



Sybase® jConnect for JDBC™
Referenzhandbuch für Programmierer

jConnect for JDBC

Version 4.2 und 5.2

Dokument-ID: 38165-01-0520-01

Letzte Überarbeitung: Oktober 1999

Copyright © 1989-1999 Sybase Inc. Alle Rechte vorbehalten.

Dieses Handbuch ist Bestandteil der Dokumentation des Sybase-Datenbanksystems und jeder weiteren Version, sofern in neueren Ausgaben oder technischen Hinweisen nicht anders angegeben. Änderungen sind ohne vorherige Ankündigung jederzeit möglich. Die hierin beschriebene Software unterliegt einer Lizenzvereinbarung und darf nur im Rahmen der darin enthaltenen Bestimmungen eingesetzt oder kopiert werden.

Kunden in den USA und Kanada können weitere Dokumentationsteile anfordern. Wenden Sie sich dazu an die Abteilung Customer Fulfillment unter der Telefonnummer (800) 685-8225 und der Faxnummer (617) 229-9845.

Kunden außerhalb der USA mit einer amerikanischen Lizenz können mit unserer Abteilung Customer Fulfillment über Telefax oder mit Telesales unter Tel. 0130-73 91 87 (Deutschland) Kontakt aufnehmen. Alle anderen internationalen Kunden sollten sich an ihre Sybase Geschäftsstelle oder an ihren örtlichen Vertriebsbeauftragten wenden. Upgrades sind erst nach dem regulären Erscheinungsdatum der Softwareversion erhältlich. Kein Teil dieses Dokuments darf ohne vorherige schriftliche Genehmigung der Sybase Inc. in irgendeiner Form, sei es elektronisch, mechanisch, manuell, optisch oder auf sonstige Weise, fotokopiert, reproduziert oder in eine andere Sprache übersetzt werden.

Sybase, das Sybase-Logo, 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 Designor, S-Designor, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, 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 und XP Server sind Warenzeichen von Sybase Inc. 9/99

Unicode und das Unicode-Logo sind eingetragene Warenzeichen von Unicode Inc.

Alle anderen Unternehmens- oder Produktbezeichnungen sind Warenzeichen des jeweiligen Eigentümers.

Gebrauch, Vervielfältigung oder Veröffentlichung durch die US-Regierung sind Gegenstand der Beschränkungen DFARS 52.227-7013 (c)(1)(ii) für das Verteidigungsministerium und FAR 52.227-19(a)-(d) für zivile Behörden.

Sybase Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Inhaltsverzeichnis

Über dieses Handbuch	7
KAPITEL 1	Einleitung
	Was versteht man unter JDBC?
	Was ist jConnect?
KAPITEL 2	Informationen für die Programmierung
	jConnect einrichten
	Version von jConnect setzen
	Aufruf des jConnect-Treibers
	Verbindung einrichten
	Verbindungseigenschaften festlegen
	Verbindung mit Adaptive Server Enterprise
	Verbindung mit Adaptive Server Anywhere
	Verbindung mit einem Server über JNDI aufnehmen
	Benutzerdefinierte Socket-Plug-Ins implementieren
	SYB SOCKET_FACTORY, Verbindungseigenschaft
	Einen angepassten Socket erstellen und konfigurieren
	Internationalisierungs- und Sprachanpassungsfragen
	Zeichensatzkonverter von jConnect
	Mit Datenbanken arbeiten
	Unterstützung für Hochverfügbarkeits-Notumschaltung implementieren
	Remote Procedure Call von Server zu Server ausführen
	Zugriff auf Datenbank-Metadaten
	Cursor mit Ergebnismengen verwenden
	Unterstützung von Aktualisierungen in Form von Anweisungsfolgen
	Datenbank aus der Ergebnismenge einer gespeicherten Prozedur aktualisieren
	Mit Datenbanken arbeiten
	Erweiterte Funktionen implementieren
	Ereignismeldungen benutzen

Umgang mit Fehlermeldungen	71
Java-Objekte als Spaltendaten in einer Tabelle speichern	76
Dynamisches Laden von Klassen	81
Unterstützung für die Erweiterungen des JDBC 2.0- Optionspakets	85
Umgang mit Beschränkungen, Einschränkungen und Abweichungen von den JDBC-Standards	97
Anpassungen für Multithreading	97
<i>ResultSet.setCursorName()</i> verwenden	97
<i>setLong()</i> mit großen Parameterwerten verwenden	98
COMPUTE-Anweisungen verwenden	98
gespeicherte Prozeduren ausführen	99

KAPITEL 3

Fehlersuche	101
Debugging mit jConnect	102
Instanz der Debug-Klasse erhalten	102
Fehlersuchfunktion in Ihrer Anwendung einschalten	103
Fehlersuchfunktion in Ihrer Anwendung ausschalten	103
CLASSPATH für die Fehlersuche einstellen	103
Debug-Methoden verwenden	104
TDS-Kommunikation aufzeichnen	106
Verbindungseigenschaft PROTOCOL_CAPTURE	106
Die Methoden pause() und resume() in der Capture-Klasse ..	107
Fehler bei erfolglosen Verbindungen	108
Gateway-Verbindung abgelehnt	108
Verbindung mit 4.9.2 SQL Server unmöglich	109
Speichernutzung in jConnect-Anwendungen	110
Fehler bei gespeicherten Prozeduren	111
RPC gibt weniger Ausgabeparameter zurück als registriert ..	111
Fetch/State-Fehler bei der Rückgabe von Ausgabeparametern durch die gespeicherte Prozedur	111
Gespeicherte Prozedur im unverketteten Transaktionsmodus ausgeführt	111
Fehler bei der Implementierung des benutzerdefinierten Sockets	113

KAPITEL 4

Performance und Optimierung	115
Performance von jConnect steigern	116
BigDecimal-Skalierungsfaktor	117
REPEAT_READ, Verbindungseigenschaft	117
Zeichensatzkonvertierung	118
Optimierung der Performance für Prepared Statements in Dynamic SQL	119
Wahl zwischen Prepared Statements und gespeicherte	

	Prozeduren	120
	Prepared Statements in portierbaren Anwendungen	121
	Prepared Statements in Anwendungen mit jConnect- Erweiterungen	121
	Connection.prepareStatement()	123
	DYNAMIC_PREPARE, Verbindungseigenschaft	123
	SybConnection.prepareStatement()	125
	Cursor-Performance	126
	LANGUAGE_CURSOR, Verbindungseigenschaft	126
KAPITEL 5	jConnect-Anwendungen migrieren	129
	jConnect-Anwendungen migrieren	130
	Anwendungen auf jConnect 4.1 migrieren	130
	Anwendungen auf jConnect 5.x migrieren	130
	Anwendungen auf jConnect 4.2 und 5.2 migrieren	131
	Änderungen bei Sybase-Erweiterungen	133
	Änderungsbeispiel	133
	Geänderte Methodennamen	134
	Debug-Klasse einrichten	134
KAPITEL 6	Webserver Gateways	137
	Hinweise zu Webserver Gateways	138
	TDS-Tunnelling	138
	jConnect und Gateway-Konfiguration	139
	Cascade Gateway verwenden	141
	Cascade Gateway installieren	142
	Cascade Gateway starten	142
	Cascade Gateway testen	144
	Die Datei index.html lesen	145
	Muster-IsqI-Applet ausführen	145
	Verbindung zum Cascade Gateway definieren	146
	TDS-Tunnelling Servlet benutzen	147
	TDS-tunnelling-Servlet, erforderliche Systemausstattung	148
	Servlet installieren	148
	Servlet aufrufen	150
	Überwachung aktiver TDS-Sitzungen	150
	TDS-Sitzung wieder aufnehmen	151
	TDS-tunnelling und Netscape Enterprise Server 3.5.1 auf Solaris verwenden	151
ANHANG A	SQLExceptions und SQLWarnings Meldungsübersicht	153

ANHANG B	jConnect Beispielprogramme	171
	IsqlApp ausführen	172
	jConnect-Beispielprogramme und -Code ausführen	175
	Beispielanwendungen	175
	Beispiel-Programmcode	176
Index		179

Über dieses Handbuch

Die Dokumentation *Sybase jConnect for JDBC Referenzhandbuch für Programmierer* beschreibt das Produkt jConnect for JDBC und erklärt, wie auf Daten zugegriffen wird, die in relationalen Datenbank-Management-Systemen gespeichert sind.

Zielgruppe

Dieses Handbuch ist für Programmierer von Datenbank Anwendungen bestimmt, die die Programmiersprache Java, JDBC und Transact-SQL[®], die von Sybase erweiterte Version der Structured Query Language, bereits kennen.

Weitere Handbücher

Folgende Dokumentationen enthalten gegebenenfalls weitere nützliche Hinweise:

- *Sybase jConnect for JDBC Installationshandbuch*
- *Versionshinweise für Sybase jConnect for JDBC*
- Die Javadoc-Dokumentation der jConnect-Erweiterungen zu JDBC. Das Java-Entwicklungskit (JDK) von JavaSoft enthält ein Javadoc-Skript, mit dem Kommentare aus Quellcode-Dateien extrahiert werden können. Dieses Skript wurde verwendet, um eine Dokumentation der Pakete, Klassen und Methoden von jConnect aus jConnect-Quellcode-Dateien zu extrahieren. Wenn Sie jConnect installieren, wird die Javadoc-Information in das Verzeichnis *javadocs* kopiert.

Installation_directory/docs/en/javadocs

Andere Informationsquellen

Benutzen Sie die Sybase Technical Library CD (früher: SyBooks) und die Technical Library Website, wenn Sie mehr über das Produkt erfahren wollen:

- Die Technical Library CD wird mit der von Ihnen erworbenen Software mitgeliefert und enthält Handbücher und Technische Dokumentationen zur Software. Der DynaText-Browser, der auf der Technical Library CD enthalten ist, ermöglicht den Zugriff auf technische Informationen über die Software in einem aktualisierten, leicht bedienbaren Format.

Anweisungen zum Installieren und Starten der Technical Library finden Sie in der Dokumentation *Installationsanweisung für die Technical Library* in Ihrem Dokumentationspaket.

- Die Website Technical Library enthält eine Sektion "Product Manuals", in der eine HTML-Version der Technical Library CD gespeichert ist, die über einen normalen Web-Browser abgerufen werden kann. Außerdem finden Sie dort Verknüpfungen mit der Website "Technical Documents" (früher als "Tech Info Library" bezeichnet), zur Webseite "Solved Cases" und zu den Newsgroups Sybase/Powersoft.

Um auf die Website "Technical Library" zuzugreifen, rufen Sie die Adresse support.sybase.com auf, klicken auf das Register "Electronic Support Services" und wählen eine Verknüpfung unter dem Titel "Technical Library".

Sybase-Zertifizierungen auf dem Web

Die technischen Informationen auf der Sybase-Website werden in regelmäßigen Abständen aktualisiert.

So finden Sie die neuesten Informationen zu Produkt-Zertifizierungen und/oder den EBF-Rollups:

- 1 Schalten Sie in Ihrem Web-Browser auf die Technical Documents Library unter folgender Webadresse:
bei techinfo.sybase.com
- 2 Klicken Sie im Abschnitt "Browse" auf den Eintrag "What's Hot".
- 3 Informieren Sie sich über das Thema Ihres Interesses: "Hot Docs" (Aktuelle Dokumente) zu verschiedenen Themen oder "Hot Links" (Verknüpfungen) zu den "Technical News" (Aktuelle technische Informationen), "Certification Reports" (Leistungsberichte), "Partner Certifications" (Informationen über Partner), etc.

Wenn Sie ein registrierter Support**Plus**-Benutzer sind:

- 1 Schalten Sie in Ihrem Web-Browser auf die Technical Documents Library unter folgender Webadresse:
bei techninfo.sybase.com
- 2 Klicken Sie im Abschnitt "Browse" auf den Eintrag "What's Hot".
- 3 Klicken Sie auf "EBF Rollups".
Sie können die EBF-Rollups über die Technical Documents abrufen und EBFs mit Electronic Software Distribution (ESD) herunterladen.
- 4 Folgen Sie den zu den Support**Plus**SM Online Services-Einträgen gehörenden Anweisungen.

Wenn Sie kein registrierter Support**Plus**-Benutzer sind und einer werden wollen:

Sie können die Registrierung direkt anhand der Anleitungen der Website vornehmen.

Um Support**Plus** benutzen zu können, brauchen Sie:

- Einen Web-Browser, der den Secure Sockets Layer (SSL) unterstützt, z.B. Netscape Navigator 1.2 oder später
- Eine aktive Support-Lizenz
- Eine bestimmte Kontaktperson des technischen Kundendiensts
- Ihre Benutzer-ID und das Kennwort

Gleichgültig ob Sie registrierter Support**Plus**-Benutzer sind oder nicht:

Sie können die Technical Documents von Sybase benutzen. Die Certification Reports gehören zu den auf dieser Site dokumentierten Funktionen.

- 1 Schalten Sie in Ihrem Web-Browser auf die Technical Documents Library unter folgender Webadresse:
bei techninfo.sybase.com
- 2 Klicken Sie im Abschnitt "Browse" auf den Eintrag "What's Hot".
- 3 Klicken Sie auf das Thema, für das Sie sich interessieren.

Konventionen

In diesem Handbuch werden folgende Konventionen für die Syntaxangaben und die Schreibweise eingehalten:

- Klassen, Schnittstellen, Methoden und Pakete werden in **Helvetica fett** geschrieben, wenn sie sich im Text eines Absatzes befinden. Beispiel:

Klasse **SybConnection**

Schnittstelle **SybEventHandler**

Methode **setBinaryStream()**

Paket **com.sybase.jdbcx**

- Objekte, Instanzen und Parameternamen werden kursiv geschrieben. Beispiel:

“Im folgenden Beispiel ist *ctx* ein **DirContext**-Objekt.”

“*eventHdlr* ist eine Instanz der Klasse **SybEventHandler**, die Sie implementieren.”

“Der Parameter *classes* ist eine Zeichenfolge, die spezielle Klassen auflistet, die Sie debuggen wollen.”

- Programmcode wird in Courierschrift dargestellt. Variable in Code-Fragmenten (z.B. Wörter, die für Werte stehen, die Sie ausfüllen) sind kursiv dargestellt. Beispiel:

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

Wenn Sie Hilfe brauchen

Für jede Sybase-Installation, für die ein Kundendienstvertrag abgeschlossen wurde, werden eine oder mehrere Personen bestimmt, die sich im Bedarfsfall mit dem Technischen Kundendienst von Sybase in Verbindung setzen. Wenn Sie ein Problem nicht mit Hilfe der Handbücher oder der Online-Hilfe lösen können, wenden Sie sich an diese Person, damit sie den technischen Kundendienst von Sybase informiert.

Wenn Sie Hilfe brauchen

Einleitung

Dieses Kapitel enthält eine Einführung in jConnect for JDBC und eine Beschreibung der Konzepte und Komponenten dieser Software.

Dieses Kapitel enthält die folgenden Abschnitte:

Name	Seite
Was versteht man unter JDBC?	2
Was ist jConnect?	4

Was versteht man unter JDBC?

JDBC (Java Database Connectivity) ist eine von der Java Software Division von Sun Microsystems, Inc. entwickelte Standardspezifikation für eine Anwendungsprogrammschnittstelle (Application Program Interface, API), über die Java-Anwendungen auf viele Datenbank-Management-Systeme zugreifen, die mit der Structured Query Language (SQL) arbeiten. Der JDBC-Treibermanager arbeitet mit mehreren Treibern, die eine Verbindung mit unterschiedlichen Datenbanken herstellen.

Die Standard-JDBC-API enthält eine Gruppe von Java-Interfaces, damit Verbindungen zu Datenbanken eingerichtet, SQL-Befehle abgearbeitet und Ergebnisse verarbeitet werden können. Die Schnittstellen werden in [Tabelle 1-1](#) beschrieben.

Tabelle 1-1: JDBC-Interfaces

Interface	Beschreibung
java.sql.Driver	Ermittelt den Standort eines Treibers für einen Datenbank-URL
java.sql.Connection	Verbindet mit einer bestimmten Datenbank
java.sql.Statement	Führt SQL-Anweisungen aus
java.sql.PreparedStatement	Verarbeitet parametergesteuerte SQL-Anweisungen
java.sql.CallableStatement	Verarbeitet Aufrufe von gespeicherten Prozeduren in Datenbanken
java.sql.ResultSet	Übernimmt die Ergebnismenge von SQL-Anweisungen
java.sql.DatabaseMetaData	Wird verwendet, um auf nützliche Informationen über DBMS und Datenbank einer Verbindung zuzugreifen
java.sql.ResultSetMetaData	Wird verwendet, um auf nützliche Informationen über die Attribute eines ResultSets zuzugreifen

Jedes relationale Datenbank-Management-System benötigt einen Treiber, um diese Interfaces integrieren zu können. Alle JDBC-Programmaufrufe werden an den JDBC-Treibermanager geschickt, der sie an den jeweils richtigen Treiber weiterleitet.

Es gibt vier Arten von JDBC-Treibern:

- *Typ 1 JDBC-ODBC Bridge* – Übersetzt JDBC-Aufrufe in ODBC-Aufrufe, ruft sie auf und übergibt sie an einen ODBC-Treiber. ODBC-Software muss teilweise auf dem Clientrechner untergebracht sein. Zum Teil befindet sich auch Client-Datenbankcode auf dem Clientrechner.
- *Typ 2 eigene-API Teil-Java-Treiber* – Konvertiert JDBC-Aufrufe in Datenbank-spezifische Aufrufe. Dieser Treiber, der direkt mit dem Datenbankserver kommuniziert, benötigt auch einen Teil des Binärcodes auf dem Clientrechner.
- *Typ 3 Net-Protocol-All-Java-Treiber* – Kommuniziert mit einem Mittelschicht-Server, indem er ein DBMS-unabhängiges Netzprotokoll verwendet. Ein Gateway in der Mittelschicht konvertiert dann die Anforderung in ein herstellerspezifisches Protokoll.
- *Typ 4 Native-Protocol-All-Java-Treiber* – Konvertiert JDBC-Aufrufe in das Hersteller-spezifische DBMS-Protokoll und ermöglicht Client-Anwendungen eine direkte Kommunikation mit dem Datenbankserver.

Was ist jConnect?

jConnect ist der von Sybase entwickelte Hochleistungs-JDBC-Treiber. Er hat eine Doppelfunktion:

- Er ist ein Net-Protocol-All-Java-Treiber in einem Dreischichtenmodell und
- Ein Native-Protocol-All-Java-Treiber in einem Zweischichtenmodell.

Das von jConnect benutzte Protokoll ist TDS 5.0 (Tabular Data Stream™, Version 5), das native Protokoll für Anwendungen unter Adaptive Server® und Open Server™. jConnect implementiert den JDBC-Standard für eine optimale Datenbankbindung an die komplette Sybase-Produktfamilie, d.h. Zugang zu über 25 Enterprise- und Legacy-Systemen, darunter:

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

Hinweis Seitdem der Name von Sybase SQL Server™ in Adaptive Server Enterprise geändert wurde, verweist Sybase über die Namen Adaptive Server und Adaptive Server Enterprise allgemein auf alle unterstützten Versionen von Sybase SQL Server und Adaptive Server Enterprise.

jConnect for JDBC kann darüber hinaus auf Oracle, AS/400 und andere Datenquellen zugreifen, indem als Transport Sybase DirectConnect™ verwendet wird.

Unter gewissen Voraussetzungen weicht die jConnect-Implementierung von JDBC von den Spezifikationen JDBC 1.x oder 2.x ab. Weitere Hinweise finden Sie unter ["Umgang mit Beschränkungen, Einschränkungen und Abweichungen von den JDBC-Standards"](#) auf Seite 97.

Informationen für die Programmierung

In diesem Kapitel werden die Grundelemente und die Programmiererfordernisse beschrieben, die für jConnect for JDBC gelten. Es wird erklärt, wie der jConnect-Treiber aufgerufen wird, Verbindungseigenschaften gesetzt werden und eine Verbindung mit einem Datenbankserver hergestellt wird. Außerdem werden Hinweise zur Verwendung diverser jConnect-Funktionen gegeben.

Hinweis Informationen über JDBC-Programmierung finden Sie bei <http://java.sun.com/jdbc>.

Wenn Sie die Dokumentation *JDBC-Handbuch: Erste Orientierung* für JDBC 1.0 benötigen, verbinden Sie sich mit folgender Adresse:
bei <http://java.sun.com/products/jdk/1.1/docs/guide/jdbc>.

So greifen Sie auf die Dokumentation *JDBC-Handbuch: Erste Orientierung* für JDBC 2.0 zu. Gehen Sie auf die folgende Adresse:
bei <http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/>.

Folgende Themen werden in diesem Kapitel behandelt:

Name	Seite
jConnect einrichten	6
Verbindung einrichten	12
Benutzerdefinierte Socket-Plug-Ins implementieren	28
Internationalisierungs- und Sprachanpassungsfragen	34
Mit Datenbanken arbeiten	40
Erweiterte Funktionen implementieren	68
Umgang mit Beschränkungen, Einschränkungen und Abweichungen von den JDBC-Standards	97

jConnect einrichten

In diesem Abschnitt werden die Arbeitsschritte beschrieben, die Sie durchführen müssen, bevor Sie mit jConnect arbeiten.

Version von jConnect setzen

Es gibt mehrere Versionen von jConnect. Mit der Versionseinstellung legen Sie folgende Elemente fest:

- Den Standardwert für die Verbindungseigenschaft LANGUAGE
- Die versionsspezifisch verfügbaren Funktionen
- Den Standard-Zeichensatz, wenn in der Verbindungseigenschaft CHARSET kein Zeichensatz festgelegt wird
- Den Standardwert für die Verbindungseigenschaft CHARSET_CONVERTER
- Den Standardwert für die Verbindungseigenschaft CANCEL_ALL. Sie wird benutzt, um das Verhalten von **Statement.cancel()** festzulegen. Damit wird standardmäßig das Objekt abgebrochen, für das die Eigenschaft aufgerufen wird, sowie andere **Statement**-Objekte, die mit der Ausführung begonnen haben und auf Ergebnisse warten.

[Tabelle 2-1](#) zeigt die verfügbaren Versionseinstellungen und ihre Funktionen.

Tabelle 2-1: jConnect-Versionseinstellungen und ihre Merkmale

Versions-Konstante	Merkmale	Kommentare
VERSION_5	<ul style="list-style-type: none"> • Der Standardwert der LANGUAGE-Verbindungseigenschaft ist "null". • Wenn die CHARSET-Verbindungseigenschaft keinen Zeichensatz festlegt, verwendet jConnect den Standardzeichensatz der Datenbank. Der Standardwert für CHARSET_CONVERTER ist die Klasse PureConverter. • Standardmäßig bricht Statement.cancel() nur das Statement-Objekt ab, für das diese Methode aufgerufen wird. • JDBC-2.0-Methoden können benutzt werden, um Java-Objekte als Spaltendaten zu speichern und abzurufen. 	<p>Für jConnect Version 5.x ist der Standardwert VERSION_5.</p> <p>Weitere Kommentare finden Sie bei VERSION_4.</p>
VERSION_4	<ul style="list-style-type: none"> • Der Standardwert der LANGUAGE-Verbindungseigenschaft ist "null". • Wenn die CHARSET-Verbindungseigenschaft keinen Zeichensatz festlegt, verwendet jConnect den Standardzeichensatz der Datenbank. Der Standardwert für CHARSET_CONVERTER ist die Klasse PureConverter. • Standardmäßig bricht Statement.cancel() nur das Statement-Objekt ab, für das diese Methode aufgerufen wird. • JDBC-2.0-Methoden können benutzt werden, um Java-Objekte als Spaltendaten zu speichern und abzurufen. 	<p>Für jConnect Version 4.x und früher ist der Standardwert VERSION_2.</p> <p>Servermeldungen werden entsprechend der Spracheinstellung in Ihrer lokalen Umgebung angepasst. Folgende Sprachen werden unterstützt: Chinesisch, US Englisch, Französisch, Deutsch, Japanisch, Koreanisch, Portugiesisch und Spanisch.</p> <p>Das Standardverhalten für Statement.cancel() ist JDBC-kompatibel.</p> <p>Legen Sie das Verhalten von Statement.cancel() mit Hilfe von CANCEL_ALL fest. Siehe "CANCEL_ALL, Verbindungseigenschaft" auf Seite 10.</p> <p>Hinweise zu Java-Objekten als Spaltendaten finden Sie unter "Java-Objekte als Spaltendaten in einer Tabelle speichern" auf Seite 76.</p>

Versions-Konstante	Merkmale	Kommentare
VERSION_3	<ul style="list-style-type: none"> Der Standardwert der LANGUAGE-Verbindungseigenschaft ist us_english. Wenn die CHARSET-Verbindungseigenschaft keinen Zeichensatz liefert, wird der Standard-Zeichensatz der Datenbank verwendet. Der Standardwert für CHARSET_CONVERTER ist die Klasse PureConverter. Standardmäßig bricht Statement.cancel() das Objekt ab, für das diese Methode aufgerufen wird, und alle anderen Statement-Objekte, die mit der Ausführung begonnen haben und auf Ergebnisse warten. 	<p>Der Standardwert ist VERSION_2. Siehe Kommentare für VERSION_2.</p>
VERSION_2	<ul style="list-style-type: none"> Der Standardwert der LANGUAGE-Verbindungseigenschaft ist us_english. Wenn die CHARSET-Verbindungseigenschaft keinen Zeichensatz liefert, wird der Standard-Zeichensatz auf iso_1 gesetzt. Der Standardwert für CHARSET_CONVERTER ist die Klasse TruncationConverter, wenn nicht die Verbindungseigenschaft CHARSET einen Mehrbyte- oder 8-Bit-Zeichensatz festlegt, in welchem Fall der Standard-CHARSET_CONVERTER die Klasse PureConverter ist. Standardmäßig bricht Statement.cancel() das Objekt ab, für das diese Methode aufgerufen wird, und alle anderen Statement-Objekte, die mit der Ausführung begonnen haben und auf Ergebnisse warten. 	<p>Die Standard-Versionseinstellung für jConnect Version 2.x ist VERSION_2.</p> <hr/> <p>Hinweis VERSION_5 ist die Standard-Versionseinstellung für jConnect Version 5.x.</p> <hr/> <p>Die LANGUAGE-Verbindungseigenschaft legt die Sprache fest, in der Meldungen von jConnect und dem Server angezeigt werden.</p> <p>Hinweise zu den Verbindungsklassen CHARSET und CHARSET_CONVERTER finden Sie unter "Zeichensatzkonverter von jConnect" auf Seite 34.</p> <p>Das VERSION_2-Standardverhalten von Statement.cancel() ist nicht JDBC-kompatibel. Legen Sie das Verhalten von Statement.cancel() mit Hilfe von CANCEL_ALL fest. Siehe "CANCEL_ALL, Verbindungseigenschaft" auf Seite 10.</p>

Die Versionswerte sind Konstantenwerte aus der **SybDriver**-Klasse. Für die Referenzierung der Versionskonstanten verwenden Sie folgende Syntax:

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

Mit **SybDriver.setVersion()** setzen Sie die jConnect-Version. Das folgende Codebeispiel zeigt, wie der jConnect-Treiber geladen und die Version gesetzt wird.

