSET (définition) 33
 - CHARSET_CONVERTER 6
 - CHARSET_CONVERTER_CLASS 13, 33
 - CONNECTION_FAILOVER 13, 21
 - définition 12
 - définition dans l'URL 19
 - DYNAMIC_PREPARE 13
 - EXPIRESTRING 13
 - HOSTNAME 13
 - HOSTPROC 13
 - IGNORE_DONE_IN_PROC 14
 - JCONNECT_VERSION 9, 14
 - LANGUAGE 6, 14
 - LANGUAGE_CURSOR 14, 124
 - LITERAL_PARAMS 14
 - PACKETSIZE 14
 - password 14
 - PROTOCOL_CAPTURE 14
 - PROXY 15
 - REMOTEPWD 15
 - REPEAT_READ 15, 115
 - REQUEST_HA_SESSION 15
 - SELECT_OPENS_CURSOR 16
 - SERIALIZE_REQUESTS 16
 - SERVICENAME 16, 19
 - SESSION_ID 16

- SESSION_TIMEOUT 16
- SQLNITSTRING 17
- STREAM_CACHE_SIZE 17
- SYB SOCKET_FACTORY 17
- USE_METADATA 17
- user 17
- VERSIONSTRING 17
- propriété système
 - jdbc.drivers 10
- protocole
 - natif/tout Java 3
- protocole natif/tout Java 3
- protocole réseau/tout Java 2

R

- reprise
 - sessions TDS 149
- RPC
 - interserveur 44

S

- servlet 136, 147
 - argument SkipDoneProc 147
 - argument TdsResponseSize 147
 - argument TdsSessionIdleTimeout 147
 - encapsulation 136
- servlet d'encapsulation 146
- suppression
 - de colonnes dans des jeux de résultats curseur 50
 - par position avec des méthodes JDBC 1.x 50
 - par position avec des méthodes JDBC 2.0 51
- symbole
 - euro 36

T

- TDS encapsulée 148
- Technical Library x
- TextPointer 62
- TextPointer.sendData
 - mise à jour 63

TextPointer.sendData()
 mise à jour 62
transactions distribuées
 support 90
type de données
 Time, Date et Timestamp 64

U

URL
 paramètres de propriété de connexion 19
 syntaxe 18

V

variable d'environnement
 CLASSPATH 101

X

XAServer 90