Für **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);
```

Für **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);
```

Sie können **setVersion()** mehrfach aufrufen, um die Versionseinstellung zu ändern. Neue Verbindungen erben das Verhalten, das der Versionseinstellung beim Aufbau der Verbindung zugeordnet ist. Wenn die Versionseinstellung während einer Sitzung geändert wird, beeinflusst dies die aktuelle Verbindung nicht.

Wie im folgenden Abschnitt beschrieben, können Sie mit **JCONNECT_VERSION** die Versionseinstellung **SybDriver** überschreiben und eine andere Versionseinstellung für eine bestimmte Verbindung angeben.

JCONNECT_VERSION, Verbindungseigenschaft

Stellen Sie mit **JCONNECT_VERSION** die Version für eine bestimmte Sitzung ein. Sie können **JCONNECT_VERSION** auf einen Ganzzahlwert von "2," "3," "4" oder "5" setzen, abhängig von den gewünschten Merkmalen (siehe [Tabelle 2-1](#)).

CANCEL_ALL, Verbindungseigenschaft

CANCEL_ALL ist eine mit Booleschen Werten operierende Eigenschaft für die Festlegung des Verhaltens der Methode **Statement.cancel()**.

Hinweis In jConnect Version 4.0 und früher ist der Standardwert für CANCEL_ALL "true". Wenn Sie in jConnect Version 4.1 und höher die Verbindungseigenschaft JCONNECT_VERSION auf "4" oder höher setzen, um der JDBC-Spezifikation zu entsprechen, ist die Standardeinstellung für CANCEL_ALL "false".

Die Einstellungen für CANCEL_ALL haben folgende Auswirkungen auf **Statement.cancel()**:

- Wenn CANCEL_ALL "false" ist, wird durch den Aufruf von **Statement.cancel()** nur das **Statement**-Objekt abgebrochen, für das diese Methode aufgerufen wird. Falls **stmtA** ein **Statement**-Objekt ist, bricht **stmtA.cancel()** die Ausführung der SQL-Anweisung ab, die in **stmtA** in der Datenbank enthalten ist, es werden aber keine anderen Anweisungen betroffen. **stmtA** wird abgebrochen, gleichgültig, ob sich die Anweisung im Cache befindet und auf ihre Ausführung wartet, oder bereits ausgeführt ist und auf Ergebnisse wartet.
- Wenn CANCEL_ALL auf "true" gesetzt ist, bricht der Aufruf von **Statement.cancel()** nicht nur das Objekt ab, für das die Eigenschaft aufgerufen ist, sondern auch alle anderen **Statement**-Objekte in derselben Verbindung, die ausgeführt wurden und auf Ergebnisse warten.

Im folgenden Beispiel wird CANCEL_ALL auf "false" gesetzt. *props* ist ein **Properties**-Objekt für die Festlegung von Verbindungseigenschaften.

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

Hinweis Um die Ausführung aller **Statement**-Objekte in einer Verbindung abubrechen, auch wenn sie schon mit der Ausführung auf dem Server begonnen haben, verwenden Sie die Erweiterungsmethode **SybConnection.cancel()**.

Aufruf des jConnect-Treibers

Um den jConnect-Treiber von Sybase zu registrieren und aufzurufen, verwenden Sie eine der folgenden empfohlenen Methoden:

Methode 1

Für jConnect 4.x:

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

Für jConnect 5.x:

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

Methode 2

Fügen Sie den jConnect-Treiber der Systemeigenschaft **jdbc.drivers** hinzu. Bei der Initialisierung versucht die Klasse **DriverManager**, die Treiber zu laden, die in **jdbc.drivers** registriert sind. Dies ist eine weniger wirksame Lösung als der erstgenannte Ansatz. Sie können mehrere Treiber in dieser Eigenschaft registrieren, indem Sie sie durch einen Doppelpunkt (:) voneinander trennen. Das folgende Quellcode-Beispiel zeigt, wie Sie in einem Programm der Eigenschaft **jdbc.drivers** einen Treiber hinzufügen:

Für 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());
```

Für 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());
```

Hinweis **System.getProperties()** ist für Java-Applets nicht zulässig. Verwenden Sie stattdessen die Methode **Class.forName()**.

Verbindung einrichten

In diesem Abschnitt wird beschrieben, wie Sie eine Verbindung zu einer Datenbank auf Adaptive Server Enterprise oder Adaptive Server Anywhere mit jConnect aufbauen.

Verbindungseigenschaften festlegen

Tabelle 2-2 zeigt die Verbindungseigenschaften für jConnect und ihre Standardwerte. Sie müssen die Verbindungseigenschaften festlegen, bevor Sie eine Verbindung aufbauen.

Verbindungseigenschaften können auf zwei Arten gesetzt werden:

- Methode **DriverManager.getConnection()** in Ihrer Anwendung einsetzen
- Bei der Definition des URL

Hinweis Treiber-Verbindungseigenschaften, die im URL festgelegt wurden, überschreiben keine entsprechenden Verbindungseigenschaften, die in der Anwendung mit Hilfe der Methode **DriverManager.getConnection()** festgelegt wurden.

Um eine aktuelle Liste der Eigenschaften eines Treibers zu erhalten, verwenden Sie die Methode **Driver.getPropertyInfo(String url, Properties props)**, die ein Feld von **DriverPropertyInfo**-Objekten zurückgibt. Die Tabelle enthält folgende Objekte:

- Die Treibereigenschaften
- Die aktuellen Einstellungen, auf denen die Treibereigenschaften basieren
- Die übergebenen Informationen für URL und **props**

Die Namen der Treiber-Verbindungseigenschaften beachten nicht die Gross/Kleinschreibung (jConnect verwendet die Methode **String.equalsIgnoreCase(String)**, um Eigenschaftsnamen zu vergleichen).

Tabelle 2-2: Verbindungseigenschaften

Eigenschaft	Beschreibung	Standardwert
APPLICATIONNAME	Eine benutzerdefinierte Eigenschaft. Die Serverseite kann programmiert werden, den dieser Eigenschaft verliehenen Wert zu interpretieren.	Null

Eigenschaft	Beschreibung	Standardwert
CANCEL_ALL	Legt das Verhalten der Methode <code>Statement.cancel()</code> fest. Siehe "CANCEL_ALL, Verbindungseigenschaft" auf Seite 10.	Abhängig von der Versionseinstellung. (Siehe "Version von jConnect setzen" auf Seite 6).
CHARSET	Legt den Zeichensatz für Zeichenfolgen fest, die durch TDS durchgegeben werden. Wenn Sie CHARSET festlegen, muss der Zeichensatz mit dem in <code>syscharsets</code> aufgelisteten Zeichensatz übereinstimmen. Wenn er Null ist, verwendet jConnect den Standard-Zeichensatz des Servers.	Null
CHARSET_CONVERTER_CLASS	Sie können diese Eigenschaft verwenden, um die Zeichensatz-Konvertierungsklasse anzugeben, die jConnect verwenden soll. jConnect benutzt die Versionseinstellung in <code>SybDriver.setVersion()</code> , um zu ermitteln, welche Standard-Zeichensatzkonverterklasse verwendet werden soll. Weitere Hinweise finden Sie unter "Zeichensatz-Konverter auswählen" auf Seite 35.	Versionsabhängig
CONNECTION_FAILOVER	Für den Einsatz mit dem Java Naming and Directory Interface (JNDI). Siehe "CONNECTION_FAILOVER, Verbindungseigenschaft" auf Seite 25.	true
DYNAMIC_PREPARE	Legt fest, ob dynamische SQL-Prepared Statements in der Datenbank vorkompiliert werden. Siehe "DYNAMIC_PREPARE, Verbindungseigenschaft" auf Seite 123.	false
EXPIRESTRING	Eine Read-Only-Eigenschaft, die das Ablaufdatum der Lizenz enthält. Das Ablaufdatum ist auf "nie" gesetzt, außer bei Testversionen von jConnect.	Nie
HOSTNAME	Der Name des aktuellen Hosts.	Keiner
HOSTPROC	Identifiziert den Prozess der Anwendung auf dem Hostrechner.	Keiner
IGNORE_DONE_IN_PROC	Wenn diese Eigenschaft auf "true" gesetzt ist, werden Zwischenergebnisse von Aktualisierungen (wie in gespeicherten Prozeduren) nicht zurückgegeben. Es wird nur die endgültige Ergebnismenge zurückgegeben.	false

Eigenschaft	Beschreibung	Standardwert
JCONNECT_VERSION	Verwenden Sie diese Eigenschaft, um versionsspezifische Eigenschaften zu setzen. Siehe "JCONNECT_VERSION, Verbindungseigenschaft" auf Seite 9.	5
LANGUAGE	Setzen Sie diese Eigenschaft für Fehlermeldungen, die vom Server zurückgegeben werden, und für jConnect-Meldungen. Die hier definierte Sprache muss einer der in <i>syslanguages</i> registrierten Sprachen entsprechen.	Versionsabhängig Siehe "Version von jConnect setzen" auf Seite 6.
LANGUAGE_CURSOR	Setzen Sie diese Eigenschaft auf "true", wenn jConnect "language cursor" anstelle von "protocol cursor" verwenden soll. Siehe "Cursor-Performance" auf Seite 126.	false
LITERAL_PARAMS	Diese Eigenschaft wird nur mit Adaptive Server Anywhere verwendet, der verlangt, dass Sie Prepared Statement-Parameter als Literale senden. Bei allen anderen Sybase-Datenbanken kann diese Eigenschaft auf "false" gesetzt werden. Wenn diese Eigenschaft auf "true" gesetzt ist, werden alle durch die setXXX-Methoden im PreparedStatement-Schnittstelle festgelegten Parameter literal in die SQL-Anweisung eingefügt, wenn diese ausgeführt wird. Wenn die Eigenschaft auf "false" gesetzt wird, werden Parametermarkierungen in der SQL-Anweisung hinterlassen, und die Parameterwerte werden getrennt an den Server geschickt.	false
PACKETSIZE	Netzwerk-Paketgröße.	512
PASSWORD	Das Login-Kennwort. Diese Eigenschaft wird automatisch gesetzt, wenn die Methode getConnection(String, String, String) verwendet wird oder explizit, wenn getConnection(String, Props) verwendet wird.	Keiner
PROTOCOL_CAPTURE	Die Verbindungseigenschaft PROTOCOL_CAPTURE wird benutzt, um eine Datei für das Aufzeichnen einer TDS-Kommunikation zwischen einer Anwendung und Adaptive Server anzugeben.	Null

Eigenschaft	Beschreibung	Standardwert
PROXY	<p>Gateway-Adresse. Für das HTTP-Protokoll ist der URL: <i>http://host:port</i>.</p> <p>Wenn das HTTPS-Protokoll verwendet werden soll, das die Verschlüsselung unterstützt, benutzen Sie den URL <i>https://host:port/servlet_alias</i>.</p>	Keiner
REMOTEPWD	<p>Kennwörter von Fremdservern für den Zugriff über Remote Procedure Call von Server zu Server. Siehe "Remote Procedure Call von Server zu Server ausführen" auf Seite 45.</p>	Keiner
REPEAT_READ	<p>Legt fest, ob der Treiber Kopien der Spalten und Ausgabeparameter behält, damit Spalten unsortiert oder wiederholt ausgelesen werden können. Siehe "REPEAT_READ, Verbindungseigenschaft" auf Seite 117.</p>	true
REQUEST_HA_SESSION	<p>Diese Eigenschaft zeigt an, ob der verbindende Client eine HA-Notumschaltungs-Sitzung mit Adaptive Server Version 12 oder höher, konfiguriert für HA-Notumschaltung, beginnen will.</p> <p>Wenn diese Einstellung auf "true" gesetzt ist, versucht jConnect eine HA-Notumschaltungs-Anmeldung. Wenn Sie diese Verbindungseigenschaft nicht festlegen, startet keine HA-Notumschaltungs-Sitzung, auch wenn der Server für HA-Notumschaltung konfiguriert ist.</p> <p>Sie können die Eigenschaft nicht mehr zurücksetzen, sobald eine Verbindung aufgebaut ist.</p> <p>Wenn Sie größere Flexibilität beim Anfordern von HA-Notumschaltungs-Sitzungen wollen, codieren Sie die Client-Anwendung so, dass REQUEST_HA_SESSION zur Laufzeit festgelegt wird.</p>	false

Eigenschaft	Beschreibung	Standardwert
SELECT_OPENS_CURSOR	<p>Wenn diese Eigenschaft auf "true" gesetzt ist, werden Aufrufe von <code>Statement.executeQuery()</code> automatisch einen Cursor erzeugen, wenn die Abfrage eine "FOR UPDATE"-Klausel enthält.</p> <p>Wenn Sie vorher einen Aufruf von <code>Statement.setFetchSize()</code> oder <code>Statement.setCursorName()</code> für dieselbe Anweisung durchgeführt haben, hat die Einstellung "true" für <code>SELECT_OPENS_CURSOR</code> keine Wirkung.</p> <hr/> <p>Hinweis Es kann zu Verschlechterungen der Performance kommen, wenn <code>SELECT_OPENS_CURSOR</code> auf "true" gesetzt ist .</p> <hr/> <p>Weitere Hinweise zur Verwendung eines Cursors in jConnect finden Sie unter "Cursor mit Ergebnismengen verwenden" auf Seite 48.</p>	false
SERIALIZE_REQUESTS	Wenn diese Eigenschaft auf "true" gesetzt ist, wartet jConnect auf Antworten vom Server, bevor er zusätzliche Anforderungen sendet.	false
SERVICENAME	Der Name eines Backend-Datenbankservers, den ein DirectConnect-Gateway versorgt. Wird auch benutzt, um die Datenbank anzuzeigen, mit der sich Adaptive Server Anywhere verbinden will.	Keiner
SESSION_ID	Wenn diese Eigenschaft gesetzt wird, nimmt jConnect an, dass eine Anwendung versucht, die Kommunikation über eine bestehende TDS-Sitzung wieder aufzunehmen, die vom TDS-Tunnelling-Gateway aufrechterhalten wird. jConnect überspringt die Login-Dialogprozedur und leitet alle Anforderungen von der Anwendung an die angegebene Sitzungs-ID weiter.	Null

Eigenschaft	Beschreibung	Standardwert
SESSION_TIMEOUT	Stellen Sie mit dieser Eigenschaft die Zeitdauer (in Sekunden) ein, die eine http-tunnelled Sitzung (erstellt mit Hilfe des TDS-tunnelling-Servlet von jConnect) geöffnet bleibt, während sie inaktiv ist. Nach der angegebenen Zeit wird die Verbindung automatisch geschlossen. Weitere Informationen über das TDS-tunnelling-Servlet finden Sie auf Seite 147 .	Null
SQLINITSTRING	Verwenden Sie diese Eigenschaft, um eine Gruppe von Befehlen zu definieren, die an den Backend-Datenbankserver weitergegeben werden. Es muss sich dabei um SQL-Befehle handeln, die mit der Methode <code>Statement.executeUpdate()</code> ausgeführt werden können.	Null
SYB SOCKET_FACTORY	Benutzen Sie diese Eigenschaft, um jConnect in die Lage zu versetzen, die benutzerdefinierte Socket-Implementierung zu verwenden. Setzen Sie SYB SOCKET_FACTORY auf folgende Optionen: <ul style="list-style-type: none"> • Den Namen einer Klasse, die <code>com.sybase.jdbcx.SybSocketFactory</code> implementiert. • "DEFAULT", womit eine Instanz eines neuen <code>java.net.Socket()</code> erzeugt wird. Siehe " Benutzerdefinierte Socket-Plug-Ins implementieren " auf Seite 28.	Null
STREAM_CACHE_SIZE	Maximale Größe für den Cache der Antwort-Streams für eine Anweisung.	Null (unbegrenzte Cachegröße)

Eigenschaft	Beschreibung	Standardwert
USE_METADATA	<p>Wenn diese Eigenschaft auf "true" gesetzt ist, wird bei der Einrichtung einer Verbindung ein DatabaseMetaData-Objekt erstellt und initialisiert. Das DatabaseMetaData-Objekt ist für die Verbindung zu einer angegebenen Datenbank erforderlich.</p> <p>jConnect verwendet DatabaseMetaData für einige Funktionen einschließlich Verwaltungsunterstützung für verteilte Transaktionen (JTA/JTS) und Dynamic Class Loading (DCL).</p> <p>Falls der Fehler 010SJ ausgewiesen wird, der anzeigt, dass Ihre Anwendung Metadaten erfordert, installieren Sie die gespeicherten Prozeduren für die Rückgabe von Metadaten, die in jConnect enthalten sind (siehe "Gespeicherte Prozeduren installieren" in Kapitel 3 der Dokumentation <i>jConnect for JDBC Installationshandbuch</i>).</p>	true
USER	<p>Login-ID.</p> <p>Diese Eigenschaft wird automatisch gesetzt, wenn die Methode getConnection(String, String, String) verwendet wird oder explizit, wenn getConnection(String, Props) verwendet wird.</p>	Keiner
VERSIONSTRING	Read-Only-Versionsinformationen für den JDBC-Treiber.	jConnect-Treiberversion

Im folgenden Programmcode wird anhand eines Beispiels gezeigt, wie die Verbindungseigenschaften gesetzt werden. Die mit jConnect mitgelieferten Codemuster enthalten ebenfalls Beispiele für das Setzen dieser Eigenschaften.

```

Properties props = new Properties();
props.put("user", "Benutzerkennung");
props.put("password", "Benutzerkennwort");
/*
 * Wenn das Programm ein Applet ist, das auf einen
 * Server zugreifen will, der nicht auf demselben
 * Host liegt wie der Web-Server, verwendet es ein *
Proxy-Gateway.
 */
props.put("proxy", "localhost:Port");
/*
 * Die Verbindungseigenschaften müssen vor dem
 * Aufbau einer Verbindung gesetzt werden. Sie

```

```
* können die Eigenschaften auch im URL setzen.
*/
Connection con = DriverManager.getConnection
("jdbc:sybase:Tds:Host:Port", props);
```

Verbindung mit Adaptive Server Enterprise

In Ihrer Java-Anwendung definieren Sie einen URL mit Hilfe des jConnect-Treibers, um eine Verbindung mit einer Adaptive Server-Installation herzustellen. Das Basisformat des URL ist wie folgt:

```
jdbc:sybase:Tds:Host:Port
```

Dabei gilt:

jdbc:sybase – Identifiziert den Treiber

Tds – Das Sybase-Kommunikationsprotokoll für Adaptive Server

Host:Port – der Hostname von Adaptive Server und der überwachte Anschluss. Siehe in *SYBASE/interfaces* (UNIX) oder *%SYBASE%\ini\sql.ini* (Windows) den Eintrag, der von Ihrer Datenbank oder der Open-Server-Anwendung benutzt wird. Sie können den Wert für *Host:Port* dem Eintrag "query" entnehmen.

Sie können die Verbindung zu einer bestimmten Datenbank unter Verwendung des folgenden Formats herstellen:

```
jdbc:sybase:Tds:Host:Port/Datenbank
```

Hinweis Wenn Sie eine Verbindung mit einer speziellen Datenbank mit Hilfe von Adaptive Server Anywhere 6.x oder DirectConnect herstellen wollen, benutzen Sie die Verbindungseigenschaft "Connection Properties" zur Angabe des Datenbanknamens anstelle von "/Datenbank".

Beispiel

Der nachstehend gezeigte Programmcode stellt eine Verbindung zu einer Adaptive Server-Installation auf dem Host "Meinserver" und mit dem Listening-Port 3697 her:

```
SysProps.put("user", "Benutzerkennung");
SysProps.put("password", "Benutzer_Kennwort");
String url = "jdbc:sybase:Tds:Meinserver:3697";
Connection con =
    DriverManager.getConnection(url, SysProps);
```

Parameter für die Verbindungseigenschaft im URL

Sie können die Werte für die Verbindungseigenschaften des jConnect-Treibers angeben, wenn Sie einen URL definieren.

Hinweis Treiber-Verbindungseigenschaften, die im URL festgelegt wurden, überschreiben keine entsprechenden Verbindungseigenschaften, die in der Anwendung mit Hilfe der Methode **DriverManager.getConnection()** festgelegt wurden.

Um eine Verbindungseigenschaft in einem URL zu setzen, hängen Sie den Eigenschaftsnamen und seinen Wert an die URL-Definition an. Verwenden Sie dabei folgende Syntax:

```
jdbc:sybase:Tds:Host:Port/Datenbank?  
Eigenschaftsname=Wert
```

Wenn Sie mehrere Verbindungseigenschaften setzen wollen, fügen Sie danach die einzelnen Verbindungseigenschaften und Werte an, setzen aber vorher das Zeichen "&" als Verbinder. Beispiel:

```
jdbc:sybase:Tds:MeinServer:1234/MeineDatenbank?  
LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=MeinHost
```

Wenn der Wert einer dieser Verbindungseigenschaften den Verbinder "&" als Bestandteil bereits enthält, setzen Sie vor dieses "&"-Zeichen im Wert der Verbindungseigenschaft den Rückstrich (\). Beispiel: Wenn Ihr Hostname "a&bhost" lautet, verwenden Sie folgende Syntax:

```
jdbc:sybase:Tds:MeinServer:1234/MeineDatenbank?  
LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=  
a\&bhost
```

Benutzen Sie keine Anführungszeichen in den Werten der Verbindungseigenschaften, auch wenn es sich um Zeichenfolgen handelt. Richtig:

```
HOSTNAME=MeinHost
```

Falsch:

```
HOSTNAME="MeinHost"
```


Verbindung mit Adaptive Server Anywhere

Wenn Sie jConnect mit Adaptive Server Anywhere verwenden wollen, müssen Sie ein Upgrade auf Adaptive Server Anywhere Version 6.x vornehmen.

Mit Adaptive Server Anywhere 5.x.x verbinden

Wenn Sie sich mit Adaptive Server Anywhere Version 5.x.x über jConnect verbinden wollen, müssen Sie Adaptive Server Anywhere Open Server Gateway, das mit Adaptive Server Anywhere mitgeliefert wird, über dbos50 starten.

Hinweis Die kostenlose Download-Version von Adaptive Server Anywhere, die auf der Website von Powersoft zur Verfügung steht, enthält dieses Open Server Gateway nicht. Rufen Sie bei der Sybase-Vertretung in Ihrem Land (in den USA unter (800) 265-4555) an: Sie erhalten eine CD zugeschickt, die das Open Server Gateway und die erforderlichen Open-Server-DLLs enthält. Versand- und Lieferspesen werden berechnet.

- 1 Installieren Sie Open Server Gateway 5.5.x3 oder höher und die Open Server DLLs. Verwenden Sie die Open Server DLLs der Version 11.1.
- 2 Fügen Sie einen Eintrag für das Gateway in die Datei *%SYBASE%\ini\sql.ini* ein (z.B. mit **sqledit**).
- 3 Starten Sie das Gateway mit folgender Eingabe:

```
Start dbos50 gateway-demo
```

Dabei gilt: *gateway-demo* ist der Gateway-Name, der in Schritt 2 definiert wurde.

- 4 Wenn das Open Server Gateway läuft, können Sie eine Verbindung wie folgt definieren:

```
jdbc:sybase:Tds:Host:Port
```

Host ist der Hostname, auf dem Adaptive Server Anywhere und Open Server Gateway laufen, und *Port* ist die in *sql.ini* festgelegte Anschlussnummer.

Hinweis Um mehrere Adaptive Server Anywhere-Datenbanken zu unterstützen, verwenden Sie **sqledit**, um einen Eintrag mit einem unterschiedlichen Anschluss für jede Datenbank hinzuzufügen, dann starten Sie das Open Server Gateway für jede Datenbank.

Verbindung mit einem Server über JNDI aufnehmen

In jConnect 4.0 und höher können Sie die Java-Schnittstelle für Namens- und Verzeichnisdienste (Java Naming and Directory Interface, JNDI) benutzen, um Verbindungsinformationen bereitzustellen, was mit folgenden Vorteilen verbunden ist:

- Hostnamen und Ports für die Verbindung mit einem Server können an einem zentralen Ort bereitgehalten werden. Das Hartkodieren eines bestimmten Hostnamens oder einer Portnummer in einer Anwendung erübrigt sich daher.
- An einem zentralen Ort können Sie Verbindungseigenschaften und eine Standarddatenbank festlegen und für alle Anwendungen bereithalten.
- Sie können die jConnect-Eigenschaft **CONNECTION_FAILOVER** von jConnect 4.0 verwenden, um das Verhalten nach fehlgeschlagenen Verbindungsversuchen festzulegen. Wenn **CONNECTION_FAILOVER** auf "true" gesetzt ist, versucht jConnect die Verbindung mit einer Folge von Host/Port-Serveradressen im JNDI-Namensbereich, bis eine Verbindung erfolgreich verläuft.

Wenn Sie jConnect mit JNDI verwenden wollen, müssen Sie dafür sorgen, dass bestimmte Informationen in allen Verzeichnisdiensten vorhanden sind, auf die JNDI zugreift, und dass die nötigen Informationen in der Klasse **javax.naming.Context** gesetzt sind. Dieser Abschnitt behandelt folgende Themen:

- [URL für die Verbindung bei Verwendung von JNDI](#)
- [Erforderliche Verzeichnisdienst-Informationen](#)
- [CONNECTION_FAILOVER, Verbindungseigenschaft](#)
- [Bereitstellung von JNDI-Kontextinformationen](#)

URL für die Verbindung bei Verwendung von JNDI

Um festzulegen, dass jConnect JNDI für die Übernahme der Verbindungsinformationen verwenden soll, setzen Sie "jndi" als Subprotokoll des URLs nach "sybase":

```
jdbc:sybase:jndi:Protokollinformationen-für-JNDI
```

Alles, was im URL nach "jndi" kommt, wird über JNDI verwaltet. Beispiel: Wenn Sie JNDI mit dem Lightweight Directory Access Protocol (LDAP) verwenden wollen, können Sie folgende Eingabe vornehmen:

```
jdbc:sybase:jndi:ldap://LDAP_Hostname:Portnummer/Servername=
Sybase11,o=MeineFirma,c=US
```

Dieser URL weist JNDI an, Informationen von einem LDAP-Server zu beziehen, übergibt den Hostnamen und die Portnummer des LDAP-Servers und liefert den Namen eines Datenbankservers in einer LDAP-spezifischen Form.

Erforderliche Verzeichnisdienst-Informationen

Wenn JNDI mit jConnect verwendet wird, muss JNDI folgende Informationen für den Ziel-Datenbankserver zurückgeben:

- Einen Hostnamen und eine Portnummer, mit denen die Verbindung eingerichtet werden soll
- Den Namen der Datenbank, die benutzt werden soll
- Alle Verbindungseigenschaften, die einzelne Anwendungen nicht selbst setzen dürfen

Diese Informationen müssen in einem festgelegten Format in jedem Verzeichnisdienst gespeichert werden, der für die Bereitstellung von Verbindungsinformationen verwendet wird. Das benötigte Format besteht aus einem numerischen Objektbezeichner (OID), der die Art der bereit gestellten Informationen angibt (z.B. die Ziel-Datenbank), gefolgt von der formatierten Information. [Tabelle 2-3](#) zeigt die erforderliche Formatierung.

Tabelle 2-3: Erforderliche Verzeichnisdienstinformationen für JNDI

Art der Information	Objektbezeichner (Object Identifier, OID)	Format	Kommentare
Host und Port	1.3.6.1.4.1.897.4.2.5	TCP#1# <i>Hostname</i> <i>Portnummer</i>	Sie können mehrfache Hosts und Ports als getrennte Eingaben festlegen, wodurch Sie die Option erhalten, CONNECTION_FAILOVER zu verwenden.
Verbindungseigenschaft	1.3.6.1.4.1.897.4.2.10	<i>Eigenschaft1=Wert&Eigenschaft2=Wert&Eigenschaft3=Wert&...</i>	Mehrere Verbindungseigenschaften können in Form von getrennten Einträgen für jede Eigenschaft oder durch Eingabe mehrerer, durch das Zeichen "&" getrennter Eigenschaften in einem Eintrag definiert werden.
Datenbank	1.3.6.1.4.1.897.4.2.11	<i>Datenbankname</i>	Der Name der Datenbank, mit der Sie sich verbinden wollen. Diese Eigenschaft funktioniert wie <code>"/Datenbank"</code> in der JDBC URL.
Verbindungsprotokoll	1.3.6.1.4.1.897.4.2.9	Tds	Fakultativ, aber wenn Sie ein Verbindungsprotokoll verwenden, muss der Parameter immer "Tds" sein.

Das folgende Beispiel zeigt Verbindungsinformationen, die für den Datenbankserver SYBASE11 unter einem LDAP-Verzeichnisdienst eingegeben werden:

```
dn: Servername=SYBASE11,o=MeineFirma,c=US
  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
```

Im vorstehenden Beispiel kann der Zugriff auf SYBASE11 über Import 1266 oder Port 1337 auf dem Host "giotto" und über Port 4444 auf dem Host "standby1" erfolgen. Zwei Verbindungseigenschaften, REPEAT_READ und PACKETSIZE, werden in einem Eintrag gesetzt. Die Verbindungseigenschaft CONNECTION_FAILOVER wird als getrennter Eintrag gesetzt. Anwendungen, die sich mit SYBASE11 verbinden, werden sofort mit der Datenbank *pubs2* verbunden. Sie müssen kein Verbindungsprotokoll festlegen. Wenn Sie dies aber tun, müssen Sie das Attribut als "Tds" eingeben und nicht als "TDS".

CONNECTION_FAILOVER, Verbindungseigenschaft

CONNECTION_FAILOVER ist eine mit Booleschen Werten operierende Verbindungseigenschaft, die benutzt wird, wenn jConnect JNDI für die Übernahme von Verbindungsinformationen verwendet.

Wenn CONNECTION_FAILOVER auf "true" gesetzt ist, kann jConnect mehrfache Versuche unternehmen, mit einem Server Verbindung aufzunehmen. Wenn eine versuchte Verbindung mit einem Hostnamen und einer Portnummer fehlschlägt, die mit einem Server verknüpft sind, verwendet jConnect JNDI, um den nächsten Hostnamen und die nächste Portnummer zu beziehen, die diesem Server zugeordnet wurden, um eine Verbindung einzurichten. Die Verbindungsversuche werden sequenziell über alle Hostnamen und Portnummern fortgeführt, die mit einem Server verknüpft sind.

Beispiel: CONNECTION_FAILOVER ist auf "true" gesetzt. Ein Datenbankserver ist mit den folgenden Hostnamen und Portnummern verknüpft (wie im weiter oben gezeigten LDAP-Beispiel):

```
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
```

Um eine Verbindung mit dem Server aufzunehmen, versucht jConnect, mit dem Host "giotto" an der Portnummer 1266 zu verbinden. Wenn dies fehlschlägt, versucht jConnect die Portnummer 1337 auf "giotto". Wenn dies fehlschlägt, versucht jConnect eine Verbindung mit dem Host "standby1" über Portnummer 4444.

Der Standardwert für CONNECTION_FAILOVER ist "true".

Wenn `CONNECTION_FAILOVER` auf "false" gesetzt ist, versucht jConnect die Verbindung mit einem ersten Hostnamen und einer ersten Portnummer. Wenn der Versuch fehlschlägt, gibt jConnect einen Ausnahmefehler aus und führt keinen weiteren Versuch durch.

Bereitstellung von JNDI-Kontextinformationen

Wenn ein Entwickler jConnect mit JNDI einsetzen will, muss er die JNDI-Spezifikation von Sun Microsystems kennen, die auf dem Web unter folgender Adresse zu beziehen ist:

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

Der Entwickler muss vor allem dafür sorgen, dass die erforderlichen Initialisierungseigenschaften in **`javax.naming.directory.DirContext`** gesetzt sind, wenn JNDI und jConnect gemeinsam eingesetzt werden. Diese Eigenschaften können auf Systemebene oder zur Laufzeit gesetzt werden.

Zwei wichtige Eigenschaften sind:

- `Context.INITIAL_CONTEXT_FACTORY`

Diese Eigenschaft übernimmt den voll qualifizierten Klassennamen der `InitialContextFactory` für die Verwendung durch JNDI. Damit wird der JNDI-Treiber festgelegt, der mit dem in der Eigenschaft `Context.PROVIDER_URL` festgelegten URL verwendet wird.

- `Context.PROVIDER_URL`

Diese Eigenschaft übernimmt den URL des Verzeichnisdienstes, auf den der Treiber (z.B. der LDAP-Treiber) zugreifen soll. Der URL muss eine Zeichenfolge sein, z.B. "ldap://ldaphost:427".

Das folgende Beispiel zeigt, wie die Context-Eigenschaften zur Laufzeit gesetzt werden, und wie eine Verbindung mit JNDI und LDAP aufgebaut werden kann. In diesem Beispiel wird die Eigenschaft `INITIAL_CONTEXT_FACTORY` so gesetzt, dass die Implementierung eines LDAP-Service-Providers von Sun Microsystems aufgerufen wird. Die Context-Eigenschaft `PROVIDER_URL` wird auf den URL eines LDAP-Verzeichnisdienstes gesetzt, der sich auf dem Host "ldap_server1" am Port 983 befindet.

```
Properties props = new Properties();
```

```
/* Wir wollen LDAP benutzen, daher wird INITIAL_CONTEXT_FACTORY  
* auf den Klassennamen einer LDAP-ContextFactory gesetzt. In  
* diesem Fall wird die ContextFactory durch die
```

```

* Sun-Implementierung eines Treibers für den LDAP-Verzeichnisdienst
* bereitgestellt. */
props.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

/* Jetzt setzen wir PROVIDER_URL auf den URL des LDAP-Servers,
* der Verzeichnisinformationen für die Verbindung bereitstellt. */
props.put(Context.PROVIDER_URL, "ldap://ldap_server1:983");

/* Gegebenfalls zusätzliche Context-Eigenschaften setzen. */
props.put("user", "xyz");
props.put("password", "123");

/* Verbindung aufnehmen */
Connection con = DriverManager.getConnection
    ("jdbc:sybase:jndi:ldap://ldap_server1:983" +
    "/Servername=Sybasell,o=MeineFirma,c=US",props);

```

Beachten Sie, dass die an **getConnection()** übergebene Verbindungs-Zeichenfolge LDAP-spezifische Informationen enthält, die der Entwickler vorsehen muss.

Wenn JNDI-Eigenschaften zur Laufzeit gesetzt werden, wie dies im vorstehenden Beispiel der Fall ist, übergibt sie jConnect an JNDI für die Initialisierung eines Servers, wie im folgenden jConnect-Code:

```

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

```

jConnect bezieht dann die benötigten Verbindungsinformationen aus JNDI durch Aufruf von **DirContext.getAttributes()**, wie im folgenden Beispiel gezeigt wird, in dem *ctx* ein **DirContext**-Objekt ist:

```

javax.naming.directory.Attributes attrs =
    ctx.getAttributes("ldap://ldap_server1:983/Servername=
    Sybasell, SYBASE_SERVER_ATTRIBUTES");

```

In diesem Beispiel ist SYBASE_SERVER_ATTRIBUTES eine Tabelle von Zeichenfolgen, die in jConnect definiert sind. Die Tabellenwerte sind die OID für die erforderlichen Verzeichnisinformationen, die in [Tabelle 2-3](#) aufgelistet sind.

Benutzerdefinierte Socket-Plug-Ins implementieren

In diesem Abschnitt wird erklärt, wie eine benutzerdefinierte Socketimplementierung in eine Anwendung eingebaut werden kann, um die Kommunikation zwischen einem Server und einem Client anzupassen.

javax.net.ssl.SSLSocket ist ein Beispiel für einen Socket, den Sie gegebenenfalls anpassen müssen, um die Verschlüsselung zu aktivieren.

com.sybase.jdbcx.SybSocketFactory ist eine Erweiterungs-Schnittstelle von Sybase. Sie enthält die Methode **createSocket(String, int, Properties)**, die einen **java.net.Socket** zurückgibt. Damit ein jConnect-Treiber Version 4.1 oder höher einen benutzerdefinierten Socket laden kann, muss die Anwendung folgende Voraussetzungen erfüllen:

- Implementierung dieser Schnittstelle
- Definition der Methode **createSocket(..)**

jConnect benutzt den neuen Socket für die nachfolgenden Eingabe/Ausgabe-Operationen. Klassen, die **SybSocketFactory** implementieren, erstellen Sockets und bieten die Rahmenbedingungen für die Hinzufügung der als public definierten Funktionen auf Socketebene.

```
/**
 * Gibt einen mit einem ServerSocket verbundenen Socket auf dem
 * benannten Host und einem gegebenen Port zurück.
 * @param host ist der Serverhost
 * @param port ist der Serverport
 * @param props sind Eigenschaften auf der Verbindung
 * @returns Socket
 * @exception IOException, UnknownHostException
 */
public java.net.Socket createSocket(String host, int port, Properties props)
throws IOException, UnknownHostException;
```

Die Übergabe von Eigenschaften ermöglicht es den Instanzen von **SybSocketFactory**, Verbindungseigenschaften zu verwenden, um einen intelligenten Socket zu implementieren.

Wenn Sie **SybSocketFactory** implementieren, um einen Socket zu produzieren, kann derselbe Anwendungscode verschiedene Socketarten verwenden, indem an die Anwendung die unterschiedlichen Arten von Factories oder Pseudo-Factories übergeben werden, die Sockets erstellen. Sie können Factories mit Parametern anpassen, die beim Socketaufbau verwendet werden. Sie können z.B. Factories so konfigurieren, dass sie Sockets mit unterschiedlichen Netzwerk-Zeitüberschreitungen oder bereits konfigurierten Sicherheitsparametern zurückgeben. Die an die Anwendung zurückgegebenen Anwendungen können Unterklassen von **java.net.Socket** sein, um neue APIs direkt an neuen Funktionen wie Kompression, Sicherheit, Datensatzmarkierung, Statistiksammlung oder Firewall-Durchgang anzubinden (**javax.net.SocketFactory**).

Hinweis SybSocketFactory ist eine vereinfachte **javax.net.SocketFactory**, mit der Anwendungen eine Überbrückung von **java.net.*** zu **javax.net.*** schaffen können, falls nötig.

So verwenden Sie einen benutzerdefinierten Socket mit jConnect:

- 1 Stellen Sie eine Java-Klasse bereit, die **com.sybase.jdbcx.SybSocketFactory** implementiert. Siehe ["Einen angepassten Socket erstellen und konfigurieren"](#) auf Seite 30.
- 2 Setzen Sie die Verbindungseigenschaft SYB_SOCKET_FACTORY so, dass jConnect Ihre Implementierung verwenden kann, um einen Socket zu erhalten.

SYB_SOCKET_FACTORY, Verbindungseigenschaft

Wenn Sie einen benutzerdefinierten Socket mit jConnect verwenden wollen, setzen Sie die Verbindungseigenschaft SYB_SOCKET_FACTORY auf eine Zeichenfolge mit einer den folgenden Eigenschaften:

- Name einer Klasse, die **com.sybase.jdbcx.SybSocketFactory** implementiert
- oder
- DEFAULT, was einen neuen **java.net.Socket()** instanziiert.

Unter "[Verbindungseigenschaften festlegen](#)" auf Seite 12 finden Sie Hinweise, wie SYB SOCKET_FACTORY gesetzt wird.

Einen angepassten Socket erstellen und konfigurieren

Sobald jConnect einen benutzerdefinierten Socket erhalten hat, benutzt die Software den Socket, um sich mit einem Server zu verbinden. Etwaige Konfigurationen des Sockets müssen abgeschlossen sein, bevor jConnect ihn bezieht.

In diesem Abschnitt wird beschrieben, wie Sie mit jConnect das Plug-In einer SSL Socket-Implementierung vornehmen, wie z.B. **javax.net.ssl.SSLSocket**.

Hinweis Derzeit wird SSL von keinem Sybase-Server unterstützt.

Im folgenden Beispiel wird gezeigt, wie eine Implementierung von SSL eine Instanz eines **SSLSocket** erstellen, konfigurieren und dann zurückgeben kann. In diesem Beispiel implementiert **MySSLSocketFactory** die Klasse **SybSocketFactory** und erweitert **javax.net.ssl.SSLSocketFactory**, um SSL zu implementieren. Es enthält zwei **createSocket**-Methoden, eine für **SSLSocketFactory** und eine für **SybSocketFactory**. Diese Methoden übernehmen folgende Aufgaben:

- Sie erstellen einen SSL-Socket.
- Sie rufen **SSLSocket.setEnabledCipherSuites()** auf, um die Cipher-Suites anzugeben, die für die Verschlüsselung zur Verfügung stehen.
- Sie geben den Socket zurück, der von jConnect verwendet werden soll.

Beispiel

```
public class MySSLSocketFactory extends SSLSocketFactory
    implements SybSocketFactory
{
    /**
     * Socket erstellen, nutzbare Cipher-Suites setzen, Socket
     * zurückgeben.
     * Zeigt, wie Cipher-Suites in der Implementierung
     * hartkodiert werden können.
     *
     * Siehe javax.net.SSLSocketFactory#createSocket
     */
```

```

public Socket createSocket(String host, int port)
    throws IOException, UnknownHostException
{
    // Array vorbereiten, das die zu aktivierenden Cipher-
    // Suites enthält.
    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(host, port);
    ((SSLSocket)s).setEnabledCipherSuites(enableThese);
    return s;
}
/**
 * Rückgabe eines SSLSockets.
 * Veranschaulicht, wie Cipher-Suites basierend auf
 * Verbindungseigenschaften der folgenden Art gesetzt werden:
 * Properties _props = new Properties();
 * Andere url, password, etc. Eigenschaften setzen.
 * _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);
 *
 * Siehe com.sybase.jdbcx.SybSocketFactory#createSocket
 */
public Socket createSocket(String host, int port,
    Properties props)
    throws IOException, UnknownHostException
{
    // Nachsehen, ob Cipher-Suites in den
    // Verbindungseigenschaften gesetzt sind.
    Vector cipherSuites = new Vector();
    String cipherSuiteVal = null;
    int cipherIndex = 1;
    do
    {
        if((cipherSuiteVal = props.getProperty("CIPHER_SUITES_"
            + cipherIndex++)) == null)

```

Benutzerdefinierte Socket-Plug-Ins implementieren

```
{
    if(cipherIndex <= 2)
    {
        // Keine Cipher-Suites verfügbar.
        // Rückgabe, was das Objekt als für seinen
        // Standard-SSLSocket hält, mit aktivierten
// Cipher-Suites.
        return createSocket(host, port);
    }
    else
    {
        // Wir haben mindestens eine Cipher-Suite pro
        // Anforderung auf der Verbindung zu aktivieren.
        break;
    }
    else
    {
        // Zur Cipher-Suite Vector hinzufügen, damit wir
        // sie gemeinsam aktivieren können.
        cipherSuites.addElement(cipherSuiteVal);
    }
}
while(true);
// Ermöglicht String[] aus dem erstellten Vektor.
String enableThese[] = new String[cipherSuites.size()];
cipherSuites.copyInto(enableThese);
// Cipher-Suites aktivieren
Socket s =
    SSLSocketFactory.getDefault().createSocket
        (host, port);
((SSLSocket)s).setEnabledCipherSuites(enableThese);
// SSLSocket zurückgeben
return s;
}
// Andere Methoden
}
```

Da jConnect keine Informationen über die Art seines Sockets benötigt, müssen Sie Konfigurationen abschließen, bevor Sie einen Socket zurückgeben.

Weitere Hinweise finden Sie unter:

- *Encrypt.java* – in den Unterverzeichnissen *sample* (jConnect 4.x) und *sample2* (jConnect 5.x) in Ihrem jConnect-Verzeichnis. Dieses Beispiel zeigt Ihnen, wie die Schnittstelle **SybSocketFactory** mit jConnect-Anwendungen verwendet wird.

- *MySSLSocketFactory.java* – befindet sich ebenfalls in den Unterverzeichnissen *sample* (jConnect 4.x) und *sample2* (jConnect 5.x) in Ihrem jConnect-Verzeichnis. Es ist eine Beispiel-Implementierung der **SybSocketFactory**-Schnittstelle, die Sie in Ihre Anwendung einbauen und verwenden können.

Internationalisierungs- und Sprachanpassungsfragen

In diesem Abschnitt werden Fragen der Internationalisierung und Lokalisierung behandelt, die für jConnect von Bedeutung sind.

Zeichensatzkonverter von jConnect

jConnect benutzt spezielle Klassen für alle Zeichensatz-Konvertierungen. Sie legen fest, wie jConnect Einbyte- und Mehrbyte-Zeichensatz-Konvertierungen durchführt, und welche Auswirkungen die Zeichensatzkonvertierungen auf die Performance Ihrer Anwendungen haben, indem Sie die Zeichensatz-Konvertierungsklasse wählen, die jConnect benutzen soll.

Es gibt zwei Zeichensatz-Konvertierungsklassen. Die Konvertierungsklasse, die jConnect verwendet, basiert auf der Versionseinstellung (z.B. `VERSION_4`) und den Verbindungseigenschaften `CHARSET` und `CHARSET_CONVERTER_CLASS`.

- Die Klasse **TruncationConverter** funktioniert nur mit Einbyte-Zeichensätzen, die ASCII-Zeichen verwenden, z.B. `iso_1` und `cp850`. Sie funktioniert nicht mit Mehrbyte-Zeichensätzen oder Einbyte-Zeichensätzen, die Zeichen außerhalb des ASCII-Standards verwenden.

Wenn die Klasse **TruncationConverter** eingesetzt wird, verarbeitet jConnect 5.x Zeichensätze auf dieselbe Weise wie jConnect Version 2.2. Die Klasse **TruncationConverter** ist der Standardkonverter, wenn die Versionseinstellung `VERSION_2` ist.

- Die Klasse **PureConverter** ist ein reiner Java-Zeichensatzkonverter für Mehrbyte-Zeichensätze. jConnect benutzt diese Konverterklasse, wenn die Versionseinstellung `VERSION_4` oder höher lautet. jConnect benutzt diesen Konverter auch mit `VERSION_2`, wenn ein in der Verbindungseigenschaft `CHARSET` angegebener Zeichensatz festgestellt wird, bei dem die Klasse **TruncationConverter** nicht verwendet werden kann.

Obwohl die Klasse **PureConverter** Mehrbyte-Zeichensatzkonvertierung ermöglicht, kann sie die Performance des jConnect-Treibers negativ beeinflussen. Wenn die Performance des Treibers ein wesentliches Kriterium ist, siehe ["Die Performance der Zeichensatzkonvertierung verbessern"](#) auf Seite 36.

Zeichensatz-Konverter auswählen

jConnect benutzt die Versionseinstellung in **SybDriver.setVersion()**, um zu ermitteln, welche Standard-Zeichensatzkonverterklasse verwendet werden soll. Für **VERSION_2** ist der Standardwert **TruncationConverter**. Für **VERSION_4** und höher ist der Standardwert **PureConverter**.

Sie können auch die Verbindungseigenschaft **CHARSET_CONVERTER_CLASS** setzen, um festzulegen, welchen Zeichensatzkonverter jConnect verwenden soll. Dies ist sinnvoll, wenn Sie einen anderen als den Standard-Zeichensatz-Konverter für Ihre jConnect-Version verwenden wollen.

Beispiel: Wenn Sie jConnect auf **VERSION_4** oder höher setzen, aber lieber die Klasse **TruncationConverter** verwenden als die für Mehrbyte-Zeichensätze geeignete **PureConverter**-Klasse, können Sie **CHARSET_CONVERTER_CLASS** wie folgt einrichten:

Für jConnect 4.1:

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

Für jConnect 5.x:

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

Verbindungseigenschaft CHARSET festlegen

Sie können festlegen, welcher Zeichensatz in Ihrer Anwendung zu verwenden ist, indem Sie die Treiber-Eigenschaft **CHARSET** setzen. Wenn Sie die Eigenschaft **CHARSET** nicht setzen, gilt folgendes:

- Bei **VERSION_2** benutzt jConnect **iso_1** als Standard-Zeichensatz.
- Für **VERSION_3**, **VERSION_4** und **VERSION_5** verwendet jConnect den Standard-Zeichensatz der Datenbank und passt sich automatisch an, so dass erforderliche Zeichensatzkonvertierungen auf der Clientseite durchgeführt werden.

Sie können auch den Programm-Startparameter **-J charset** für die **IsqlApp**-Anwendung verwenden, um einen Zeichensatz festzulegen.

Um zu ermitteln, welche Zeichensätze auf Ihrer Adaptive Server-Installation vorhanden sind, führen Sie die folgende SQL-Abfrage auf Ihrem Server aus:

```
select name from syscharsets
```

Bei der Klasse **PureConverter** gilt: Wenn der bezeichnete CHARSET mit der Java Virtual Machine (VM) des Clients nicht zusammenarbeitet, schlägt die Verbindung fehl, und eine **SQLException**-Ausnahmebedingung wird angezeigt, die darauf hinweist, dass Sie den CHARSET-Wert auf einen Zeichensatz festlegen müssen, der von Adaptive Server und vom Client unterstützt wird.

Bei der Klasse **TruncationConverter** wird die Zeichenkürzung immer angewendet, gleichgültig ob der bezeichnete CHARSET 7-Bit ASCII ist oder nicht.

Die Performance der Zeichensatzkonvertierung verbessern

Wenn Sie Mehrbyte-Zeichensätze verwenden und die Treiber-Performance verbessern müssen, können Sie die Klasse **SunioConverter** verwenden, die mit den jConnect-Beispielen mitgeliefert wird. Weitere Hinweise finden Sie unter ["Zeichensatzkonvertierung" auf Seite 118](#).

Unterstützte Zeichensätze

[Tabelle 2-4](#) enthält die Sybase-Zeichensätze, die in dieser Version von jConnect unterstützt werden. Die Tabelle enthält darüber hinaus auch den JDK-Byte-Konverter für jeden unterstützten Zeichensatz.

Obwohl jConnect UCS-2 unterstützt, wird UCS-2 derzeit von keiner Sybase-Datenbank und keinem Open Server unterstützt.

Der Sybase-Zeichensatz *sjis* enthält keine IBM- oder Microsoft-Erweiterungen von JIS, während der JDK SJIS Byte-Konverter diese Erweiterungen enthält. Konvertierungen von Java-Zeichenfolgen in einer Sybase-Datenbank, in der *sjis* verwendet wird, kann daher Zeichenwerte ergeben, die von der Sybase-Datenbank nicht unterstützt werden. Bei Konvertierungen von *sjis* auf Java-Zeichenfolgen in einer Sybase-Datenbank dürfte dieses Problem allerdings nicht auftreten.

[Tabelle 2-4](#) zeigt die Zeichensätze, die derzeit von Sybase unterstützt werden.

Tabelle 2-4: Unterstützte Sybase-Zeichensätze

SybCharset-Name	JDK-Byte-Konverter
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
iso_1	8859_1
iso88592	8859-2
is088595	8859_5

SybCharset-Name	JDK-Byte-Konverter
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 (see note)	SJIS
tis620	MS874
utf8	UTF8

Unterstützung für das Euro-Währungssymbol

jConnect Version 4.1 und höher unterstützen die Verwendung des neuen Euro-Währungssymbols "*euro*" und seine Umsetzung zum und vom UCS-2 Unicode.

Der *euro* wurde folgenden Sybase-Zeichensätzen hinzugefügt: cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258, cp874, iso885915 und utf8.

Die Zeichensätze cp1257, cp1258 und iso885915 sind neu.

So verwenden Sie das *euro*-Symbol:

- Benutzen Sie die **PureConverter**-Klasse, einen reinen Java-Konverter für Mehrbyte-Zeichensätze. Weitere Hinweise finden Sie unter ["Zeichensatzkonverter von jConnect" auf Seite 34](#).
- Prüfen Sie, ob die neuen Zeichensätze auf dem Server installiert sind.
Das *euro*-Symbol wird derzeit nur auf Adaptive Server Enterprise Version 11.9.2 und darüber unterstützt. Adaptive Server Anywhere unterstützt das *euro*-Symbol nicht.
- Wählen Sie den geeigneten Zeichensatz auf dem Client. Weitere Hinweise finden Sie unter ["Verbindungseigenschaft CHARSET festlegen" auf Seite 35](#).

- Führen Sie ein Upgrade auf JDK 1.1.7 oder die Java™ 2 Plattform durch.

Nicht unterstützte Zeichensätze

Folgende Sybase-Zeichensätze werden in jConnect 5.x nicht unterstützt, da es für die Sybase-Zeichensätze keine analogen JDK-Bytekonverter gibt:

- cp1047
- euccns
- greek8
- roman8
- turkish8

Sie können diese Zeichensätze mit der Klasse **TruncationConverter** benutzen, wenn die Anwendung nur die 7-Bit-ASCII-Untergruppen dieser Zeichen verwendet.

Mit Datenbanken arbeiten

In diesem Abschnitt werden Fragen erörtert, die das Zusammenspiel der Datenbanken mit jConnect betreffen. Folgende Themen werden behandelt:

- Unterstützung für Hochverfügbarkeits-Notumschaltung implementieren
- Remote Procedure Call von Server zu Server ausführen
- Zugriff auf Datenbank-Metadaten
- Cursor mit Ergebnismengen verwenden
- Unterstützung von Aktualisierungen in Form von Anweisungsfolgen
- Datenbank aus der Ergebnismenge einer gespeicherten Prozedur aktualisieren
- Mit Datenbanken arbeiten

Unterstützung für Hochverfügbarkeits-Notumschaltung implementieren

jConnect-Versionen 4.2 und 5.2 unterstützen die Funktion der Notumschaltung von Sybase für Hochverfügbarkeitssysteme, die in Adaptive Server Enterprise Version 12.0 verfügbar ist.

Hinweis Die Sybase-Notumschaltung in einem Hochverfügbarkeitssystem unterscheidet sich von der Funktion "Notumschaltung einer Verbindung". Sybase empfiehlt ausdrücklich, dass Sie diesen Abschnitt *sehr sorgfältig* lesen, wenn Sie beide verwenden wollen.

Überblick

Mit Hilfe der Sybase-Notumschaltung können Sie zwei Adaptive Server Version 12.0 als einander begleitende Systeme konfigurieren. Falls das primäre System ausfällt, können die Devices, Datenbanken und Verbindungen vom zweiten begleitenden System übernommen werden.

Sie können ein Hochverfügbarkeitssystem entweder asymmetrisch oder symmetrisch konfigurieren.

Eine *asymmetrische* Konfiguration beinhaltet zwei Adaptive Server, wobei sich jeder physikalisch auf einem anderen Rechner befindet. Diese Rechner sind miteinander verbunden, und wenn einer der Server herunterfährt, übernimmt der andere die Arbeit. Der sekundäre Adaptive Server funktioniert als "Hot Standby" und beginnt erst bei einer Notumschaltung zu arbeiten.

Eine *symmetrische* Konfiguration beinhaltet ebenfalls zwei Adaptive Server, die auf verschiedenen Rechnern laufen. Wenn jedoch ein System ausfällt, kann jede Adaptive Server-Installation als primäres oder sekundäres System für die andere Adaptive Server-Installation arbeiten. In dieser Konfiguration ist jede Adaptive Server-Installation mit ihren eigenen Devices, Systemdatenbanken und Benutzeranmeldungen voll funktionsfähig.

In beiden Installationen werden die beiden Rechner für Dual-Zugriff konfiguriert, was bewirkt, dass die Plattenspeicher für beide Rechner sichtbar sind und beide darauf zugreifen können.

Sie können die Sybase-Notumschaltung in jConnect aktivieren und eine Client-Anwendung mit einer Adaptive Server-Installation verbinden, der für die Notumschaltung konfiguriert ist. Wenn der primäre Server auf den sekundären Server umgeschaltet wird, schaltet die Client-Anwendung ebenfalls automatisch auf den sekundären Server um und nimmt die Netzwerkverbindung wieder auf.

Hinweis Detailliertere Informationen über die Sybase-Notumschaltung finden Sie in der Dokumentation *Sybase Failover in High Availability-Systemen*.

Erfordernisse, Abhängigkeiten und Einschränkungen

- Sie müssen zwei Adaptive Server Version 12.0 für die Notumschaltung konfigurieren.
- Sie müssen jConnect 4.2 oder jConnect 5.2 verwenden. Ältere Treiber unterstützen diese Funktion nicht.
- Wenn der Client umschaltet, werden nur die Änderungen beibehalten, die an die Datenbank vor der Notumschaltung übermittelt wurden.
- Die Verbindung der Client-Anwendung muss mit JNDI aufgebaut worden sein. Siehe ["Verbindung mit einem Server über JNDI aufnehmen"](#) auf [Seite 22](#).
- Die Ereignisbenachrichtigung in jConnect funktioniert bei einer Umschaltung nicht. Siehe ["Ereignismeldungen benutzen"](#) auf [Seite 68](#).

- Schließen Sie alle Anweisungen, wenn sie nicht mehr verwendet werden. jConnect speichert Informationen über Anweisungen, um die Notumschaltung zu ermöglichen. Wenn Sie die Anweisungen nicht schließen, treten Speicherausfälle auf.

Die Notumschaltung in jConnect implementieren

So implementieren Sie die Unterstützung für die Notumschaltung in jConnect:

- 1 Konfigurieren Sie den primären und den sekundären Adaptive Server für die Notumschaltung.
- 2 Schließen Sie einen Eintrag für den primären Server und einen weiteren Eintrag für den sekundären Server in die Informationsdatei für den Verzeichnisdienst mit ein, die JNDI benötigt. Der Eintrag für den primären Server hat ein Attribut (HA OID), das sich auf den Eintrag für den sekundären Server bezieht.

Wenn Sie LDAP als Dienstanbieter für JNDI verwenden, kann das HA-Attribut zwei verschiedene Formen haben:

- *Relative Distinguished Name (RDN)* – Diese Form setzt voraus, dass die Basis für die Suche (normalerweise durch das Attribut **java.naming.provider.url** geliefert), kombiniert mit dem Wert dieses Attributs, genügt, um den sekundären Server zu identifizieren. Beispiel: Wir nehmen an, dass sich der primäre Server an Hostname:4200 und der sekundäre Server an Hostname:4202 befindet:

```
dn: servername=happrimary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#Hostname 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary
objectclass: sybaseServer
```

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

- *Distinguished Name (DN)* – Diese Form setzt voraus, dass der Wert des HA-Attributs den sekundären Server eindeutig identifiziert. In der Basis für die Suche gefundene Werte werden gegebenenfalls verdoppelt. Beispiel:

```
dn: servername=happrimary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#Hostname 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary,
```

```
o=Sybase, c=US ou=Accounting
objectclass: sybaseServer
```

```
dn: servername=hasecondary, o=Sybase, c=US, ou=Accounting
1.3.6.1.4.1.897.4.2.5: TCP#1#Hostname 4202
objectclass: sybaseServer
```

Beachten Sie, dass sich **hasecondary** in einem anderen Zweig des Baums befindet (das sehen Sie im zusätzlichen Qualifizierer **ou=Accounting**).

- *Full LDAP URL* – Diese Form setzt für die Basis der Suche nichts voraus. Das HA-Attribut wird als voll-qualifizierter LDAP URL erwartet, der zum Identifizieren des sekundären Servers verwendet wird (es kann auch auf einen anderen LDAP-Server zeigen). Beispiel:

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

dn: servername=secondary, o=Sybase, c=US, ou=Accounting
1.3.6.1.4.1.897.4.2.5: TCP#1#Hostname 4202
objectclass: sybaseServer
```

- 3 Setzen Sie in der Informationsdatei für den Verzeichnisdienst, die für JNDI erforderlich ist, die Verbindungseigenschaft REQUEST_HA_SESSION, um eine Notumschaltungssitzung für jede Verbindung zu aktivieren, die Sie aufbauen.

Die neue Verbindungseigenschaft REQUEST_HA_SESSION zeigt an, dass der Client eine Notumschaltungssitzung mit Adaptive Server Version 12.0 beginnen will, der für Notumschaltung konfiguriert ist. Wenn diese Eigenschaft auf "true" gesetzt wird, versucht jConnect eine Notumschaltungsanmeldung. Wenn Sie diese Verbindungseigenschaften nicht setzen, startet keine Notumschaltungssitzung, auch wenn der Server korrekt konfiguriert ist. Der Standardwert für REQUEST_HA_SESSION ist "false".

Setzen Sie die Verbindungseigenschaften wie jede andere Verbindungseigenschaft. Sie können die Eigenschaft nicht zurücksetzen, sobald eine Verbindung aufgebaut ist.

Wenn Sie größere Flexibilität beim Anfordern von HA-Notumschaltungssitzungen wollen, codieren Sie die Client-Anwendung so, dass REQUEST_HA_SESSION zur Laufzeit festgelegt wird.

Das folgende Beispiel zeigt Verbindungsinformationen, die für den Datenbankserver SYBASE11 unter einem LDAP-Verzeichnisdienst eingegeben werden:

```
dn: servername=SYBASE11,o=MyCompany,c=US
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=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1#moorea 6000
```

Hierbei gilt: "tahiti" ist der primäre Server und "moorea" ist der sekundäre Begleitserver.

4 Fordern Sie eine Verbindung mit Hilfe von JNDI und LDAP an.

- jConnect verwendet den Verzeichnisdienst des LDAP-Servers, um den Namen und die Position des primären und sekundären Servers zu ermitteln.

```
/* Verbindung aufnehmen */
Connection con = DriverManager.getConnection
( "jdbc:sybase:jndi:ldap://ldap_server1:983" +
  "/servername=Sybasell,o=MyCompany,c=US",props );
```

Alternative:

- Geben Sie eine Basis für die Suche an:

```
props.put(Context.PROVIDER_URL,
  "ldap://ldap_server1:983/ o=MyCompany, c=US");

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

Beim primären Server anmelden

Wenn eine Adaptive Server-Installation nicht für die Notumschaltung konfiguriert ist oder eine Notumschaltungssitzung aus irgend einem Grund nicht gewähren kann, darf sich der Client nicht anmelden, und die folgende Warnung wird angezeigt:

```
'Der Server hat die angeforderten Hochverfügbarkeits-
funktionen zurückgewiesen.'
```


Konfigurieren Sie Ihre Datenbank neu oder fordern Sie keine Hochverfügbarkeits-Sitzung an.'

Zum sekundären Server umschalten

Wenn eine Notumschaltung erfolgt, wird eine Ausnahmebedingung ausgegeben, und der Client verbindet sich automatisch mit Hilfe von JNDI mit der sekundären Datenbank.

Beachten Sie Folgendes:

- Die Identität der Datenbank, mit der der Client verbunden war, und alle festgeschriebenen Transaktionen werden beibehalten.
- Nur teilweise gelesene Ergebnismengen, Cursor und Aufrufe von gespeicherten Prozeduren gehen verloren.
- Wenn eine Notumschaltung erfolgt, muss Ihre Anwendung möglicherweise eine Prozedur erneut starten oder zur letzten vollständigen Transaktion oder Aktivität zurückgehen.

Zum primären Server zurückschalten

Der Client schaltet an einem bestimmten Punkt vom sekundären Server zurück zum primären. Wann eine Rückschaltung ausgeführt wird, bestimmt der Systemadministrator, der **sp_failback** auf dem sekundären Server ausführt. Danach kann der Client mit demselben Verhalten und denselben Ergebnissen auf dem primären Server rechnen, wie unten beschrieben. ["Zum sekundären Server umschalten" auf Seite 45](#)

Remote Procedure Call von Server zu Server ausführen

Ein Befehl oder eine gespeicherte Prozedur in der Transact-SQL-Sprache, die auf einem Server laufen, können eine gespeicherte Prozedur ausführen, die sich auf einem anderen Server befindet. Der Server, mit dem sich die Anwendung verbunden hat, meldet sich beim Fremdserver an und führt einen entfernten Prozeduraufruf (Remote Procedure Call) von Server zu Server durch.

Eine Anwendung kann ein "universelles" Kennwort für die Kommunikation von Server zu Server einrichten. Dieses Kennwort wird in allen Verbindungen von Server zu Server benutzt. Wenn die Verbindung eingerichtet wurde, benutzt der Server dieses Kennwort, um sich bei einem beliebigen Fremdserver anzumelden.

Standardmäßig benutzt jConnect das Kennwort der aktuellen Verbindung als Standardkennwort für die Kommunikation von Server zu Server.

Wenn die Kennwörter, die ein Benutzer auf den einzelnen Servern benutzt, unterschiedlich sind, und dieser Benutzer Remote Procedure Calls von Server zu Server ausführt, muss die Anwendung für jeden Server, den sie verwenden möchte, explizit ein Kennwort definieren.

jConnect Version 4.1 und höher enthält eine Eigenschaft, mit der Sie ein universelles "entferntes" Kennwort bzw. unterschiedliche Kennwörter für mehrere Server setzen können. Sie können diese Eigenschaft in jConnect setzen und konfigurieren, indem Sie die Methode **setRemotePassword()** in der Klasse **SybDriver** verwenden:

```
Properties connectionProps = new Properties();

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

Um diese Methode zu benutzen, muss die Anwendung die Klasse **SybDriver** unterstützen und dann die Methode aufrufen.

Für jConnect 4.x:

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

Für jConnect 5.x:

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

Hinweis Um unterschiedliche entfernte Kennwörter für diverse Server zu setzen, wiederholen Sie den vorstehenden Aufruf (je nach Version von jConnect) für jeden Server.

Dieser Aufruf fügt das angegebene Servername/Kennwort-Paar dem vorhandenen Objekt **Properties** hinzu, welches dann von der Anwendung an **DriverManager** in **DriverManager.getConnection (Server_URL, Eigenschaften)** übergeben werden kann.

Wenn **serverName** NULL ist, wird das universelle Kennwort für nachfolgende Verbindungen zu allen Servern auf **password** gesetzt, mit Ausnahme derjenigen, die durch frühere Aufrufe spezifisch für **setRemotePassword()** gesetzt wurden.

Wenn eine Anwendung die Eigenschaft REMOTEPWD setzt, liefert jConnect das universelle Kennwort nicht mehr.

Zugriff auf Datenbank-Metadaten

Um die **DatabaseMetaData**-Methoden zu unterstützen, bietet Sybase eine Gruppe von gespeicherten Prozeduren, die jConnect für die Metadaten einer Datenbank aufrufen kann. Diese Gespeicherte Prozeduren müssen auf dem Server installiert sein, damit die JDBC-Metadaten-Methoden funktionieren.

Wenn die gespeicherten Prozeduren für die Bereitstellung der Metadaten auf einem Sybase-Server noch nicht installiert sind, können Sie sie mit einem von zwei Skripten für die Installation von gespeicherten Prozeduren installieren, die mit jConnect mitgeliefert werden:

- *sql_server.sql* installiert gespeicherte Prozeduren in Datenbanken der Vorversion 12.0 von Adaptive Server.
- *sql_server12.sql* installiert gespeicherte Prozeduren in Datenbanken von Adaptive Server Version 12.0.
- *sql_anywhere.sql* installiert gespeicherte Prozeduren in einer Datenbank auf Adaptive Server Anywhere.

Hinweis Die jeweils neueste Version dieser Skripten ist mit allen Versionen von jConnect kompatibel.

Umfassende Hinweise zur Installation von gespeicherten Prozeduren finden Sie in der Dokumentation *Sybase jConnect for JDBC Installationsanleitung*.

Um die Metadaten-Methoden zu installieren, muss außerdem die Verbindungseigenschaft USE_METADATA auf "true" (Standardwert) gesetzt werden, wenn Sie eine Verbindung einrichten.

Über temporäre Tabellen in einer Datenbank können keine Metadaten bezogen werden.

Hinweis Die Methode **DatabaseMetaData.getPrimaryKeys()** findet Primärschlüssel, die in einer Tabellendefinition (CREATE TABLE) oder mit "alter table" (ALTER TABLE ADD CONSTRAINT) deklariert werden. Sie findet hingegen keine Schlüssel, die mit **sp_primarykey** definiert wurden.

Serverseitige Metadateninstallation

Die Metadata-Unterstützung kann auf der Clientseite (ODBC, JDBC) oder in der Datenquelle (gespeicherte Prozeduren auf dem Server) eingerichtet werden. jConnect bietet Metadaten-Unterstützung auf dem Server, wodurch sich folgende Vorteile ergeben:

- Das geringe Volumen von jConnect bleibt gewahrt, so dass der Treiber vom Internet rasch heruntergeladen werden kann.
- Die Laufzeitwirkung wird durch vorher geladene gespeicherte Prozeduren auf der Datenquelle verbessert.
- Bietet Flexibilität. jConnect kann sich mit einer Vielzahl von Datenbanken verbinden.

Cursor mit Ergebnismengen verwenden

jConnect 5.x implementiert viele JDBC 2.0 Cursor- und Aktualisierungsmethoden. Diese Methoden vereinfachen den Einsatz eines Cursors und die Aktualisierung von Zeilen in einer Tabelle basierend auf Werten in einer Ergebnismenge.

Hinweis Wenn Sie volle JDBC 2.0 Unterstützung benötigen, verwenden Sie jConnect Version 5.x oder höher. jConnect Version 4.x bietet einige JDBC 2.0-Funktionen für Sybase-Erweiterungen und den Beispielcode

ScrollableResultSet.java im Unterverzeichnis *sample* des jConnect-Installationsverzeichnis. Javadocs für diese Methoden finden Sie in den Paketen **com.sybase.jdbcx** und **sample**.

In JDBC 2.0 werden **ResultSets** durch ihren Typ und ihre Parallelität gekennzeichnet. Die Werte für Typ und Parallelität sind Teil der **java.sql.ResultSet**-Schnittstelle und werden in den jeweiligen javadocs beschrieben.

[Tabelle 2-5](#) zeigt die Eigenschaften von **java.sql.ResultSet**, die in jConnect 5.x verfügbar sind.

Tabelle 2-5: In jConnect 5.x verfügbare Optionen für java.sql.ResultSet

Parallelität	Typ		
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
CONCUR_READ_ONLY	Unterstützt in 5.x	Unterstützt in 5.x	Nicht verfügbar in 5.x
CONCUR_UPDATABLE	Unterstützt in 5.x	Nicht verfügbar in 5.x	Nicht verfügbar in 5.x

Dieser Abschnitt behandelt folgende Themen:

- [Cursor erstellen](#)
- [Positionierte Aktualisierungen und Löschungen mit JDBC 1.x Methoden](#)
- [Cursor mit einem PreparedStatement verwenden](#)
- [Unterstützung für SCROLL_INSENSITIVE ResultSets in jConnect](#)

Cursor erstellen

Um mit jConnect 4.x einen Cursor zu erstellen, benötigen Sie **SybStatement.setCursorName()** oder **SybStatement.setFetchSize()**. Wenn Sie **SybStatement.setCursorName()** benutzen, wird dem Cursor explizit ein Name zugeordnet. Die Signatur für **SybStatement.setCursorName()** ist:

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

Sie verwenden **SybStatement.setFetchSize()**, um einen Cursor zu erstellen und die Anzahl der Zeilen festzulegen, die bei jedem Fetch aus der Datenbank ausgelesen werden. Die Signatur für **SybStatement.setFetchSize()** ist:

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

Wenn Sie **setFetchSize()** benutzen, um einen Cursor zu erstellen, gibt der jConnect-Treiber dem Cursor einen Namen. Um den Namen des Cursors zu beziehen, benutzen Sie **ResultSet.setCursorName()**.

In `jdbc` Version 5.x erstellen Sie einen `Cursor` auf dieselbe Weise wie in Version 4.x. Da Version 5.x aber `JDBC 2.0` unterstützt, gibt es eine weitere Methode, um einen `Cursor` zu erstellen. Sie können festlegen, welche Art von **ResultSet** von einer Anweisung zurückgegeben werden soll, indem Sie die folgende `JDBC 2.0`-Methode auf der Verbindung benutzen:

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

Der Typ und die Parallelität entsprechen den Typen und Parallelitäten der Schnittstelle **ResultSet**, die in [Tabelle 2-5](#) aufgelistet ist. Wenn Sie nicht unterstützte **ResultSet** anfordern, wird auf der Verbindung eine `SQL`-Warnung ausgegeben. Wenn das zurückgegebene **Statement** ausgeführt wird, erhalten Sie die Art des **ResultSets**, das dem angeforderten Typ am ehesten entspricht. Weitere Hinweise zum Verhalten dieser Methode finden Sie in der `JDBC 2.0`-Spezifikation.

Wenn Sie **createStatement()** nicht verwenden oder `jdbc` Version 4.x benutzen, sind die Standardtypen von **ResultSet** wie folgt:

- Wenn Sie nur **Statement.executeQuery()** aufrufen, dann ist das zurückgegebene **ResultSet** ein **SybResultSet** mit `TYPE_FORWARD_ONLY` und `CONCUR_READ_ONLY`.
- Wenn Sie **setFetchSize()** oder **setCursorName()** aufrufen, dann ist das von **executeQuery()** zurückgegebene **ResultSet** ein **SybCursorResultSet** mit `TYPE_FORWARD_ONLY` und `CONCUR_UPDATABLE`.

Um zu prüfen, ob der Typ des **ResultSet**-Objekts dem beabsichtigten entspricht, hat die `JDBC 2.0` API für **ResultSet** zwei Methoden hinzugefügt:

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

Folgende Schritte sind für die Erstellung und die Verwendung eines `Cursors` auszuführen:

- 1 Erstellen Sie den `Cursor` mit **Statement.setCursorName()** oder **SybStatement.setFetchSize()**.
- 2 Rufen Sie **Statement.executeQuery()** auf, um den `Cursor` für eine Anweisung zu öffnen und eine `Cursor`-Ergebnismenge zurückzugeben.
- 3 Rufen Sie **ResultSet.next()** auf, um Zeilen abzurufen und den `Cursor` in der Ergebnismenge zu positionieren.

Das folgende Beispiel benutzt beide Methoden für die Erstellung eines Cursors und die Rückgabe einer Ergebnismenge. Es benutzt auch **ResultSet.setCursorName()**, um den Namen des Cursors zu übernehmen, der von **SybStatement.setFetchSize()** erstellt wurde.

```
// Mit conn als Verbindungsobjekt erstelle ein
// Statement-Objekt und ordne es mit
// Statement.setCursorName() einem Cursor zu.
Statement stmt = conn.createStatement();
stmt.setCursorName("author_cursor");

// Verwende das Statement zum Ausführen einer
// Abfrage und Rückgabe eines Cursor-ResultSets.
ResultSet rs = stmt.executeQuery("SELECT au_id,
    au_lname, au_fname FROM authors
    WHERE city = 'Oakland'");
while(rs.next())
{
    ...
}

// Erstelle ein zweites Statement-Objekt und
// erstelle mit SybStatement.setFetchSize() einen
// Cursor, der je 10 Zeilen zurückgibt.
SybStatement syb_stmt = conn.createStatement();
syb_stmt.setFetchSize(10);

// Benutze syb_stmt zur Ausführung einer Abfrage
// Abfrage und Rückgabe eines Cursor-ResultSets.
SybCursorResultSet rs2 =
    (SybCursorResultSet)syb_stmt.executeQuery
    ("SELECT au_id, au_lname, au_fname FROM authors
    WHERE city = 'Pinole'");
while(rs2.next())
{
    ...
}

// Besorge den Namen des mit der Methode
// setFetchSize() erstellten Cursors.
String cursor_name = rs2.setCursorName();
...
// Erstelle ein drittes Statement-Objekt für jCon-
// nect 5.x
// mit der neuen Methode auf Connection
// und übernahm ein SCROLL_INSENSITIVE ResultSet.
```

```
// Hinweis: downcast des Statements oder des
// ResultSets ist nicht mehr erforderlich.
Statement stmt = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
ResultSet rs3 = stmt.executeQuery
    ("SELECT ... [whatever]");
// Führe eine der JDBC 2.0 Methoden aus, die
// für Read Only ResultSets zulässig sind.
rs3.next();
rs3.previous();
rs3.relative(3);
rs3.afterLast();
...
```

Positionierte Aktualisierungen und Löschungen mit JDBC 1.x Methoden

Im folgenden Beispiel wird gezeigt, wie die Methoden in JDBC 1.x benutzt werden können, um eine positionsbasierte Aktualisierung durchzuführen. Das Beispiel erstellt zwei **Statement**-Objekte, eines für die Selektion von Zeilen in eine Cursor-Ergebnismenge und das zweite für eine Aktualisierung der Datenbank mit Zeilen aus der Ergebnismenge.

```
// Erstelle zwei Statement-Objekte und einen
// Cursor für das vom ersten Statement stmt1
// zurückgegebene ResultSet. Benutze stmt1
// zum Ausführen einer Abfrage und Rückgabe
// eines Cursor-ResultSets.
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");

// Besorge den Namen des mit stmt1 erstellten
// Cursors, damit er mit stmt2 verwendet
// werden kann.
String cursor = rs.getCursorName();

// Benutze stmt2 zur Aktualisierung der Datenbank aus
dem
// von stmt1 zurückgegebenen ResultSet.
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);
    }
}
```

Spalten in einer Ergebnismenge löschen

Das folgende Beispiel benutzt das **Statement**-Objekt *stmt2* aus dem vorherstehenden Programmcodebeispiel für ein positionsbasiertes Löschen:

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

Positionierte Aktualisierungen und Löschungen mit JDBC 2.0 Methoden

In diesem Abschnitt werden Methoden nach dem Standard JDBC 2.0 für die Aktualisierung von Spalten in der aktuellen Cursorzeile und die Aktualisierung der Datenbank aus der aktuellen Cursorzeile in einer Ergebnismenge besprochen. Danach folgt ein Beispiel.

Spalten in einer Ergebnismenge aktualisieren

JDBC 2.0 legt eine Reihe von Methoden fest, mit denen Spaltenwerte aus einer Ergebnismenge im Speicher auf dem Client aktualisiert werden können. Die aktualisierten Werte können dann benutzt werden, um eine Aktualisierung, eine Einfügung oder eine Löschung in einer der Basisdatenbanken auszuführen. Alle diese Methoden sind in der Klasse **SybCursorResultSet** implementiert.

Beispiele für einige JDBC-2.0-Aktualisierungsmethoden, die in jConnect verfügbar sind:

```
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;
```

Methoden für die Aktualisierung der Datenbank aus einer Ergebnismenge

JDBC 2.0 legt zwei neue Methoden für die Aktualisierung oder das Löschen von Zeilen in der Datenbank, basierend auf den aktuellen Werten in einer Ergebnismenge fest. Diese Methoden sind formell einfacher als **Statement.executeUpdate()** in JDBC 1.x und benötigen keinen Cursornamen. Sie sind in der Klasse **SybCursorResultSet** implementiert:

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

Hinweis Die Parallelität der Ergebnismenge muss **CONCUR_UPDATABLE** sein, da sonst bei den oben genannten Methoden eine Ausnahmebedingung ausgegeben wird. Für **insertRow()** müssen alle Tabellenspalten angegeben sein, bei denen Nullwerte nicht zulässig sind.

Die für **DatabaseMetaData** gelieferten Methoden bestimmen, wann diese Änderungen sichtbar sind.

Beispiel

Das folgende Beispiel erstellt ein einzelnes **Statement**-Objekt, das benutzt wird, um eine Cursor-Ergebnismenge zurückzugeben. Für jede Zeile in der Ergebnismenge erfolgt erst eine Aktualisierung der Spaltenwerte im Speicher, und danach wird die Datenbank mit den neuen Spaltenwerten der Zeile aktualisiert.

```
// Erstelle ein Statement-Objekt und setze die  
// Fetch-Größe auf 25. Damit wird ein Cursor  
// für das Statement-Objekt erstellt. Benutze  
// es, um die Ergebnismenge zurückzugeben.  
SybStatement syb_stmt =  
(SybStatement)conn.createStatement();  
syb_stmt.setFetchSize(25);  
SybCursorResultSet syb_rs =  
(SybCursorResultSet)syb_stmt.executeQuery(  
    "SELECT * from T1 WHERE ...")  
  
// Aktualisiere alle Zeilen im ResultSet gemäß dem  
// Code in der folgenden while-Schleife. jConnect  
// ruft je 25 Zeilen ab, bis nur mehr weniger als  
// 25 Zeilen bleiben. Der letzte Fetch ruft alle  
// verbleibenden Zeilen ab.  
while(syb_rs.next())  
{  
    // Aktualisiere Spalten 2 und 3 jeder Zeile,  
    // wenn Spalte 2 ein 'varchar' in der Datenbank  
    // und Spalte 3 ein 'integer' Wert ist.
```

```

syb_rs.updateString(2, "xyz");
syb_rs.updateInt(3,100);
// Jetzt aktualisiere die Zeile in der Datenbank.
syb_rs.updateRow();
}
// Erstelle ein Statement-Objekt mit der
// in jConnect 5.x implementierten JDBC 2.0-Methode
Statement stmt = conn.createStatement
(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
// Benutze Statement zur Rückgabe des aktualisierbaren ResultSets
ResultSet rs = stmt.executeQuery("SELECT * FROM T1 WHERE...");
// In jConnect 5.x ist downcasting auf bSybCursorResultSet nicht
// erforderlich. Jede Zeile im ResultSet auf dieselbe Weise wie
// oben aktualisieren
while (rs.next())
{
rs.updateString(2, "xyz");
rs.updateInt(3,100);
rs.updateRow();
}

```

Zeile aus einer Ergebnismenge löschen

Um eine Spalte aus einer Cursor-Ergebnismenge zu löschen, können Sie **SybCursorResultSet.deleteRow()** wie folgt verwenden:

```

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

```

Zeile in eine Ergebnismenge einfügen

Das folgende Beispiel zeigt, wie Einfügungen mit der JDBC 2.0 API erfolgen, die nur in jConnect 5.x verfügbar ist. Ein downcast zu einem **SybCursorResultSet** ist nicht erforderlich.

```

// Vorbereitung zum Einfügen
rs.moveToInsertRow();
// Neue Zeile mit Spaltenwerten füllen
rs.updateString(1, "Neuer Eintrag für col 1");
rs.updateInt(2, 42);

```

```
// Neue Zeile in db einfügen
rs.insertRow();
// Rückkehr zur aktuellen Zeile in der
// Ergebnismenge
rs.moveToCurrentRow();
```

Cursor mit einem PreparedStatement verwenden

Wenn ein **PreparedStatement**-Objekt erstellt wurde, kann es mehrere Male mit denselben oder veränderten Eingabeparameterwerten benutzt werden. Wenn Sie einen Cursor mit einem **PreparedStatement**-Objekt verwenden, müssen Sie den Cursor nach jedem Einsatz schließen und dann wieder öffnen, um ihn wieder verwenden zu können. Ein Cursor wird geschlossen, wenn Sie seine Ergebnismenge (ResultSet) schließen (**ResultSet.close()**). Er wird geöffnet, wenn Sie sein Prepared Statement ausführen (**PreparedStatement.executeQuery()**).

Das folgende Beispiel zeigt, wie ein **PreparedStatement**-Objekt erstellt wird, wie man ihm einen Cursor zuweist und dann das **PreparedStatement**-Objekt zweimal ausführt, indem der Cursor geschlossen und dann wieder geöffnet wird.

```
// Erstelle ein Prepared Statement-Objekt mit
// einer parametergesteuerten Abfrage.
PreparedStatement prep_stmt =
conn.prepareStatement(
"SELECT au_id, au_lname, au_fname "+
"FROM authors WHERE city = ? "+
"FOR UPDATE OF au_lname");

// Erstelle einen Cursor für das Prep-Statement.
prep_stmt.setCursorName("author_cursor");

// Weise dem Parameter in der Abfrage einen Wert
// zu. Führe das Prep-Statement aus, um eine
// Ergebnismenge zurückzugeben.
prep_stmt.setString(1, "Oakland");
ResultSet rs = prep_stmt.executeQuery();

// Verarbeite die Ergebnismenge.
while(rs.next())
{
    ...
}

// Schließe das ResultSet und damit den Cursor.
```

```
rs.close();

// Die vorbereitete Anweisung nochmals ausführen mit
// einem neuen Parameterwert.
prep_stmt.setString(1,"San Francisco");
rs = prep_stmt.executeQuery();
// Öffnet den Cursor erneut
```

Unterstützung für **SCROLL_INSENSITIVE** ResultSets in jConnect

jConnect Version 5.x unterstützt nur **TYPE_SCROLL_INSENSITIVE** Ergebnismengen.

jConnect benutzt den Tabular Data Stream (TDS), das von Sybase bereitgestellte Protokoll, zur Kommunikation mit Sybase-Datenbankservern. Ab jConnect 5.x unterstützt TDS keine abrollbaren Cursor. Um abrollbare Cursor zu unterstützen, setzt jConnect die Zeilendaten auf Anforderung auf dem Client bei jedem Aufruf von **ResultSet.next()** in den Cache. Wenn jedoch das Ende der Ergebnismenge erreicht ist, wird die komplette Ergebnismenge im Speicher des Clients gespeichert. Da dies eine Beeinträchtigung der Performance nach sich ziehen kann, empfehlen wir, **TYPE_SCROLL_INSENSITIVE**-Ergebnismengen nur zu verwenden, wenn die Ergebnismenge entsprechend klein ist.

Hinweis Wenn Sie **TYPE_SCROLL_INSENSITIVE ResultSets** in jConnect 5.x verwenden, können Sie die Methode **isLast()** erst aufrufen, nachdem die letzte Zeile der **ResultSet** gelesen wurde. Wenn Sie **isLast()** aufrufen, bevor die letzte Zeile erreicht wurde, wird der Fehler **UnimplementedOperationException** ausgegeben.

Ein Codebeispiel wurde in jConnect Version 4.x eingefügt, das ein begrenztes **TYPE_SCROLL_INSENSITIVE ResultSet** mit JDBC 1.0 Schnittstellen bereitstellt.

Diese Implementierung benutzt Standardmethoden gemäß JDBC 1.0 zur Erstellung eines "scroll-insensitive, read-only" ResultSets, d.h. also eine statische Sicht der Basisdaten, die gegen Änderungen unempfindlich ist, während die Ergebnismenge offen ist. **ExtendedResultSet** setzt alle **ResultSet**-Zeilen in den Cache auf dem Client. Wenn Sie große Ergebnismengen verwenden, gehen Sie mit dieser Klasse vorsichtig um.

Die Schnittstelle **sample.ScrollableResultSet**:

- Sie ist eine Erweiterung des JDBC 1.0 **java.sql.ResultSet**.

- Sie definiert zusätzliche Methoden, die dieselben Signaturen haben wie das JDBC 2.0 **java.sql.ResultSet**.
- Sie enthält *keine* JDBC 2.0 Methoden. Die fehlenden Methoden dienen zur Änderung des **ResultSets**.

Die Methoden von der JDBC 2.0 API, die *tatsächlich* definiert sind:

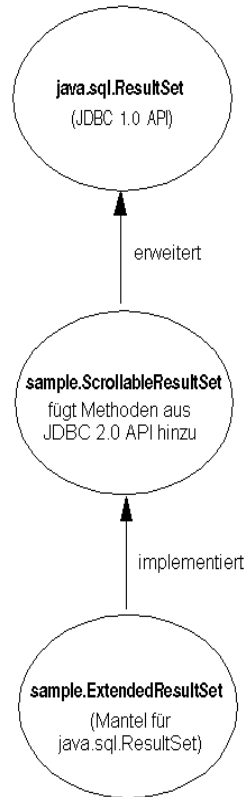
```
boolean previous() throws SQLException;
boolean absolute(int row) throws SQLException;
boolean relative(int rows) throws SQLException;
boolean first() throws SQLException;
boolean last() throws SQLException;
void beforeFirst() throws SQLException;
void afterLast() throws SQLException;
boolean isFirst() 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;
```

Um die neuen Beispiellklassen zu verwenden, erstellen Sie ein **ExtendedResultSet** mit einem beliebigen JDBC 1.0 **java.sql.ResultSet**.
Nachstehend finden Sie die diesbezüglichen Programmcodausschnitte (Java 1.1 Umgebung wird vorausgesetzt):

```
// Beispiellcode dateien importieren
import sample.*;
// JDBC 1.0 Klassen importieren
import java.sql.*;
// Mit einer db über einen Treiber verbinden
// Anweisung und Abfrage erstellen
// Referenz zu einem JDBC 1.0 ResultSet holen
ResultSet rs = stmt.executeQuery(_query);
// ScrollableResultSet damit erstellen
ScrollableResultSet srs = new ExtendedResultSet(rs);
// Methoden von der JDBC 2.0 API aufrufen
srs.beforeFirst();
// oder Methoden aus der JDBC 1.0 API aufrufen
if (srs.next())
    String column1 = srs.getString(1);
```

Abbildung 2-1 ist ein Klassendiagramm, das die Beziehungen zwischen den neuen Beispielklassen und der JDBC API zeigt.

Abbildung 2-1: Klassendiagramm



Weitere Hinweise finden Sie unter der JDBC 2.0 API an der Adresse <http://java.sun.com/products/jdbc/jdbcse2.html>.

Unterstützung von Aktualisierungen in Form von Anweisungsfolgen

Mit Aktualisierungen in Form einer Anweisungsfolge kann ein **Statement**-Objekt mehrfache Aktualisierungsbefehle en bloc an eine Datenbank übergeben, um sie gleichzeitig ausführen zu lassen.

Hinweis Um Aktualisierungen in Form von Anweisungsfolgen (Batches) verwenden zu können, müssen Sie die SQL-Skripten im Verzeichnis *sp* Ihres jConnect-Installationsverzeichnis aktualisieren.

Ein Beispiel für die Verwendung von Aktualisierungen in Form von Anweisungsfolgen mit **Statement**, **PreparedStatement** und **CallableStatement** finden Sie in *BatchUpdates.java* im Unterverzeichnis *sample* (jConnect 4.x) und *sample2* (jConnect 5.x).

jConnect unterstützt auch dynamische **PreparedStatement**s in Form von Anweisungsfolgen.

Hinweise zur Implementierung

jConnect implementiert Aktualisierungen in Form von Anweisungsfolgen gemäß der Spezifikation in der JDBC 2.0 API, mit nachstehenden Ausnahmen:

- Wenn der JDBC 2.0 Standard für die Implementierung von **BatchUpdateException.getUpdateCounts()** in Zukunft geändert oder freigegeben wird, implementiert jConnect den Originalstandard, indem **BatchUpdateException.getUpdateCounts()** eine **int[]** Länge von $M < N$ zurückgibt, wodurch angezeigt wird, dass die ersten M Anweisungen in der Anweisungsfolge erfolgreich waren, dass die Anweisung $M+1$ fehlgeschlagen ist und dass die Anweisungen $M+2..N$ nicht ausgeführt wurden. Hierbei entspricht "N" der Gesamtanzahl der Anweisungen in der Anweisungsfolge.

- Aktualisierungen in Form von Anweisungsfolgen für gespeicherte Prozeduren

Um gespeicherte Prozeduren in einer Anweisungsfolge (unverkettet) aufzurufen, müssen Sie die gespeicherte Prozedur im unverketteten Modus erstellen. Weitere Hinweise finden Sie unter ["Gespeicherte Prozedur im unverketteten Transaktionsmodus ausgeführt"](#) auf Seite 111.

- Adaptive Server Enterprise Version 11.5.x und höher

BatchUpdateException.getUpdateCounts() gibt nur eine **int[]** Länge Null zurück. Die komplette Transaktion wird zurückgesetzt, wenn ein Fehler auftritt, so dass insgesamt null erfolgreiche Zeilen vorhanden sind.

- Adaptive Server Enterprise Version 11.0.1
Gibt "0 (zero) rows affected" für gespeicherte Prozeduren zurück.
- SQL Anywhere Version 5.5.x
 - SQL Anywhere Version 5.5.x ermöglicht keine Rückgabe der Anzahl der eingefügten Zeilen aus gespeicherten Prozeduren, die Einfügungen enthalten. Beispiel:

```
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
```

Wenn **executeBatch** für die vorstehenden gespeicherten Prozeduren ausgeführt wird, kommt es zu folgenden Ergebnissen:

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

- In einer Anweisungsfolge gibt es keine Unterstützung von dynamischen **PreparedStatements**.
- Da SQL Anywhere 5.5.x in nativer Form keine Aktualisierungen in Form von Anweisungsfolgen gemäß der JDBC 2.0 Spezifikation unterstützt, werden Aktualisierungen in Form von Anweisungsfolgen in einer **executeUpdate** Schleife ausgeführt.
- Aktualisierungen in Form von Anweisungsfolgen bei Datenbanken, die keine Batch-Aktualisierungen unterstützen
jConnect führt Aktualisierungen in Form von Anweisungsfolgen in einer **executeUpdate** Schleife aus, auch wenn Ihre Datenbank keine Aktualisierungen in Form von Anweisungsfolgen unterstützt. Sie können daher denselben Batch-Programmcode für alle Datenbanken benutzen.

Siehe *Sun Microsystems, Inc. JDBC™ 2.0 API* mit weiteren Hinweisen zu Aktualisierungen in Form von Anweisungsfolgen.

Datenbank aus der Ergebnismenge einer gespeicherten Prozedur aktualisieren

jConnect enthält Aktualisierungs- und Löschungsmethoden, um in der von einer gespeicherten Prozedur zurückgegebenen Ergebnismenge einen Cursor zu setzen. Sie können dann die Cursorposition benutzen, um Zeilen in der Basistabelle zu aktualisieren oder zu löschen, aus der die Ergebnismenge extrahiert wurde. Die Methoden befinden sich in der Klasse

SybCursorResultSet:

`void updateRow(String Tabellenname) throws SQLException;`

`void deleteRow(String Tabellenname) throws SQLException;`

Der Parameter *Tabellenname* steht für die Datenbanktabelle, aus der die Ergebnismenge bezogen wurde.

Wenn Sie einen Cursor in der Ergebnismenge setzen wollen, die von einer gespeicherten Prozedur zurückgegeben wurde, müssen Sie vor dem Ausführen des Callable-Statements, das die Prozedur enthält, entweder

SybCallableStatement.setCursorName() oder

SybCallableStatement.setFetchSize() benutzen. Das folgende Beispiel zeigt, wie ein Cursor in der Ergebnismenge einer gespeicherten Prozedur erstellt wird, die Werte in der Ergebnismenge aktualisiert werden, und dann eine Aktualisierung der Basistabelle mit der Methode

SybCursorResultSet.update() erfolgt:

```
// Erstelle ein CallableStatement-Objekt
// zum Ausführen der gespeicherten Prozedur.
CallableStatement sproc_stmt =
    conn.prepareCall("{call update_titles}");

// Setze die Anzahl der mit jedem Fetch aus der Datenbank
// zurückzugebenden Zeilen. Dies erstellt einen Cursor
// im ResultSet.
(SybCallableStatement)sproc_stmt.setFetchSize(10);

// Führe die gespeicherte Prozedur aus und beziehe ein ResultSet.
SybCursorResultSet sproc_result = (SybCursorResultSet)
    sproc_stmt.executeQuery();

// Gehe Zeile für Zeile durch das ResultSet und aktualisiere Werte
// in der aktuellen Zeile des Cursors und aktualisiere die
// Basistabelle 'titles' mit den geänderten Zeilenwerten.
while(sproc_result.next())
{
    sproc_result.updateString(...);
    sproc_result.updateInt(...);
}
```

```

    ...
    sproc_result.updateRow(titles);
}

```

Mit Datenbanken arbeiten

Bilddaten senden

jConnect enthält eine **TextPointer**-Klasse mit **sendData()**-Methoden für die Aktualisierung einer *image*-Spalte in einer Adaptive-Server-Enterprise- oder Adaptive-Server-Anywhere-Datenbank. In früheren Versionen von jConnect mussten Bilddaten mit der **setBinaryStream()**-Methode in **java.sql.PreparedStatement** gesendet werden. Die **TextPointer.sendData()**-Methoden benutzen **java.io.InputStream** und verbessern die Performance deutlich, wenn Bilddaten an eine Adaptive-Server-Datenbank gesendet werden.

Um Instanzen der **TextPointer**-Klasse zu erhalten, können Sie eine der beiden **getTextPtr()**-Methoden in **SybResultSet** verwenden:

```

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

```

Public-Methoden in der TextPointer-Klasse

Das **com.sybase.jdbc**-Paket enthält die **TextPointer**-Klasse. Die Schnittstelle für die Public-Methode ist:

```

    public void sendData(InputStream EingabeStream, boolean Log)
        throws SQLException
    public void sendData(InputStream EingabeStream, int Länge,
        boolean Log) throws SQLException
    public void sendData(InputStream EingabeStream, int Offset,
        int Länge, boolean Log) throws SQLException
    public void sendData(byte[] Byte-Eingabe, int Offset,
        int Länge, boolean Log) throws SQLException

```

sendData(InputStream *EingabeStream*, boolean *Log*) – Aktualisiert eine image-Spalte mit Daten im angegebenen Eingabestream.

sendData(InputStream *is*, int *Länge*, boolean *Log*) – Aktualisiert eine image-Spalte mit Daten im angegebenen Eingabestream. *Länge* ist die Anzahl der gesendeten Byte.

sendData(InputStream EingabeStream, int Offset, int Länge, boolean Log) – Aktualisiert eine image-Spalte mit Daten im angegebenen Eingabestream, wobei an der Byte-Position begonnen wird, die im Parameter *Offset* festgelegt wurde, und so lange fortgesetzt wird, wie dies im Parameter *Länge* definiert ist.

sendData(byte[] Byte-Eingabe, int Offset, int Länge, boolean Log) – Aktualisiert eine Spalte mit Bilddaten, die in der Byte-Tabelle enthalten sind, welche im Parameter *Byte-Eingabe* festgelegt wird. Die Aktualisierung beginnt am Byte-Ursprung, der im Parameter *Offset* festgelegt wird und setzt sich über die Anzahl von Byte fort, die im Parameter *Länge* definiert wurde.

Jede Methode hat einen *Log*-Parameter. Der Parameter *Log* gibt an, ob image-Daten im Datenbank-Transaktionslog vollständig protokolliert werden. Wenn der Parameter *Log* auf "true" gesetzt ist, wird das komplette Binärimage in das Transaktionslog geschrieben. Wenn der Parameter *Log* auf "false" gesetzt ist, wird der Aktualisierungsvorgang protokolliert, das Image selbst aber nicht in das Log geschrieben.

❖ **Image-Spalte mit *TextPointer.sendData()* aktualisieren**

So aktualisieren Sie eine Spalte mit Bilddaten:

- 1 Holen Sie ein **TextPointer**-Objekt für die Zeile und die Spalte, die aktualisiert werden sollen.
- 2 Benutzen Sie **TextPointer.sendData()**, um den Aktualisierungsbefehl auszuführen.

Die nächsten beiden Abschnitte zeigen diese Vorgehensweise anhand eines Beispiels. In diesem Beispiel werden image-Daten der Datei *Anne_Ringer.gif* gesendet, um die Spalte *pic* der Tabelle *au_pix* in der Datenbank *pubs2* zu aktualisieren. Die Aktualisierung gilt für die Zeile mit der "author ID" 899-46-2035.

TextPointer-Objekt
holen

text- und image-Spalten enthalten Zeitstempel und Seiten-Standortdaten, die von den Text- und Bilddaten getrennt registriert werden. Wenn Daten in einer text- oder image-Spalte ausgewählt werden, ist diese Zusatzinformation als Teil der Ergebnismenge "verborgen".

Ein **TextPointer**-Objekt für die Aktualisierung einer image-Spalte benötigt diese verborgenen Informationen, nicht aber den Image-Teil der Spaltendaten. Um diese Informationen zu erhalten, müssen Sie eine Select-Anweisung in einem **ResultSet**-Objekt ausführen und dann **SybResultSet.getTextPtr()** verwenden (siehe das Beispiel, das auf den nächsten Absatz folgt).

SybResultSet.getTextPtr() extrahiert Textzeiger-Informationen, ignoriert Bilddaten und erstellt ein **TextPointer**-Objekt.

Wenn eine Spalte ein bedeutendes Volumen von Bilddaten enthält, ist die Auswahl der Spalte für eine oder mehr Zeilen und das Warten auf das Einlesen der Daten wahrscheinlich ineffizient, da die Daten nicht benutzt werden. Sie können den Vorgang abkürzen, indem Sie den Befehl **set textsize** verwenden, um die Menge von Daten zu reduzieren, die in einem Paket zurückgegeben werden. Das folgende Programmcodebeispiel für das Holen eines **TextPointer**-Objekts umfasst die Verwendung von **set textsize** für diesen Zweck.

```
/*
 * Definiere eine Zeichenfolge für die Auswahl von Daten
 * der Spalte 'pic' für 'author ID' 899-46-2035.
 */
String getColumnData = "select pic from au_pix where au_id = '899-46-2035'";

/*
 * Benutze 'set textsize' zur Rückgabe von nur einem Byte
 * Spaltendaten an ein Statement-Objekt. Das Paket mit
 * den Spaltendaten enthält die 'verborgene' Information,
 * die für das Erstellen eines TextPointer-Objektes
 * erforderlich ist.
 */
Statement stmt= connection.createStatement();
stmt.executeUpdate("set textsize 1");

/*
 * 'Select' die Spaltendaten in ein ResultSet-Objekt -
 * verbinde ResultSet mit SybResultSet da die getTextPtr
 * Methode in SybResultSet ist, die ResultSet erweitert.
 */
SybResultSet rs = (SybResultSet)stmt.executeQuery(getColumnData);

/*
 * Positioniere den ResultSet-Cursor auf die zurückgegebenen Spaltendaten
 * und erstelle das gewünschte TextPointer-Objekt.
 */
rs.next();
TextPointer tp = rs.getTextPtr("pic");
```

```
/*
 * Wir wollen nur eine Zeile aktualisieren und brauchen den
 * minimum textsize set nicht für die nächste Rückgabe des
 * Servers und setzen daher textsize auf den Standardwert.
 */
stmt.executeUpdate("set textsize 0");
```

Die Aktualisierung
ausführen mit
TextPointer.sendData

Der folgende Programmcode benutzt das **TextPointer**-Objekt aus dem
vorherigen Abschnitt zur Aktualisierung der *pic*-Spalte mit Bilddaten in der
Datei *Anne_Ringer.gif*.

```
/*
 * Zuerst definiere einen Eingabestream für die Datei.
 */
FileInputStream in = new FileInputStream("Anne_Ringer.gif");

/*
 * Bereite Senden des Eingabestreams ohne Protokollierung der
 * Bilddaten im Transaktionslog vor.
 */
boolean log = false;

/*
 * Sende die Bilddaten in Anne_Ringer.gif zum Aktualisieren
 * der 'pic'-Spalte für 'author ID' 899-46-2035.
 */
tp.sendData(in, log);
```

Weitere Informationen finden Sie im Beispiel *TextPointers.java* im
Unterverzeichnis *sample* (jConnect 4.x) und *sample2* (jConnect 5.x) in Ihrem
jConnect-Installationsverzeichnis.

Date- und Time-Datentypen verwenden

JDBC benutzt drei Datentypen für die Zeitangabe: Time, Date und Timestamp. Adaptive Server benutzt nur einen Datentyp für die Zeitangabe, *datetime*, mit dem Datum und Uhrzeit angegeben werden, ähnlich wie im Datentyp Timestamp von JDBC. Der *datetime*-Datentyp von Adaptive Server unterstützt die Sekundenauflösung bis zu 1/300stel einer Sekunde.

Alle drei JDBC-Datentypen werden serverseitig als *datetime*-Datentypen behandelt. Ein JDBC-Timestamp ist im wesentlichen dasselbe wie ein *datetime* von Adaptive Server, daher erübrigt sich die Konvertierung. Für die Übersetzung eines JDBC-Time oder Date -Datentyps in oder von einem Server *datetime* -Datentyp ist eine Konvertierung erforderlich.

- Um Time in datetime zu konvertieren, wird das Datum 1 Jan 1970 hinzugefügt.
- Um Date in datetime zu konvertieren, wird "00:00:00" angehängt.
- Um eine datetime- in eine Date-Variable oder eine Time-Variable zu konvertieren, werden die nicht benutzten Daten herausgefiltert.

Hinweise zur Implementierung

- Der Datentyp Timestamp von JDBC unterscheidet sich vom Adaptive Server timestamp-Datentyp. Der timestamp-Datentyp von Adaptive Server ist ein eindeutiger varbinary-Wert, der benutzt wird, wenn Aktualisierungen mit der Strategie "optimistische Parallelität (optimistic concurrency)" durchgeführt werden.
- Wenn ein Wert als Time-Datentyp eingefügt wird, wird der Datumsteil nicht gebraucht, so dass der Wert nur mit einem Time-Datentyp zurückgeholt werden soll, niemals als Date- oder Timestamp-Datentyp.
- Wenn Sie **getObject()** mit einer Adaptive Server Anywhere *date* oder *time* Spalte verwenden, wird der Wert als JDBC *Timestamp* Datentyp zurückgegeben.

Char/Varchar/Text-Datentypen und **getBytes()**

Verwenden Sie **rs.getBytes()** nicht in einem char-, varchar- oder text-Feld, es sei denn, die Daten wären Hex, Oktal oder Dezimal.

Erweiterte Funktionen implementieren

Dieser Abschnitt beschreibt, wie Sie erweiterte jConnect-Funktionen verwenden können. Er behandelt die folgenden Themen:

- [Ereignismeldungen benutzen](#)
- [Umgang mit Fehlermeldungen](#)
- [Java-Objekte als Spaltendaten in einer Tabelle speichern](#)
- [Dynamisches Laden von Klassen](#)
- [Unterstützung für die Erweiterungen des JDBC 2.0-Optionspakets](#)

Ereignismeldungen benutzen

Sie können die Ereignismeldungsfunktion von jConnect benutzen, damit Ihre Anwendung benachrichtigt wird, wenn eine Open-Server-Prozedur ausgeführt wird.

Um diese Funktion zu benutzen, müssen Sie die Klasse **SybConnection** verwenden, die die Schnittstelle **Connection** erweitert. **SybConnection** enthält eine Methode **regWatch()** für das Aktivieren und eine Methode **regNoWatch()** für das Deaktivieren der Ereignismeldung.

Ihre Anwendung muss auch die Schnittstelle **SybEventHandler** implementieren. Diese Schnittstelle enthält eine Public-Methode, **void event(String proc_name, ResultSet params)**, die aufgerufen wird, wenn das angegebene Ereignis eintritt. Die Parameter des Ereignisses werden an **event()** übermittelt, und diese Methode weist die Anwendung an, wie sie zu reagieren hat.

Wenn Sie die Ereignismeldung in Ihrer Anwendung verwenden wollen, rufen Sie **SybConnection.regWatch()** auf, um Ihre Anwendung in der Meldungsliste einer registrierten Prozedur zu registrieren. Verwenden Sie dabei folgende Syntax:

```
SybConnection.regWatch(Prozedurname,Ereignis-  
Steuerungsroutine,Option)
```

- *Prozedurname* ist eine Zeichenfolge die dem Namen der registrierten Prozedur entspricht, von der die Benachrichtigung generiert wird.
- *Ereignis-Steuerungsroutine* ist eine Instanz der Klasse **SybEventHandler**, die Sie implementieren.

- *Option* ist NOTIFY_ONCE oder NOTIFY_ALWAYS. Benutzen Sie NOTIFY_ONCE, wenn die Anwendung nur das erste Mal benachrichtigt werden soll, wenn eine Prozedur ausgeführt wird. Benutzen Sie NOTIFY_ALWAYS, wenn die Anwendung jedes Mal benachrichtigt werden soll, wenn die Prozedur ausgeführt wird.

Wenn ein Ereignis mit dem bezeichneten *Prozedurname* auf dem Open Server eintritt, ruft jConnect **eventHdlr.event()** aus einem separaten Thread auf. Die Ereignisparameter werden an **eventHdlr.event()** übergeben, wenn sie ausgeführt wird. Da es sich um einen separaten Thread handelt, blockiert die Ereignismeldung die Ausführung der Anwendung nicht.

Wenn *Prozedurname* keine registrierte Prozedur ist oder Open Server nicht in der Lage ist, den Client der Liste der Ereignismeldungen hinzuzufügen, wird beim Aufruf von **regWatch()** ein SQL-Ausnahmefehler generiert.

Wenn Sie die Ereignismeldung deaktivieren wollen, benutzen Sie folgenden Aufruf:

```
SybConnection.regNoWatch(proc_name)
```

Hinweis Wenn Sie Sybase-Erweiterungen für die Ereignismeldung benutzen, muss die Anwendung die **close()** Methode für die Verbindung benutzen, um einen abhängigen Thread zu entfernen, der mit dem ersten Aufruf von **regWatch()** erstellt wurde. Wenn dies nicht erfolgt, kann die Virtual Machine möglicherweise hängen bleiben, wenn die Anwendung beendet wird.

Beispiel für eine Ereignismeldung

Das folgende Beispiel zeigt, wie ein Ereignis-Steuerprogramm (EventHandler) implementiert und ein Ereignis mit einer Instanz des Ereignis-Steuerprogramms registriert werden kann, sobald eine Verbindung eingerichtet ist:

```
public class MyEventHandler implements SybEventHandler
{
    // Deklariere erforderliche Felder und Konstrukturen.
    ...
    public MyEventHandler(String eventname)
    {
        ...
    }

    // Implementiere SybEventHandler.event.
```

Erweiterte Funktionen implementieren

```
public void event(String eventName, ResultSet params)
{
    try
    {
        // Prüfe auf empfangene Fehlermeldungen vor der
        // Ereignismeldung.
        SQLWarning sqlw = params.getWarnings();
        if sqlw != null
        {
            // Verarbeite eventuelle Fehler
            ...
        }
        // Verarbeite Parameter wie ein ResultSet mit
        // einer Zeile.
        ResultSetMetaData rsmd = params.getMetaData();
        int numColumns = rsmd.getColumnCount();
        while (params.next())           // optional
        {
            for (int i = 1; i <= numColumns; i++)
            {
                System.out.println(rsmd.getColumnName(i) + " =
                    " + params.getString(i));
            }
            // Leite dem Ereignis angepasste Maßnahme ein. Beispiel:
            // Benachrichtige Anwendungs-Thread.
            ...
        }
    }
    catch (SQLException sqe)
    {
        // Verarbeite eventuelle Fehler
        ...
    }
}

public class MyProgram
{
    ...
    // Stelle Verbindung her und registriere ein Ereignis mit
    // einer Instanz von MyEventHandler.
    Connection conn = DriverManager.getConnection(...);
    MyEventHandler myHdlr = new MyEventHandler("MY_EVENT");

    // Registriere den Ereignis-Handler.
    ((SybConnection)conn).regWatch("MY_EVENT", myHdlr,
```

```

        SybEventHandler.NOTIFY_ALWAYS);
        ...
    conn.regNoWatch("MY_EVENT");
    conn.close();
}

```

Umgang mit Fehlermeldungen

jConnect bietet zwei Klassen für die Rückgabe von Sybase-spezifischen Fehlerinformationen, **SySQLException** und **SySQLWarning**, sowie die **SybMessageHandler**-Schnittstelle, mit dem Sie die Art verändern können, in der jConnect Fehlermeldungen behandelt, die vom Server kommen.

Abruf von Sybase-spezifischen Fehlerdaten

jConnect bietet eine **EedInfo**-Schnittstelle, die Methoden für den Abruf von Sybase-spezifischen Fehlermeldungen bereitstellt. Die **EedInfo**-Schnittstelle ist in **SySQLException** und **SySQLWarning** implementiert, die die Klassen **SQLException** und **SQLWarning** erweitern.

SySQLException und **SySQLWarning** enthalten folgende Methoden:

- **public ResultSet getEedParams();**
Gibt eine Ergebnismenge mit einer Zeile zurück, die Parameterwerte für die Fehlermeldung enthält.
- **public int getStatus();**
Gibt "1" zurück, wenn Parameterwerte vorhanden sind, und "0", wenn keine Parameterwerte in der Meldung enthalten sind.
- **public int getLineNumber();**
Gibt die Zeilennummer der gespeicherten Prozedur oder Abfrage zurück, die die Fehlermeldung verursacht haben.
- **public String getProcedureName();**
Gibt den Namen der Prozedur zurück, die die Fehlermeldung verursacht hat.
- **public String getServerName();**
Gibt den Namen des Servers zurück, der die Fehlermeldung generiert hat.
- **public int getSeverity();**

Gibt den Schweregrad der Fehlermeldung zurück.

- **public int getState();**

Gibt Informationen über die interne Quelle der Fehlermeldung im Server zurück. Diese Information ist nur für den Technischen Kundendienst von Sybase bestimmt.

- **public int getTranState();**

Gibt einen der folgenden Transaktionsstatuswerte zurück:

- 0 Die Verbindung ist derzeit eine erweiterte Transaktion.
- 1 Die vorherige Transaktion wurde erfolgreich festgeschrieben.
- 3 Die vorherige Transaktion wurde abgebrochen.

Beachten Sie, dass einige Fehlermeldungen **SQLException**- oder **SQLWarning**-Meldungen sein können, ohne **SybSQLException**- oder **SybSQLWarning**-Meldungen sein zu müssen. Ihre Anwendung muss den Typ der Ausnahmebedingung prüfen, die von ihr verarbeitet werden soll, bevor Sie ein Downcast zu **SybSQLException** oder **SybSQLWarning** durchführt.

Verarbeitung von Fehlermeldungen anpassen

Sie können die **SybMessageHandler**-Schnittstelle verwenden, um die Art zu verändern, in der jConnect Fehlermeldungen verarbeitet, die vom Server kommen. Die Implementierung von **SybMessageHandler** in Ihrer eigenen Klasse für die Verarbeitung von Fehlermeldungen kann folgende Vorteile haben:

- "Universelle" Fehlerbehandlung

Die Fehlerbehandlungslogik kann in Ihrem Fehlermeldungs-Steuerprogramm untergebracht werden, um zu vermeiden, dass sie überall in der Anwendung in Abfangklauseln ausprogrammiert wird.

- "Universelle" Fehlerprotokollierung

Ihr Fehlermeldungs-Steuerprogramm kann die Logik für die Verarbeitung der kompletten Fehlerprotokollierung enthalten.

- Neuuzuordnung des Schweregrads von Fehlermeldungen je nach den Anforderungen einer Anwendung.

Ihr Fehlermeldungs-Steuerprogramm kann die Logik für die Erkennung von spezifischen Fehlermeldungen und das Herunter- oder Heraufstufen ihres Schweregrads nach Maßgabe der Anwendungsspezifikation und nicht gemäß der Einstufung des Schweregrads durch den Server enthalten. Beispiel: Der Schweregrad einer Fehlermeldung über das Nichtvorhandensein einer Zeile kann während eines Aufräumvorgangs, in dessen Verlauf alte Teile gelöscht werden, herabgestuft werden. Unter anderen Bedingungen wird er möglicherweise heraufgestuft.

Hinweis Fehlermeldungs-Steuerprogramme, die die **SybMessageHandler**-Schnittstelle implementieren, erhalten nur vom Server generierte Meldungen. Sie verarbeiten keine Meldungen, die von jConnect generiert wurden.

Wenn jConnect eine Fehlermeldung erhält, wird geprüft, ob eine **SybMessageHandler**-Klasse für die Verarbeitung der Meldung registriert wurde. Wenn dies so ist, wird die **messageHandler()**-Methode aufgerufen. Die **messageHandler()**-Methode akzeptiert eine SQL-Ausnahmebedingung als ihr Argument, und jConnect verarbeitet die Meldung basierend auf dem Wert, der von der **messageHandler()**-Methode zurückgegeben wird. Das Fehlermeldungs-Steuerprogramm hat folgende Möglichkeiten:

- Es kann die SQL-Ausnahmebedingung in ihrer aktuellen Form zurückgeben.
- Es kann Null zurückgeben. jConnect ignoriert in diesem Fall die Meldung.
- Es kann eine SQL-Warnung aus einer SQL-Ausnahmebedingung generieren und diese zurückgeben. Das bedeutet, dass die Warnung der Warnungsmeldungskette hinzugefügt wird.
- Wenn die ursprüngliche Meldung eine SQL-Warnung war, kann die **messageHandler()**-Methode die SQL-Warnung als dringend auflösen und eine SQLAusnahmebedingung erstellen und zurückgeben, die ausgelöst werden soll, wenn die Kontrolle wieder an jConnect übergeben wird.

Fehler-Steuerungsprogramm einrichten

Sie können ein Fehlermeldungs-Steuerprogramm installieren, das die Klasse **SybMessageHandler** implementiert, indem die **setMessageHandler()**-Methode aus **SybDriver**, **SybConnection** oder **SybStatement** aufgerufen wird. Wenn Sie ein Fehlermeldungs-Steuerprogramm aus **SybDriver**, installieren, erben es alle nachfolgenden **SybConnection**-Objekte. Wenn Sie ein Fehlermeldungs-Steuerprogramm aus einem **SybConnection**-Objekt installieren, wird es von allen **SybStatement**-Objekten geerbt, die von dieser **SybConnection**-Klasse erstellt werden.

Diese Hierarchie gilt nur ab dem Zeitpunkt, ab dem das MessageHandler-Objekt installiert ist. Beispiel: Wenn Sie ein **SybConnection**-Objekt namens *MeineVerbindung* erstellen und dann **SybDriver.setMessageHandler()** aufrufen, um ein Fehler-MessageHandler-Objekt zu installieren, kann *MeineVerbindung* dieses Objekt nicht benutzen.

Um das aktuelle Fehler-MessageHandler-Objekt zurückzugeben, benutzen Sie **getMessageHandler()**.

Beispiel für ein Fehler-Steuerungsprogramm

Das folgende Beispiel benutzt jConnect Version 4.1.

```
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_SQLLE = 20001;

    public MyApp()
    {
        try
        {
            Class.forName("com.sybase.jdbc.SybDriver").newInstance();
            Properties props = new Properties();
```

```

        props.put("user", user);
        props.put("password", password);
        conn = (SybConnection)
        DriverManager.getConnection(server, props);
        conn.setMessageHandler(new NoResultSetHandler());
        stmt = (SybStatement) conn.createStatement();
        stmt.executeUpdate("raiserror 20001 'your error'");

    for (SQLWarning sqw = _stmt.getWarnings();
        sqw != null;
        sqw = sqw.getNextWarning());
    {
        if (sqw.getErrorCode() == AVOID_SQLLE);
        {
            System.out.println("Fehler" +sqw.getErrorCode()+
                " in der Warnliste der Anweisung gefunden.");
            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_SQLLE)
        {
            System.out.println("Benutzer " + _user + " herunterstufen " +
                AVOID_SQLLE + " auf eine Warnung");
            sqe = new SQLWarning(sqe.getMessage(),
                sqe.getSQLState(),sqe.getErrorCode());
        }
        return sqe;
    }
}

public static void main(String args[])

```

```
{  
    new MyApp ( ) ;  
}
```

Java-Objekte als Spaltendaten in einer Tabelle speichern

Einige Datenbankprodukte ermöglichen es, Java-Objekte direkt als Spaltendaten in einer Datenbank zu speichern. In solchen Datenbanken werden Java-Klassen als Datentypen behandelt, und eine Spalte kann mit einer Java-Klasse als Datentyp deklariert werden.

jdbcConnect unterstützt das Speichern von Java-Objekten in einer Datenbank durch die Implementierung der Methoden **setObject()**, die in der **PreparedStatement**-Schnittstelle definiert sind und der Methoden **getObject()**, die in den Schnittstellen **CallableStatement** und **ResultSet** definiert sind. Aus diesem Grund kann jdbcConnect mit einer Anwendung benutzt werden, die native JDBC-Klassen und Methoden verwendet, um Java-Objekte als Spaltendaten direkt zu speichern und abzurufen.

Hinweis Um die Methoden **getObject()** und **setObject()** zu verwenden, setzen Sie die jdbcConnect-Version auf VERSION_4 oder höher. Siehe ["Version von jdbcConnect setzen"](#) auf Seite 6.

In den nachfolgenden Abschnitten werden die Anforderungen und Prozeduren beschrieben, die für das Speichern und Abrufen von Objekten in einer Tabelle unter JDBC mit jdbcConnect verwendet werden:

- [Vorbereitungen für das Speichern von Java-Objekten als Spaltendaten](#)
- [Java-Objekte an eine Datenbank senden](#)
- [Java-Objekte von einer Datenbank empfangen](#)

Hinweis Adaptive Server Enterprise Version 12.0 und Adaptive Server Anywhere Version 6.0.x und höher können Java-Objekte mit einigen Einschränkungen in einer Tabelle speichern. Weitere Informationen finden Sie in der Dokumentation *jdbcConnect for JDBC Versionshinweise*.

Vorbedingungen für das Speichern von Java-Objekten als Spaltendaten

Um Java-Objekte zu speichern, die zu einer benutzerdefinierten Java-Klasse in einer Spalte gehören, müssen drei Vorbedingungen gegeben sein:

- Die Klasse muss die Schnittstelle **java.io.Serializable** implementieren. Dies ist erforderlich, da jConnect native Java-Serialisierung und Entserialisierung verwendet, um Objekte an eine Datenbank zu senden und aus der Datenbank wieder zu empfangen.
- Die Klassendefinition muss in der Zieldatenbank installiert sein *oder* Sie müssen den **DynamicClassLoader** (DCL) zum Laden eines Klassenverzeichnisses von einem Adaptive Server Anywhere- oder Adaptive Server Enterprise-Server benutzen, um es zu verwenden, als ob es im lokalen CLASSPATH aufgeführt wäre. Weitere Hinweise finden Sie unter ["Dynamisches Laden von Klassen" auf Seite 81](#).
- Das Clientsystem muss die Klassendefinition in einer *.class*-Datei halten, die über die lokale Umgebungsvariable CLASSPATH verfügbar ist.

Java-Objekte an eine Datenbank senden

Um eine Instanz einer benutzerdefinierten Klasse als Spaltendaten zu senden, müssen Sie eine der folgenden **setObject()**-Methoden verwenden, wie dies in der Schnittstelle **PreparedStatement** angegeben wird:

```
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;
```

Das folgende Beispiel definiert eine Klasse **Address**, zeigt die Definition einer Tabelle *Friends*, die eine Spalte *Address* hat, deren Datentyp die Klasse **Address** ist, und fügt eine Zeile in die Tabelle ein.

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

Erweiterte Funktionen implementieren

```
/* Dieser Code setzt eine Tabelle mit der folgenden Struktur voraus:
** Create table Friends:
** (firstname varchar(30),
** lastname varchar(30),
** address Address,
** phone varchar(15))
**/

// Verbinde zur Datenbank mit der Tabelle 'Friends'
Connection conn =
    DriverManager.getConnection("jdbc:sybase:Tds:localhost:5000",
        "username", "password");

// Erstelle ein Prepared Statement-Objekt mit einer Insert-Anweisung,
// um die Tabelle Friends zu aktualisieren.
PreparedStatement ps = conn.prepareStatement("INSERT INTO
    Friends values (?, ?, ?, ?)");

// Jetzt setze die Werte im Prep-Statement-Objekt 'ps'.
// Setze firstname auf "Joan".
ps.setString(1, "Joan");

// Setze lastname auf "Smith".
ps.setString(2, "Smith");

// "Joan_address" ist als Instanz von Address vorhanden,
// daher benutze setObject(int parameterIndex, Object x)
// um 'address'-Spalte auf "Joan_address" zu setzen.
ps.setObject(3, Joan_address);

// Setze 'phone'-Spalte auf Telnr. von Joan.
ps.setString(4, "123-456-7890");

// Führe INSERT aus.
ps.executeUpdate();
```

Java-Objekte von einer Datenbank empfangen

Eine Client-JDBC-Anwendung kann ein Java-Objekt von einer Datenbank in einer Ergebnismenge oder als Wert eines Ausgabeparameters empfangen, der von einer gespeicherten Prozedur zurückgegeben wird.

- Wenn eine Ergebnismenge ein Java-Objekt als Spaltendaten enthält, müssen Sie eine der folgenden **getObject()**-Methoden in der **ResultSet**-Schnittstelle verwenden, um das Objekt zu erhalten:

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

- Wenn ein Ausgabeparameter einer gespeicherten Prozedur ein Java-Objekt enthält, müssen Sie die folgende **getObject()**-Methode in der **CallableStatement** -Schnittstelle verwenden, um das Objekt zu erhalten:

Object getObject(int parameterIndex) throws SQLException;

Das folgende Beispiel zeigt den Einsatz von **ResultSet.getObject(int parameterIndex)**, um einer Klassenvariablen ein Objekt zuzuweisen, das in einer Ergebnismenge bezogen wurde. Das Beispiel benutzt die Klasse **Address** und die Tabelle *Friends* aus dem vorherigen Abschnitt und zeigt eine einfache Anwendung, die einen Namen und eine Adresse auf einen Briefumschlag druckt.

```
/*
** Diese Anwendung nimmt einen Vor- und Nachnamen, holt
** die Adresse der Person aus der Tabelle 'Friends' in der
** Datenbank und schreibt eine Adresse mit dem Namen und
** der abgerufenen Adresse auf einen Briefumschlag.
*/
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("Usage: Envelope <Vorname>
                               <Nachname>");
            System.exit(1);
        }
        // Erstelle Briefumschlag 4" x 10"
        Envelope e = new Envelope(4, 10);
        try
        {
            // Verbinde zur Datenbank mit der Tabelle 'Friends'
            conn = DriverManager.getConnection(
                "jdbc:sybase:Tds:localhost:5000", "Benutzername",
                "Kennwort");
```

Erweiterte Funktionen implementieren

```
// Hole die Adresse der angegebenen Person
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);
    // Setze die Adresse auf dem Briefumschlag
    e.setAddress(firstName, lastName, a);
}
conn.close();
}
catch (SQLException sqe)
{
    sqe.printStackTrace();
    System.exit(2);
}
// Wenn alles erfolgreich war, drucke den Briefumschlag
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()
{
    // Drucke Name und Adresse auf dem Briefumschlag.
    ...
}
}
```

Ein detaillierteres Beispiel für **HandleObject.java** finden Sie in den Unterverzeichnissen *sample* (jConnect 4.x) und *sample2* (jConnect 5.x) unter Ihrem jConnect-Verzeichnis.

Dynamisches Laden von Klassen

Adaptive Server Anywhere Version 6.0 und Adaptive Server Enterprise Version 12.0 bieten Java-Klassen in SQL (JCS), mit denen Sie Java-Klassen wie folgt definieren können:

- Datentypen von SQL-Spalten
- Datentypen von Transact-SQL-Variablen
- Standardwerte für SQL-Spalten

Früher waren nur Klassen zugänglich, die im CLASSPATH von jConnect aufgeführt waren. Wenn eine jConnect-Anwendung versucht hat, auf eine Instanz einer Klasse zuzugreifen, die sich nicht im lokalen CLASSPATH befand, wurde eine **java.lang.ClassNotFound**-Ausnahmebedingung ausgegeben.

jConnect Version 5.2 implementiert **DynamicClassLoader** (DCL), um eine Klasse direkt von einem Adaptive Server Anywhere- oder Adaptive Server Enterprise-Server zu laden und sie zu verwenden, als ob sie im lokalen CLASSPATH aufgeführt wäre.

Alle in der Superklasse vorhandenen Sicherheitskomponenten werden geerbt. Das in Java 2 implementierte Loader Delegation-Modell wird befolgt - zuerst versucht jConnect, eine angeforderte Klasse vom CLASSPATH zu laden. Wenn dies fehlschlägt, versucht es jConnect mit dem **DynamicClassLoader**.

Detailliertere Informationen über JCS und Adaptive Server finden Sie in der Dokumentation *Adaptive Server Enterprise Version 12.0 Feature Overview*.

DynamicClassLoader verwenden

Um DCL-Funktionalität zu verwenden, führen Sie Folgendes aus:

- 1 Erstellen Sie einen Class Loader und konfigurieren Sie ihn. Der Code Ihrer jConnect-Anwendung sollte in etwa folgendermaßen aussehen:

```
Properties props = new Properties();

// URL des Servers, auf dem sich die Klassen befinden.
String classesUrl = "jdbc:sybase:Tds:myase:1200";

// Verbindungseigenschaften für die Verbindung mit dem obigen Server.
props.put("user", "grinch");
props.put("password", "meanone");
...

// Bei SybDriver einen neuen Class Loader anfordern.
```

Erweiterte Funktionen implementieren

```
DynamicClassLoader loader = driver.getClassLoader(classesUrl, props);
```

- 2 Machen Sie den neuen Class Loader mit der Verbindungseigenschaft `CLASS_LOADER` für die Anweisung verfügbar, die die Abfrage ausführt. Sobald Sie den Class Loader erstellt haben, übergeben Sie ihn an darauffolgende Verbindungen, wie unten gezeigt. (Das Beispiel in Schritt 1 wird fortgeführt.)

```
// Stauen Sie den Class Loader, damit andere Verbindung(en)
// ihn kennen.
props.put("CLASS_LOADER", loader);

// Zusätzliche Verbindungseigenschaften
props.put("user", "joeuser");
props.put("password", "joespassword");

// URL des Servers, mit dem wir uns verbinden wollen.
String url = "jdbc:sybase:Tds:jdbc.sybase.com:4446";

// Verbindung aufbauen und weiter.
Connection conn = DriverManager.getConnection(url, props);
```

Die folgende Java-Klassendefinition wird vorausgesetzt:

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

Die folgende SQL-Tabellendefinition wird vorausgesetzt:

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

- 3 Verwenden Sie folgenden clientseitigen Code, wobei die Klasse **Addr** im `CLASSPATH` der Client-Anwendung fehlt:

```
Statement stmt = conn.createStatement();
// Einige Zeilen aus der Tabelle holen, die eine Java-Klasse
// als Feld hat.
ResultSet rs = stmt.executeQuery(
    "select * from employee where empid = '19'");
if (rs.next() {
    // Obwohl die Klasse nicht in unserem Klassenpfad ist,
    // können wir auf ihre Instanz zugreifen.
    Object obj = rs.getObject("address");
    // Die Klasse wurde vom Server geladen.
    // Wir werden sehen.
    Class c = obj.getClass();
    // Hier einige Java Reflection
    // um auf die Felder von obj zuzugreifen.
```

```
...
}
```

Die Verbindungseigenschaft `CLASS_LOADER` bietet eine praktische Möglichkeit, um einen Class Loader mit mehreren Verbindungen gemeinsam zu benutzen.

Stellen Sie sicher, dass keine Klassenkonflikte dadurch entstehen, dass ein Class Loader von mehreren Verbindungen gemeinsam verwendet wird. Wenn z.B. zwei verschiedene inkompatible Instanzen der Klasse **org.foo.Bar** in zwei verschiedenen Datenbanken bestehen, können Probleme entstehen, wenn Sie denselben Loader zum Zugriff auf beide Klassen verwenden. Die erste Klasse wird geladen, wenn eine Ergebnismenge von der ersten Verbindung untersucht wird. Wenn dann eine Ergebnismenge von der zweiten Verbindung untersucht werden soll, ist die Klasse bereits geladen. Die zweite Klasse wird nie geladen, und `jdbc` hat keine direkte Möglichkeit, diese Situation zu erkennen.

Java hat jedoch einen eingebauten Mechanismus, um sicherzustellen, dass die Version einer Klasse mit der Versionsinformation in einem deserialisierten Objekt übereinstimmt. Die obige Situation wird zuletzt von Java entdeckt und berichtet.

Klassen und Ihre Instanzen müssen sich nicht in derselben Datenbank oder auf demselben Server befinden, es gibt aber keinen Grund, warum der Loader und darauffolgende Verbindungen sich nicht auf dieselbe Datenbank oder denselben Server beziehen könnten.

Deserialisierung

Das folgende Beispiel veranschaulicht, wie ein Objekt aus einer lokalen Datei deserialisiert wird. Das serialisierte Objekt ist eine Instanz einer Klasse, die sich auf einem Server befindet und nicht im `CLASSPATH` aufgeführt ist.

SybResultSet.getObject() verwendet **DynamicObjectInputStream**, eine Unterklasse von **ObjectInputStream**, die eine Klassendefinition vom **DynamicClassLoader** lädt, anstatt vom Standard-System ("boot") Class Loader.

```
// Einen Stream auf die Datei erzeugen, die
// das serialisierte Objekt enthält.
FileInputStream fileStream = new FileInputStream("serFile");
// Einen "deserializer" darauf erzeugen. Beachten Sie, dass dies, abgesehen vom
// zusätzlichen Parameter, dasselbe ist wie
// ObjectInputStreamDynamicObjectInputStream
stream = new DynamicObjectInputStream(fileStream, loader);
// Sobald das Objekt deserialisiert ist, wird seine Klasse
```

Erweiterte Funktionen implementieren

```
// über den Loader von unserem Server zurückgegeben.  
Object obj = stream.readObject();stream.close();
```

JARS vorladen

jConnect Version 5.2 beinhaltet eine neue Verbindungseigenschaft namens `PRELOAD_JARS`. Die JAR-Dateien werden vollständig geladen, wenn sie als Komma-getrennte Liste mit JAR-Dateinamen definiert werden. In diesem Zusammenhang bezieht sich "JAR" auf den "retained JARname", der vom Server verwendet wird. Dies ist der im Java-Installationsprogramm angegebene JAR-Name. Beispiel:

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

Wenn Sie `PRELOAD_JARS` gesetzt haben, werden die JARS dem Class Loader zugeordnet, so dass sie nicht bei jeder Verbindung vorgeladen werden müssen. Sie müssen `PRELOAD_JARS` nur für eine Verbindung angeben. Darauf folgende Versuche, dieselben JARS vorzuladen, kann Performance-Probleme verursachen, da die JAR-Daten unnötigerweise vom Server geholt werden.

Hinweis Adaptive Server Anywhere 6.x und höher kann eine JAR-Datei nicht als eine Einheit zurückgeben. Deshalb holt jConnect eine Klasse nach der anderen. Adaptive Server 12.x und höher jedoch kann die gesamte JAR-Datei holen und lädt alle Klassen, die sie enthält.

Erweiterte Funktionen

DynamicClassLoader beinhaltet verschiedene öffentliche Methoden. Weitere Hinweise finden Sie in der Dokumentation "Javadocs":

JDBC_HOME/docs/en/javadocs

Die zusätzlichen Funktionen umfassen die Möglichkeit, die Datenbankverbindung eines Class Loaders "offen" zu halten, wenn eine Reihe von zu ladenden Klassen erwartet wird, und explizit eine einzelne Klasse nach dem Namen zu laden.

Öffentliche Methoden, die von **java.lang.ClassLoader** abgeleitet sind, können ebenfalls verwendet werden. Methoden in **java.lang.Class**, die Klassen laden, stehen ebenfalls zur Verfügung. Verwenden Sie jedoch diese Methoden mit Vorsicht, da einige davon entscheiden, welcher Class Loader verwendet werden soll. Verwenden Sie vor allem die 3-Argument-Version von **Class.forName()**, andernfalls wird der System ("boot") Class Loader verwendet. Siehe ["Umgang mit Fehlermeldungen" auf Seite 71](#).

Unterstützung für die Erweiterungen des JDBC 2.0-Optionspakets

Das Optionspaket *JDBC 2.0* (früher *JDBC 2.0 Standarderweiterungs-API*) definiert einige neue Funktionen, die von JDBC 2.0-Treibern implementiert werden können. jConnect Version 5.2 hat die folgenden Funktionserweiterungen im Optionspaket implementiert:

- [JNDI für Datenbankbenennung](#)
(Funktioniert mit jeder Sybase-DBMS, die von jConnect unterstützt wird.)
- [Verbindungspools](#)
(Funktioniert mit jeder Sybase-DBMS, die von jConnect unterstützt wird.)
- [Unterstützung für die Verwaltung verteilter Transaktionen](#)
(Funktioniert nur mit Adaptive Server Enterprise Version 12.0 oder Version 11.x mit einem XA-Server™)

Die oben genannten Funktionen erfordern Klassen und/oder Schnittstellen, die in standardmäßigen Java 2-Distributionen nicht vorhanden sind. Sie müssen **javax.sql.*** und **javax.naming.*** herunterladen, um JNDI für Datenbankbenennung und Verbindungspools implementieren zu können. Sie müssen ebenfalls **javax.transaction.xa.*** herunterladen, um Unterstützung für die Verwaltung verteilter Transaktionen implementieren zu können.

Hinweis Sybase empfiehlt, JNDI 1.2 zu verwenden, das zu Java 1.1.6 und höher kompatibel ist.

JNDI für Datenbankbenennung

Referenz

Das *JDBC 2.0 Optionspaket* (früher *JDBC 2.0 Standarderweiterungs-API*) Kapitel 5 "JNDI und die JDBC-API".

Verwandte Schnittstellen

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

Diese Funktion bietet JDBC-Clients eine Alternative zum standardmäßigen Vorgehen, um Datenbankverbindungen aufzubauen. Anstatt **Class.forName** ("**com.sybase.jdbc2.jdbc.SybDriver**") aufzurufen und dann eine JDBC-URL an die Methode **getConnection()** des **DriverManager** zu übergeben, können Clients auf einen JNDI-Namensserver mit Hilfe eines logischen Namens zugreifen, um ein **javax.sql.DataSource**-Objekt zu erhalten. Dieses Objekt ist für das Laden des Treibers und das Einrichten der Verbindung mit der repräsentierten physikalischen Datenbank verantwortlich. Der Client-Code ist einfacher und wieder verwendbar, da die Hersteller-spezifischen Informationen im Objekt **DataSource** enthalten sind.

Die Sybase-Implementierung des Objekts **DataSource** ist **com.sybase.jdbcx.SybDataSource** (Details finden Sie in Javadocs). Diese Implementierung unterstützt die folgenden Standardeigenschaften und verwendet das Planungsmuster für JavaBean-Komponenten.

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

roleName wird nicht unterstützt.

jConnect bietet eine Implementierung der **javax.naming.spi.ObjectFactory**-Schnittstelle, so dass das Objekt **DataSource** von den Attributen eines Namensserver-Eintrags konstruiert werden kann. Wenn eine **javax.naming.Reference** oder ein **javax.naming.Name** und ein **javax.naming.DirContext** vorhanden sind, kann diese Factory **com.sybase.jdbcx.SybDataSource**-Objekte konstruieren. Um diese Factory zu verwenden, setzen Sie die Systemeigenschaft **java.naming.object.factory** so, dass sie **com.sybase.jdbc2.SybObjectFactory** mit einschließt.

Verwendung

Sie können **DataSource** auf unterschiedliche Arten in verschiedenen Anwendungen verwenden. Alle Optionen sind unterhalb mit einigen Code-Beispielen beschrieben, um Sie durch den Prozess zu führen. Weitere Informationen finden Sie im *JDBC 2.0 Optionspaket* (früher *JDBC 2.0 Standarderweiterungs-API*) und der JNDI-Dokumentation auf der Website von Sun.

1a. Konfiguration durch den Administrator: LDAP

jConnect unterstützt LDAP-Verbindungen bereits seit Version 4.0. Deshalb wird als Vorgehensweise, die keine eigene Software erfordert, empfohlen, **DataSources** als LDAP-Einträge im LDIF-Format zu konfigurieren. Beispiel:

```
dn:servername:myASE, o=MeineFirma, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# MeinComputer 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
```

1b. Zugriff durch den Client

Dies ist eine typische JDBC-Client-Anwendung. Der einzige Unterschied besteht darin, dass Sie auf den Namensserver zugreifen, um eine Referenz auf ein **DataSource**-Objekt zu erhalten, anstatt auf den **DriverManager** zuzugreifen und eine JDBC-URL anzugeben. Sobald Sie die Verbindung bekommen haben, ist der Client-Code mit jedem anderen JDBC-Client-Code identisch. Der Code ist sehr verallgemeinert und referenziert Sybase nur beim Festlegen der Eigenschaft für die Objekt-Factory, die als Teil der Umgebung gesetzt werden kann.

Beispiel: Wenn zwei Threads auf derselben Anweisungs-Instanz arbeiten, wobei der eine Thread eine Abfrage sendet und der andere Thread Warnungen verarbeitet, müssen Sie die Aufrufe der Methoden für die Anweisung synchronisieren, sonst entstehen Konflikte. Die jConnect-Installation enthält das Beispielprogramm *sample2/SimpleDataSource.java*, das veranschaulicht, wie **DataSource** verwendet wird. Dieses Beispiel ist nur als Referenz vorgesehen. Das bedeutet, dass Sie das Beispiel nur dann ausführen können, wenn Sie Ihre Umgebung konfigurieren und das Beispiel entsprechend anpassen. *SimpleDataSource.java* enthält den folgenden entscheidenden Code:

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

// notwendige JNDI-Eigenschaften für Ihre Umgebung festlegen (dieselben wie
// oben)
Properties jndiProps = new Properties();

// von JNDI zum Erzeugen von SybDataSource verwendet
jndiProps.put(Context.OBJECT_FACTORIES,
```

Erweiterte Funktionen implementieren

```
"com.sybase.jdbc2.jdbc.SybObjectFactory");

// Der Namensserver, mit dem JNDI kommunizieren soll
jndiProps.put(Context.PROVIDER_URL,
    "ldap://some_ldap_server:238/o=MyCompany,c=Us");

// von JNDI zum Einrichten des Benennungsinhalts verwendet
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

// Eine Verbindung mit Ihrem Namensserver aufbauen
Context ctx = new InitialContext(jndiProps);
DataSource ds = (DataSource) ctx.lookup("servername=myASE");

// Erhält eine Verbindung zum Server, wie zuvor konfiguriert,
// in diesem Fall Standard-Benutzername und -Kennwort
Connection conn = ds.getConnection();

// Standard JDBC-Methoden
...
```

Beachten Sie, dass es nicht erforderlich ist, die **Properties** explizit an den **InitialContext**-Konstruktor zu übergeben, wenn die Eigenschaften bereits innerhalb der Virtual Machine definiert sind, d.h. beim Aufruf von Java folgendermaßen übergeben wurden:

```
java -Djava.naming.object.factory=com.sybase.jdbc2.jdbc.SybObjectFactory
```

Sie können auch als Teil der Browser-Eigenschaften festgelegt worden sein.

Weitere Informationen über das Setzen der Umgebungseigenschaften finden Sie in Ihrer Java VM-Dokumentation.

2a. Konfiguration durch den Administrator: Benutzerdefiniert

Dieser Abschnitt wird normalerweise von der Person erledigt, die für die Gesellschaft Datenbank-Systemadministrationen oder Anwendungsintegrationen ausführt. Ziel ist, eine Datenquelle zu definieren und sie dann für den Namensserver unter einem logischen Namen bereit zu stellen. Wenn der Server neu konfiguriert werden muss (z.B. auf einen anderen Computer, Port u.s.w. übertragen werden muss), dann führt der Administrator dieses unterhalb erwähnte Konfigurations-Dienstprogramm aus und weist den logischen Namen der neuen Datenquellenkonfiguration erneut zu. Der Client-Code ändert sich deshalb nicht, da er nur den logischen Namen kennt.

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

// Eine SybDataSource erstellen und konfigurieren
```

```
SybDataSource ds = new com.sybase.jdbc2.jdbc.SybDataSource();
ds.setUser("my_username");
ds.setPassword("my_password");
ds.setDatabaseName("my_favorite_db");
ds.setServerName("db_machine");
ds.setPortNumber(4000);
ds.setDescription("Diese Datenquelle repräsentiert den Adaptive Server
    Enterprise-Server, der auf db_machine an Port 2638 läuft. Standard-
    Benutzername und -Kennwort wurden entsprechend auf 'me' und 'mine'
    festgelegt.
    Wenn die Verbindung zustande gekommen ist, kann der Benutzer auf die
    Datenbank 'my_favorite_db' auf
    diesem Server zugreifen.");
Properties props = new Properties();
props.put("REPEAT_READ", "false");
props.put("REQUEST_HA_SESSION", "true");
ds.setConnectionProperties(props);
// Das DataSource-Objekt speichern. Normalerweise geschieht dies durch
// festlegen von JNDI-Eigenschaften, die auf den Typ des
// verwendeten JNDI-Dienstanbieters abgestimmt sind.
// Nun wird der Kontext initialisiert und das Objekt gebunden.
Context ctx = new InitialContext();
ctx.bind("jdbc/myASE", ds);
```

Sobald Ihre **DataSource** eingerichtet ist, entscheiden Sie, wo und wie Sie die Informationen speichern wollen. Zu Ihrer Unterstützung ist **SybDataSource** sowohl **java.io.Serializable** als auch **javax.naming.Referenceable**, es liegt jedoch immer noch in der Entscheidung des Administrators, wie die Daten gespeichert werden, abhängig davon, welchen Dienstanbieter Sie für JNDI verwenden.

2b. Zugriff durch den Client

Der Client erhält das **DataSource**-Objekt, indem er seine JNDI-Eigenschaften genauso festlegt, wie das **DataSource**-Objekt bereit gestellt wurde. Der Client muss eine Objekt-Factory verfügbar haben, die das Objekt in ein Java-Objekt umsetzen kann, wenn es gespeichert wird (z.B. serialisiert).

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

Verbindungspools

Referenz

Das *JDBC 2.0 Optionspaket* (früher die *JDBC 2.0 Standarderweiterungs-API*) Kapitel 6 "Verbindungspools."

Verwandte Schnittstellen

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

Überblick

Traditionelle Datenbankanwendungen erstellen eine Verbindung zu einer Datenbank, die Sie für jede Sitzung einer Anwendung verwenden. Eine Web-basierte Datenbankanwendung jedoch muss in der Lage sein, eine Verbindung mehrmals während der laufenden Anwendung zu schließen und eine neue zu öffnen. Eine wirkungsvolle Methode, Web-basierte Datenbankverbindungen zu handhaben, sind Verbindungspools, die offene Datenbankverbindungen aufrechterhalten und eine gemeinsame Verwendung von Verbindungen über verschiedene Benutzeranforderungen hinweg verwalten. So wird die Performance beibehalten und die Anzahl der inaktiven Verbindungen reduziert. Bei jeder Verbindungsanforderung ermittelt der Verbindungspool zuerst, ob es eine inaktive Verbindung im Pool gibt. Wenn eine besteht, gibt der Verbindungspool diese Verbindung zurück, anstatt eine neue Verbindung zur Datenbank aufzubauen.

Die Fähigkeiten des Verbindungspools werden von **ConnectionPoolDataSource** bereit gestellt. Wenn Sie diese Schnittstelle verwenden, können Sie Verbindungspools nutzen. Wenn Sie die **DataSource**-Schnittstelle verwenden, können Sie keine Verbindungspools nutzen.

Wenn Sie **ConnectionPoolDataSource** verwenden, überwachen Pool-Implementierungen die **PooledConnection**. Die Implementierung wird benachrichtigt, wenn ein Benutzer die Verbindung schließt oder wenn auf der Benutzerverbindung ein Fehler auftritt, der die Verbindung zerstört. An dieser Stelle entscheidet die Pool-Implementierung, was mit der **PooledConnection** geschehen soll.

Ohne Verbindungspools führt eine Transaktion Folgendes aus:

- 1 Sie erstellt eine Verbindung mit der Datenbank.
- 2 Sie sendet die Abfrage an die Datenbank.
- 3 Sie bekommt die Ergebnismenge zurück.
- 4 Sie zeigt die Ergebnismenge an.
- 5 Sie beendet die Verbindung.

Mit Verbindungspools sieht die Reihenfolge folgendermaßen aus:

- 1 Sie erkennt, wenn eine unbenutzte Verbindung im "Pool" der Verbindungen besteht.
- 2 Wenn eine besteht, verwendet sie sie. Andernfalls erstellt sie eine neue Verbindung.
- 3 Sie sendet die Abfrage an die Datenbank.
- 4 Sie bekommt die Ergebnismenge zurück.
- 5 Sie zeigt die Ergebnismenge an.
- 6 Sie gibt die Verbindung an den "Pool" zurück.
(Der Benutzer ruft immer noch "**close()**" auf, aber die Verbindung bleibt offen, und der Pool wird von der Schließen-Anforderung benachrichtigt.)

Es ist billiger, eine Verbindung wieder zu verwenden, als jedes Mal, wenn der Client eine Verbindung zur Datenbank aufbauen muss, eine neue zu erstellen.

Um einen Drittanbieter zu aktivieren, um den Verbindungspool zu implementieren, lässt die `jdbc`-Implementierung die

ConnectionPoolDataSource-Schnittstelle **PooledConnections** erzeugen, ähnlich wie die **DataSource**-Schnittstelle **Connections** erzeugt.

Die Pool-Implementierung erstellt "echte" Datenbankverbindungen, indem sie die **getPooledConnection()**-Methoden von **ConnectionPoolDataSource** verwendet. Dann registriert die Pool-Implementierung sich selbst als Listener der **PooledConnection**.

Derzeit ruft die Pool-Implementierung **getConnection()** für eine verfügbare **PooledConnection** auf, wenn ein Client eine Verbindung anfordert. Wenn der Client die Verbindung nicht mehr benötigt und **close()** aufruft, wird die Pool-Implementierung über die **ConnectionEventListener**-Schnittstelle benachrichtigt, dass die Verbindung frei ist und wiederverwendet werden kann.

Die Pool-Implementierung wird ebenfalls über die **ConnectionEventListener**-Schnittstelle benachrichtigt, wenn der Client die Datenbankverbindung auf irgendeine Weise zerstört, so dass die Pool-Implementierung diese Verbindung vom Pool entfernen kann.

Weitere Informationen finden Sie in Anhang B des *JDBC 2.0 Optionspakets* (früher *JDBC 2.0 Standarderweiterungs-API*).

Konfiguration durch
den Administrator:
LDAP

Dieser Vorgang ist derselbe, wie "[1a. Konfiguration durch den Administrator: LDAP](#)", beschrieben im Kapitel "[JNDI für Datenbankbenennung](#)", außer dass Sie eine zusätzliche Zeile in Ihrem LDIF-Eintrag hinzufügen müssen. Im folgenden Beispiel wird die hinzugefügte Zeile zur besseren Klarheit fett dargestellt.

Erweiterte Funktionen implementieren

```
dn:servername=myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.18:ConnectionPoolDataSource
```

Zugriff von Mittelschicht-Clients Dieses Verfahren initialisiert drei Eigenschaften (INITIAL_CONTEXT_FACTORY, PROVIDER_URL und OBJECT_FACTORIES, wie auf Seite 78 gezeigt) und erhält ein **ConnectionPoolDataSource**-Objekt. Ein vollständigeres Code-Beispiel finden Sie in *sample2/SimpleConnectionPool.java*. Die grundlegenden Unterschiede sehen Sie im folgenden Beispiel:

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

Unterstützung für die Verwaltung verteilter Transaktionen

Diese Funktion bietet eine Standard-Java-API, mit der verteilte Transaktionen entweder mit Adaptive Server Enterprise Version 12.x oder Version 11.x mit XA-Server ausgeführt werden können.

Hinweis Diese Funktion wurde entworfen, um in einer großen Mehrschichten-Umgebung verwendet zu werden.

Referenz

Siehe Kapitel 7, "Distributed Transactions" im *JDBC 2.0 Optionspaket* (früher *JDBC 2.0 Standarderweiterungs-API*).

Verwandte Schnittstellen

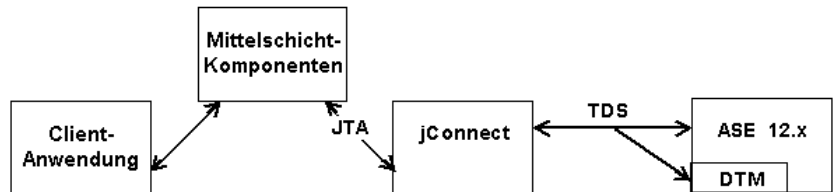
- `javax.sql.XADataSource`
- `javax.sql.XAConnection`
- `javax.transaction.xa.XAResource`

Hintergrund und erforderliche Systemausstattung

Für Adaptive Server Enterprise 12.0

- Da jConnect direkt mit dem Ressourcen-Manager innerhalb von Sybase Adaptive Server Enterprise Version 12.0 kommuniziert, muss die Installation die Verwaltung verteilter Transaktionen unterstützen.
- Jedem Benutzer, der an einer verteilten Transaktion teilnehmen will, muss die "dtm_tm_role" gewährt werden, da sonst die Transaktionen fehlschlagen.
- Um verteilte Transaktionen verwenden zu können, müssen Sie die gespeicherten Prozeduren im Verzeichnis /sp installieren. Lesen Sie hierzu "Gespeicherte Prozeduren installieren" im Kapitel 1 Ihrer Dokumentation *jConnect for JDBC Installationshandbuch*.

**Abbildung 2-2: Verwaltung verteilter Transaktionen
Unterstützung mit Version 12.x**

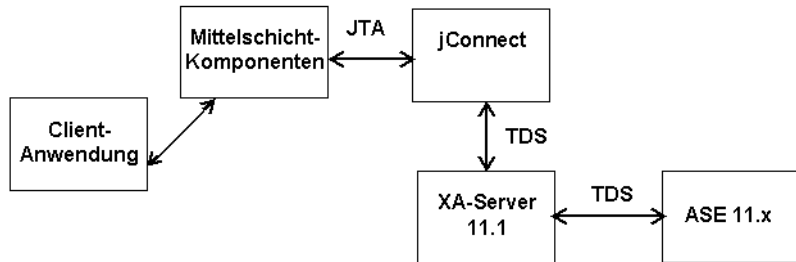


Für Adaptive Server Enterprise 11.x

jConnect bietet ebenfalls eine Standard-Java-API, um verteilte Transaktionen mit Adaptive Server Enterprise Version 11.x als Datenbankserver ausführen zu können.

- Diese Implementierung funktioniert nur mit Sybase Adaptive Server Enterprise Version 11.x und XA-Server 11.1.

Abbildung 2-3: Unterstützung für die Verwaltung verteilter Transaktionen mit Version 11.x



- Die gewählte Anmeldung kann keine Standard-Datenbankanmeldung mit *master*, *model* oder *sybsystemdb* haben. Der Grund dafür ist, dass der XA-Server sich nur dann verbindet, wenn die Arbeit des Benutzers einer verteilten Transaktion zugeordnet ist und verteilte Transaktionen auf diesen Datenbanken nicht zulässig sind.
- Auf Metadaten kann nicht zugegriffen werden. Da dies den Client einschränkt, wird dieser Teil der API nicht innerhalb der Grenzen verteilter Transaktionen verwendet.

Adaptive Server Enterprise 12.x verwenden

Konfiguration durch
den Administrator:
LDAP

Dieser Vorgang ist derselbe wie "[1a. Konfiguration durch den Administrator: LDAP](#)", beschrieben in "[JNDI für Datenbankbenennung](#)" auf Seite 85, außer dass Sie eine zusätzliche Zeile in Ihrem LDIF-Eintrag hinzufügen müssen. Im folgenden Beispiel wird die hinzugefügte Zeile zur besseren Klarheit fett dargestellt.

```
dn:servername:myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.18:XADataSource
```

Zugriff von
Mittelschicht-Clients

Dieses Verfahren initialisiert drei Eigenschaften (INITIAL_CONTEXT_FACTORY, PROVIDER_URL und OBJECT_FACTORIES) und erhält ein **XADataSource**-Objekt. Beispiel:

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

Sie können auch die Standardeinstellungen für Benutzername und Kennwort überschreiben:

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

Adaptive Server Enterprise 11.x verwenden

Konfiguration durch
den Administrator:
LDAP

Dieser Vorgang ist derselbe wie ["1a. Konfiguration durch den Administrator: LDAP"](#), beschrieben in ["JNDI für Datenbankbenennung"](#) auf Seite 85, außer dass Sie eine zusätzliche Zeile in Ihrem LDIF-Eintrag hinzufügen müssen.

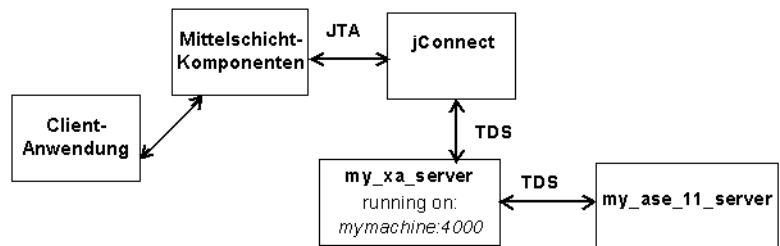
Im folgenden Beispiel wird die hinzugefügte Zeile zur besseren Klarheit fett dargestellt.

```
dn:servername=myASE, o=meineFirma, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
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
```

Hierbei gilt: **...4.2.17:1** zeigt an, dass sich jConnect mit einem XA-Server verbindet, und **userconnection** entspricht dem verwendeten Logischen Ressourcen-Manager (LRM). Der XA-Server hat eine Datei *xa_config*, die folgende Einträge enthält:

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

Abbildung 2-4: Beispielkonfiguration für die Unterstützung der Verwaltung verteilter Transaktionen



Details, wie eine *xa_config*-Datei geschrieben wird, finden Sie in Ihrer XA-Server-Dokumentation.

Erweiterte Funktionen implementieren

Zugriff von
Mittelschicht-Clients

Dieses Verfahren initialisiert drei Eigenschaften (INITIAL_CONTEXT_FACTORY, PROVIDER_URL und OBJECT_FACTORIES) und erhält ein **XADatasource**-Objekt. Beispiel:

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

Mit Adaptive Server Enterprise 11.x können Sie den Standard-Benutzernamen und das Standard-Kennwort *nicht* überschreiben. D.h., Sie können Folgendes nicht aufrufen:

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

Der Grund dafür ist, dass *lrm* einem bestimmten Benutzernamen und Kennwort zugeordnet ist.

Umgang mit Beschränkungen, Einschränkungen und Abweichungen von den JDBC-Standards

In diesem Abschnitt werden Beschränkungen und Einschränkungen besprochen, die für jConnect gelten, sowie Abweichungen der jConnect-Implementierung von den Standards JDBC 1.x und 2.0. Folgende Themen werden behandelt:

- Anpassungen für Multithreading
- `ResultSet.setCursorName()` verwenden
- `setLong()` mit großen Parameterwerten verwenden
- COMPUTE-Anweisungen verwenden
- gespeicherte Prozeduren ausführen

Anpassungen für Multithreading

Wenn mehrere Threads gleichzeitig Methoden für dieselbe **Statement**-Instanz **CallableStatement** oder **PreparedStatement** aufrufen, was wir nicht empfehlen, müssen Sie die Aufrufe der Methoden für das **Statement** manuell synchronisieren. jConnect tut dies nicht automatisch.

Beispiel: Wenn zwei Threads auf derselben **Anweisungs**-Instanz arbeiten, wobei der eine Thread eine Abfrage sendet und der andere Thread Warnungen verarbeitet, müssen Sie die Aufrufe der Methoden für die **Anweisung** synchronisieren, da sonst Konflikte entstehen.

ResultSet.setCursorName() verwenden

Einige JDBC-Treiber generieren einen Cursornamen für jede SQL-Abfrage, so dass eine Zeichenfolge immer zurückgegeben werden kann. jConnect gibt aber keinen Namen zurück, wenn **ResultSet.setCursorName()** aufgerufen wird, wenn Sie nicht entweder

- **setFetchSize()** oder **setCursorName()** für die entsprechende **Anweisung** aufgerufen haben oder
- die Verbindungseigenschaft `SELECT_OPENS_CURSOR` auf "true" gesetzt haben und Ihre Abfrage die Form `SELECT... FOR UPDATE` hatte. Zum Beispiel:

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

Wenn Sie nicht **setFetchSize()** oder **setCursorName()** für die entsprechende Anweisung aufgerufen oder die Verbindungseigenschaft **SELECT_OPENES_CURSOR** auf "true" gesetzt haben, wird ein Nullwert zurückgegeben.

Gemäß JDBC 2.0 API (Kapitel 11, "Clarifications"), brauchen alle anderen SQL-Anweisungen keinen Cursor zu öffnen und keinen Namen zurückzugeben.

Weitere Hinweise zum Einsatz eines Cursors in jConnect finden Sie unter ["Cursor mit Ergebnismengen verwenden" auf Seite 48](#).

setLong() mit großen Parameterwerten verwenden

Die Implementierungen der Methode **PreparedStatement.setLong()** setzen einen Parameterwert auf einen BIGINT-Datentyp. Die meisten Adaptiv Server-Datenbanken haben keinen 8-Byte-BIGINT-Datentyp. Wenn ein Parameterwert mehr als 4 Byte eines BIGINT-Datentyps benötigt, kann die Verwendung von **setLong()** einen Überlauffehler bewirken.

COMPUTE-Anweisungen verwenden

jConnect unterstützt keine berechneten Zeilen. Die Ergebnisse werden automatisch annulliert, wenn eine Abfrage eine berechnete Zeile enthält. Beispielsweise wird folgende Anweisung zurückgewiesen:

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

Um dieses Problem zu vermeiden, schreiben Sie die Anweisung statt dessen wie folgt:

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

gespeicherte Prozeduren ausführen

- Wenn Sie eine gespeicherte Prozedur in einem **CallableStatement**-Objekt ausführen, das Parameterwerte als Fragezeichen darstellt, erhalten Sie eine bessere Performance, als wenn Sie sowohl Fragezeichen als auch Literalwerte für Parameter verwenden. Außerdem gilt: Wenn Sie Literalwerte und Fragezeichen vermischen, können Sie mit einer gespeicherten Prozedur keine Ausgabeparameter verwenden.

Das folgende Beispiel erstellt *sp_stmt* als ein **CallableStatement**-Objekt für die Ausführung der gespeicherten Prozedur **MyProc**:

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

Die beiden Parameter in **MyProc** werden als Fragezeichen dargestellt. Sie können eine oder beide als Ausgabeparameter registrieren, indem Sie die **registerOutParameter()**-Methoden in der Schnittstelle **CallableStatement** verwenden.

Im folgenden Beispiel ist *sp_stmt2* ein **CallableStatement**-Objekt für die Ausführung der gespeicherten Prozedur **MyProc2**.

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

In *sp_stmt2* ist ein Parameterwert als Literalwert angegeben, der andere hingegen als Fragezeichen. Sie können beide Parameter nicht als Ausgabeparameter registrieren.

- Um gespeicherte Prozeduren mit RPC-Befehlen mit Hilfe von Namensbildung für Parameter auszuführen, verwenden Sie eine der folgenden Prozeduren:
 - Verwenden Sie Sprach-Befehle und übergeben Sie ihnen Eingabeparameter direkt von Java-Variablen mit Hilfe der Klasse **PreparedStatement**. Dies wird in folgendem Programmcodausschnitt gezeigt:

```
// Anweisung vorbereiten
System.out.println("Anweisung wird vorbereitet...");
String stmtString = "exec " + procname + " @p3=?, @p1=?";
PreparedStatement pstmt = con.prepareStatement(stmtString);

// Werte setzen
pstmt.setString(1, "xyz");
pstmt.setInt(2, 123);

// Abfrage senden
```

```
System.out.println("Abfrage wird ausgeführt...");  
ResultSet rs = pstmt.executeQuery();
```

- Verwenden Sie für jConnect Version 5.2 die Schnittstelle **com.sybase.jdbcx.SybCallableStatement**, das dieses Beispiel veranschaulicht.

```
import com.sybase.jdbcx.*;  
  
....  
// Vorbereiten des Aufrufs für die gespeicherte Prozedur, um sie als  
//RPC auszuführen  
String execRPC = "{call " + procName + " (?, ?)}";  
SybCallableStatement scs = (SybCallableStatement)  
con.prepareCall(execRPC);  
  
// Werte setzen und Parameter benennen.  
// ebenfalls (optional) registrieren für alle Ausgabeparameter  
scs.setString(1, "xyz");  
scs.setParameterName(1, "@p3");  
scs.setInt(2, 123);  
scs.setParameterName(2, "@p1");  
  
// RPC ausführen  
// kann auch die Ergebnisse mit getResultSet()  
// und getMoreResults() bearbeiten  
  
// Weitere Informationen über das Bearbeiten der Ergebnisse finden  
//Sie in den Beispielen.  
ResultSet rs = scs.executeQuery();
```


In diesem Kapitel werden Lösungen und Behelfslösungen für Probleme beschrieben, auf die Sie bei der Arbeit mit jConnect möglicherweise stoßen.

Dieses Kapitel behandelt folgende Themen:

Name	Seite
Debugging mit jConnect	102
TDS-Kommunikation aufzeichnen	106
Fehler bei erfolglosen Verbindungen	108
Speichernutzung in jConnect-Anwendungen	110
Fehler bei gespeicherten Prozeduren	111
Fehler bei der Implementierung des benutzerdefinierten Sockets	113

Debugging mit jConnect

jConnect enthält eine **Debug**-Klasse mit einer Serie von Debugging-Funktionen. Die **Debug**-Methoden enthalten eine Reihe von Assert-, Trace- und Timer-Funktionen, mit denen Sie den Umfang des Fehlersuchprozesses und das Ausgabegerät für die Ergebnisse der Fehlersuche festlegen können.

Die jConnect-Installation enthält auch einen kompletten Satz von Klassen mit aktivierten Debug-Eigenschaften. Diese Klassen befinden sich im Unterverzeichnis *devclasses* des jConnect-Installationsverzeichnisses. Für die Fehlersuche müssen Sie Ihre CLASSPATH-Umgebungsvariable auf die Laufzeitklassen für den Debug-Modus (*devclasses* für jConnect 4.x und *devclasses/jconn2d.jar* für jConnect 5.x) umlenken und nicht auf das Standard-jConnect-Verzeichnis *classes*. Eine derartige Referenzierung kann auch durch explizite Bereitstellung eines **-classpath**-Arguments für den **java**-Befehl erfolgen, wenn Sie ein Java-Programm ausführen.

Instanz der Debug-Klasse erhalten

Wenn Sie die Fehlersuchfunktionen von jConnect verwenden wollen, muss Ihre Anwendung die **Debug**-Schnittstelle importieren und eine Instanz der **Debug**-Klasse einrichten, indem die Methode **getDebug()** für die Klasse **SybDriver** aufgerufen wird.

Für 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();
...
```

Für 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();
...
```

Fehlersuchfunktion in Ihrer Anwendung einschalten

Wenn Sie die Methode **debug()** mit dem **Debug**-Objekt benutzen wollen, um die Fehlersuche in Ihrer Anwendung zu aktivieren, fügen Sie folgenden Aufruf hinzu:

```
sybdebug.debug(true, [Klassen], [Ausgabestream]);
```

Der Parameter *Klassen* ist eine Zeichenfolge, die getrennt durch Doppelpunkte die spezifischen Klassen auflistet, für die eine Fehlersuche erfolgen soll. Beispiel:

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

und

```
sybdebug.debug(true, "MyClass:YourClass")
```

Wenn Sie "STATIC" in die Klassen-Zeichenfolge aufnehmen, wird die Fehlersuche für alle statischen Methoden in jConnect zusätzlich zu den bezeichneten Klassen aktiviert. Beispiel:

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

Sie können "ALL" festlegen, um die Fehlersuche für alle Klassen zu aktivieren. Beispiel:

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

Der *Ausgabestream*-Parameter ist optional. Wenn Sie keinen 'Ausgabestream'-Parameter angeben, wird die Ausgabe der Fehlersuchfunktion an das Gerät geleitet, das Sie mit **DriverManager.setLogStream()** angegeben haben.

Fehlersuchfunktion in Ihrer Anwendung ausschalten

Wenn Sie die Fehlersuche ausschalten wollen, fügen Sie folgenden Aufruf ein:

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

CLASSPATH für die Fehlersuche einstellen

Bevor Sie Ihre Anwendung mit aktivierter Fehlersuche starten, definieren Sie die CLASSPATH-Umgebungsvariable um, so dass sie das Unterverzeichnis */devclasses* im jConnect-Installationsverzeichnis anspricht.

Für jConnect 4.x:

- Unter UNIX ersetzen Sie `$JDBC_HOME/classes` durch `$JDBC_HOME/devclasses`.
- Unter Windows ersetzen Sie `%JDBC_HOME%\classes` durch `%JDBC_HOME%\devclasses`.

Für jConnect 5.x:

- Unter UNIX ersetzen Sie `$JDBC_HOME/classes/jconn2.jar` durch `$JDBC_HOME/devclasses/jconn2.jar`.
- Unter Windows ersetzen Sie `%JDBC_HOME%\classes\jconn2.jar` durch `%JDBC_HOME%\devclasses\jconn2.jar`.

Debug-Methoden verwenden

Sie können Aufrufe zu anderen **Debug**-Methoden hinzufügen, um den Fehlersuchprozess anzupassen.

Bei diesen Methoden ist der erste (Objekt)-Parameter in der Regel *this*, um das aufrufende Objekt festzulegen. Wenn eine dieser Methoden statisch ist, verwenden Sie *null* für den Objektparameter.

- **println()**

Benutzen Sie diese Methode, um die Meldung zu definieren, die in das Ausgabe-Log geschrieben werden soll, wenn die Fehlersuchfunktion aktiviert und das Objekt in der Liste der von der Fehlersuche zu erfassenden Klassen enthalten ist. Die Debug-Ausgabe wird in die Ausgabedatei geschrieben, die Sie mit `sybdebug.debug()` festgelegt haben.

Hierbei gilt die folgende Syntax:

```
sybdebug.println(object,message string);
```

Beispiel:

```
sybdebug.println(this,"Query: "+ query);
```

Damit wird eine Meldung der folgenden Art in das Ausgabe-Log geschrieben:

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

- **assert()**

Benutzen Sie diese Methode für die Auswertung einer Bedingung und die Ausgabe eines Laufzeit-Ausnahmefehlers, wenn diese Bedingung nicht eingehalten wird. Sie können auch die Meldung definieren, die in das Ausgabe-Log zu schreiben ist, wenn die Bedingung nicht zutrifft. Hierbei gilt die folgende Syntax:

```
sybdebug.assert(object,boolean condition,message
string);
```

Beispiel:

```
sybdebug.assert(this,amount<=buf.length,amount+"
zu groß!");
```

Damit wird eine Meldung der folgenden Art in das Ausgabe-Log geschrieben, wenn "amount" den Wert von *buf.length* überschreitet:

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

- **startTimer()**
stopTimer()

Benutzen Sie diese Methoden, um einen Timer zu starten und zu stoppen, der die während eines Ereignisses verstrichenen Millisekunden mißt. Die Methode hält einen Timer pro Objekt und einen Timer für alle statischen Methoden. Die Syntax zum Start des Timers lautet wie folgt:

```
sybdebug.startTimer(object);
```

Die Syntax für das Stoppen des Timers lautet wie folgt:

```
sybdebug.stopTimer(object,message string);
```

Beispiel:

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

Damit wird eine Meldung der folgenden Art in das Ausgabe-Log geschrieben:

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

TDS-Kommunikation aufzeichnen

Tabular Data Stream (TDS) ist das Sybase-eigene Protokoll für die Abwicklung der Kommunikation zwischen einer Client-Anwendung und Adaptive Server. jConnect enthält eine Verbindungseigenschaft `PROTOCOL_CAPTURE`, die das Aufzeichnen von unveränderten TDS-Paketen in einer Datei ermöglicht.

Wenn Sie Probleme mit einer Anwendung haben, die Sie weder innerhalb der Anwendung, noch im Server lösen können, verwenden Sie `PROTOCOL_CAPTURE`, um die Kommunikation zwischen dem Client und dem Server in einer Datei aufzuzeichnen. Danach können Sie die Datei, die Binärdaten enthält und nicht direkt interpretiert werden kann, an den Technischen Kundendienst von Sybase zur Analyse senden.

Hinweis Sie können auch das Dienstprogramm **Ribo** verwenden, um den Protokoll-Datenstrom zwischen Client und Server aufzuzeichnen, zu übersetzen und anzuzeigen. Detail-Informationen, wie Sie **Ribo** erhalten und verwenden können, finden Sie auf der Website jConnect-Dienstprogramme:

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

Verbindungseigenschaft `PROTOCOL_CAPTURE`

Die Verbindungseigenschaft `PROTOCOL_CAPTURE` wird benutzt, um eine Datei anzugeben, die die TDS-Pakete enthalten soll, die zwischen einer Anwendung und einer Adaptive Server-Installation ausgetauscht werden. `PROTOCOL_CAPTURE` tritt sofort in Kraft, so dass die während der Einrichtung einer Verbindung ausgetauschten TDS-Pakete sofort in die Datei geschrieben werden. Alle Pakete werden weiter in die Datei geschrieben, bis **`Capture.pause()`** ausgeführt oder die Sitzung beendet wird.

Im folgenden Beispiel wird gezeigt, wie `PROTOCOL_CAPTURE` verwendet wird, um TDS-Daten in die Datei `tds_data` zu senden:

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

Hierbei gilt: *url* ist der Verbindungs-URL und *props* ist ein **Properties**-Objekt, in dem die Verbindungseigenschaften festgelegt werden.

Die Methoden `pause()` und `resume()` in der `Capture`-Klasse

Die Klasse **Capture** ist im Paket **com.sybase.jdbcx** enthalten. Sie enthält zwei Public-Methoden:

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

Capture.pause() stoppt das Aufzeichnen der uninterpretierten TDS-Pakete in einer Datei. **Capture.resume()** startet das Aufzeichnen erneut.

Die TDS-Aufzeichnungsdatei für eine komplette Sitzung kann sehr stark anschwellen. Wenn Sie wissen, wo in einer Anwendung Sie TDS-Daten aufzeichnen wollen und daher die Größe der Datei limitieren können, wählen Sie eine der folgenden Möglichkeiten:

- 1 Sofort nachdem Sie eine Verbindung eingerichtet haben, holen Sie das **Capture**-Objekt für die Verbindung und benutzen die Methode **pause()**, um die TDS-Daten aufzuzeichnen:

```
Capture cap = ((SybConnection)conn).getCapture();  
cap.pause();
```
- 2 Setzen Sie **cap.resume()** kurz vor dem Punkt, an dem Sie mit dem Aufzeichnen der TDS-Daten beginnen wollen.
- 3 Setzen Sie **cap.pause()** kurz nach dem Punkt, an dem das Aufzeichnen der Daten gestoppt werden soll.

Fehler bei erfolglosen Verbindungen

Dieser Abschnitt behandelt Probleme, die auftreten können, wenn Sie versuchen, eine Verbindung einzurichten oder ein Gateway zu starten.

Gateway-Verbindung abgelehnt

```
Gateway-Verbindung abgelehnt
HTTP/1.0 502 Bad Gateway|Restart Connection
```

Diese Fehlermeldung zeigt, dass ein Fehler im *Hostname* oder *Portnummer* aufgetreten ist, die für die Verbindung zu Adaptive Server verwendet wurden. Prüfen Sie den Eintrag [query] in *\$\$SYBASE/interfaces* (UNIX) oder in *%SYBASE%\ini\sql.ini* (Windows).

Wenn das Problem weiterhin besteht, nachdem Sie *Hostname* und *Portnummer* überprüft haben, können Sie weitere Informationen beziehen, indem Sie den HTTP-Server mit aktivierter Systemeigenschaft "verbose" starten.

Unter Windows rufen Sie ein DOS-Fenster auf und geben ein:

```
httpd -Dverbose=1 > Dateiname
```

Unter UNIX geben Sie ein:

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

Dabei gilt: *Dateiname* ist die Ausgabedatei für die Debug-Meldungen.

Wenn Sie die Verbindung nicht über das Cascade-HTTP-Gateway herstellen, unterstützt der Web-Server möglicherweise die Verbindungsmethode nicht. Applets können eine Verbindung nur mit dem Host herstellen, von dem sie heruntergeladen wurden, wenn Sie nicht das Cascade-HTTP-Gateway benutzen, durch das ein Pfad zum Datenbankserver als Proxy eingerichtet wird.

Das Cascade-HTTP-Gateway und Ihr Web-Server müssen auf demselben Host laufen: In diesem Szenario kann sich Ihr Applet mit demselben Rechner/Host über den Port verbinden, der vom Cascade-HTTP-Gateway gesteuert wird, der die Anforderung an die entsprechende Datenbank weitergibt.

Wenn Sie sehen wollen, wie dies programmiert wird, sehen Sie sich den Quellcode von *Isql.java* und *gateway.html* im Unterverzeichnis *sample* (jConnect 4.x) oder *sample2* (jConnect 5.x) unter dem jConnect-Installationsverzeichnis an. Suchen Sie nach "proxy".

Verbindung mit 4.9.2 SQL Server unmöglich

jConnect benutzt TDS 5.0 (Sybase-Transfer-Protokoll). SQL Server 4.9.x benutzt TDS 4.6, das mit TDS 5.0 nicht kompatibel ist.

SQL Server 10.0.2 oder später ist für den Einsatz von jConnect erforderlich.

Speichernutzung in jConnect-Anwendungen

Die folgenden Situationsbeschreibungen und die angegebenen Lösungsmöglichkeiten können sich als nützlich erweisen, wenn Sie in jConnect-Anwendungen übermäßige Speichernutzung feststellen.

- In jConnect-Anwendungen müssen Sie explizit alle **Statement**-Objekte und Unterklassen (z.B. **PreparedStatement**, **CallableStatement**) nach der letzten Verwendung schließen, damit sich die Anweisungen nicht im Speicher ansammeln. Das Schließen des Objekts **ResultSet** ist nicht ausreichend.

Beispiel:

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

Dies würde zu Problemen führen. Benutzen Sie lieber folgende Anweisung:

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

- jConnect benutzt den Tabular Data Stream (TDS), das von Sybase bereitgestellte Protokoll, zur Kommunikation mit Sybase-Datenbankservern. Ab jConnect 5.0 unterstützt TDS keine abrollbaren Cursors. Um abrollbare Cursor zu unterstützen, setzt jConnect die Zeilendaten auf Anforderung auf dem Client bei jedem Aufruf von **ResultSet.next()** in den Cache. Wenn jedoch das Ende der Ergebnismenge erreicht ist, wird die komplette Ergebnismenge im Speicher des Clients gespeichert. Da dies eine Beeinträchtigung der Performance nach sich ziehen kann, empfehlen wir, TYPE_SCROLL_INSENSITIVE Ergebnismengen nur zu verwenden, wenn die Ergebnismenge entsprechend klein ist.

Fehler bei gespeicherten Prozeduren

Dieser Abschnitt behandelt Probleme, die auftreten können, wenn Sie versuchen, jConnect und gespeicherte Prozeduren zu verwenden.

RPC gibt weniger Ausgabeparameter zurück als registriert

SQLState: JZ0SG - Ein Callable Statement gab nicht so viele Ausgabeparameter zurück, wie die Anwendung für sie registriert hat.

Dieser Fehler tritt auf, wenn Sie **CallableStatement.registerOutParam()** für mehr Parameter aufrufen als Sie als "OUTPUT"-Parameter in der gespeicherten Prozedur deklariert haben. Achten Sie darauf, dass alle erforderlichen Parameter als "OUTPUT" deklariert werden. Suchen Sie folgende Programmcodezeile:

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

Hinweis Wenn Sie diesen Fehler erhalten, während Sie Adaptive Server Anywhere (früher SQL Anywhere) verwenden, führen Sie ein Upgrade auf Adaptive Server Anywhere Version 5.5.04 oder höher durch.

Fetch/State-Fehler bei der Rückgabe von Ausgabeparametern durch die gespeicherte Prozedur

Wenn eine Abfrage keine Zeilendaten zurückgibt, muss sie die Methoden **CallableStatement.executeUpdate()** oder **execute()** anstelle der Methode **executeQuery()** verwenden.

jConnect gibt eine SQL-Ausnahmebedingung aus, wenn **executeQuery()** keine Ergebnismengen hat, da dies vom JDBC-Standard gefordert wird.

Gespeicherte Prozedur im unverketteten Transaktionsmodus ausgeführt

Sybase Fehler 7713 - Die gespeicherte Prozedur kann nur im unverketteten Transaktionsmodus ausgeführt werden.

Fehler bei gespeicherten Prozeduren

JDBC versucht, die Verbindung in den Modus **autocommit(true)** zu setzen. Die Anwendung kann die Verbindung auf den verketteten Modus ändern, indem **Connection.setAutoCommit(false)** oder der Sprachbefehl **"set chained on"** verwendet werden. Wenn die gespeicherte Prozedur nicht in einem kompatiblen Modus erstellt wurde, tritt dieser Fehler auf.

Verwenden Sie diese Systemprozedur, um das Problem zu beheben:

```
sp_procxmode procedure_name, "anymode"
```

Fehler bei der Implementierung des benutzerdefinierten Sockets

Es kann sein, dass Sie eine Ausnahmefehlermeldung der folgenden Art erhalten, während Sie versuchen, einen SSL Socket einzurichten und **sun.security.ssl.SSLSocketImpl.setEnabledCipherSuites** aufrufen:

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

In diesem Fall prüfen Sie, ob sich die SSL-Bibliotheken im Systembibliothekspfad befinden.

Performance und Optimierung

In diesem Kapitel wird beschrieben, wie Sie die Optimierung oder Verbesserung der Performance durchführen, wenn Sie mit jConnect arbeiten.

Folgende Themen werden behandelt:

Name	Seite
Performance von jConnect steigern	116
Optimierung der Performance für Prepared Statements in Dynamic SQL	119
Cursor-Performance	126

Performance von jConnect steigern

Es gibt eine Reihe von Möglichkeiten, die Performance einer Anwendung mit jConnect zu verbessern. Nachstehend einige Vorschläge:

- Verwenden Sie die **TextPointer.sendData()**-Methoden zum Senden von Text- und Bilddaten an eine Adaptive Server-Datenbank. Siehe ["Bilddaten senden" auf Seite 63](#).
- Erstellen Sie vorkompilierte **PreparedStatement**-Objekte für dynamische SQL-Anweisungen, die während einer Sitzung mehrmals verwendet werden. Siehe ["Optimierung der Performance für Prepared Statements in Dynamic SQL" auf Seite 119](#).
- Aktualisierungen in Form von Anweisungsfolgen verbessern die Performance durch Entlastung des Netzwerkverkehrs. Insbesondere werden alle Abfragen in einer Gruppe an den Server geschickt, und alle Antworten des Servers kommen ebenfalls blockweise zum Client zurück. Siehe ["Unterstützung von Aktualisierungen in Form von Anweisungsfolgen" auf Seite 60](#).
- Bei Sitzungen, in denen voraussichtlich Bilddaten, große Zeilenvolumina und lange Textdaten verschoben werden, verwenden Sie die Verbindungseigenschaft **PACKETSIZE**, um die maximal mögliche Paketgröße festzulegen.
- Bei TDS-tunneled HTTP setzen Sie die maximale TDS-Paketgröße und konfigurieren Ihren Webserver so, dass das HTTP1.1-Merkmal Keep-Alive unterstützt wird. Setzen Sie auch das Servlet-Argument *SkipDoneProc* auf "true" (siehe vi).
- Benutzen Sie einen Protocol-Cursor, die Standardeinstellung der Verbindungseigenschaft **LANGUAGE_CURSOR**. Weitere Hinweise finden Sie unter ["LANGUAGE_CURSOR, Verbindungseigenschaft" auf Seite 126](#).
- Sie sollten **TYPE_SCROLL_INSENSITIVE** Ergebnismengen nur verwenden, wenn die Ergebnismenge entsprechend klein ist. Weitere Hinweise finden Sie unter ["Unterstützung für SCROLL_INSENSITIVE ResultSets in jConnect" auf Seite 57](#).

Weitere Überlegungen zur Performancesteigerung werden nachstehend beschrieben.

BigDecimal-Skalierungsfaktor

Die JDBC-1.0-Spezifikation benötigt einen Skalierungsfaktor mit **getBigDecimal()**. Wenn danach ein **BigDecimal**-Objekt vom Server zurückgegeben wird, muss es mit dem Original-Skalierungsfaktor, den Sie mit **getBigDecimal()** verwendet haben, reskaliert werden.

Um die Verarbeitungszeit einzusparen, die für das Reskalieren erforderlich ist, verwenden Sie die Methode **getBigDecimal()**, die **jConnect** in der Klasse **SybResultSet** implementiert und keinen *scale* -Wert erfordert:

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

Beispiel:

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

REPEAT_READ, Verbindungseigenschaft

Sie können die Performance beim Abruf einer Ergebnismenge aus der Datenbank verbessern, wenn Sie die Verbindungseigenschaft **REPEAT_READ** auf "false" setzen. Beachten Sie aber folgende Hinweise, wenn **REPEAT_READ** auf "false" gesetzt ist:

- Sie müssen Spaltenwerte in der Reihenfolge des Spaltenindexes einlesen. Dies ist schwierig zu realisieren, wenn Sie auf Spalten nach Namen und nicht nach Spaltennummern zugreifen.
- Sie können einen Spaltenwert in einer Zeile nicht mehr als einmal lesen.

Zeichensatzkonvertierung

Wenn Sie Mehrbyte-Zeichensätze verwenden und die Treiber-Performance verbessern müssen, können Sie die Klasse **SunloConverter** verwenden, die mit den jConnect-Beispielen mitgeliefert wurde. Dieser Konverter basiert auf den **sun.io** Klassen, die von der Java Software Division von Sun Microsystems, Inc. bereitgestellt werden.

Die Klasse **SunloConverter** ist keine reine Java-Implementierung der Konvertierungsfunktion für Zeichensätze und wurde daher in das Standard-jConnect-Produkt nicht aufgenommen. Diese Konverterklasse wurde aber als Referenz beigegeben. Sie können sie mit dem jConnect-Treiber einsetzen, um die Performance der Zeichensatzkonvertierung zu erhöhen.

Hinweis Tests bei Sybase haben ergeben, dass die Klasse **SunloConverter** die Performance auf allen VMs steigern konnte, auf denen sie getestet wurde. Java Software Division von Sun Microsystems, Inc. behält sich aber das Recht vor, die Klassen **sun.io** bei zukünftigen Versionen des JDKs zu entfernen oder zu verändern, so dass die Klasse **SunloConverter** mit späteren Versionen möglicherweise nicht mehr kompatibel sein wird.

Um die Klasse **SunloConverter** einzusetzen, müssen Sie die jConnect-Beispielanwendungen installieren. Vollständige Anweisungen zum Installieren von jConnect und seinen Komponenten, einschließlich den Beispielanwendungen, finden Sie in der Dokumentation *Sybase jConnect for JDBC Installationshandbuch*. Wenn die Beispiele installiert sind, können Sie die Verbindungseigenschaft `CHARSET_CONVERTER_CLASS` so setzen, dass sie die Klasse **SunloConverter** im Unterverzeichnis *sample* (jConnect 4.x) oder *sample2* (jConnect 5.x) unter Ihrem jConnect-Installationsverzeichnis referenziert.

Optimierung der Performance für Prepared Statements in Dynamic SQL

In Embedded SQL sind dynamische Anweisungen solche SQL-Anweisungen, die zur Laufzeit und nicht statisch kompiliert werden müssen. Typischerweise enthalten dynamische Anweisungen Eingabeparameter. Dies ist aber nicht die Regel. In SQL wird der Befehl **prepare** verwendet, um eine dynamische Anweisung vorzukompilieren und zu speichern, damit sie während einer Sitzung mehrfach ausgeführt werden kann, ohne rekompiliert werden zu müssen.

Wenn eine Anweisung mehrere Male in einer Sitzung verwendet wird, erreicht man durch Vorkompilieren bessere Performance, als wenn man sie an die Datenbank sendet und jedes Mal neu kompiliert, sobald sie gebraucht wird. Je komplexer die Anweisung, desto mehr Performance-Vorteile kann man dadurch erzielen.

Wenn eine Anweisung voraussichtlich nur einige wenige Male benutzt wird, ist das Vorkompilieren möglicherweise kein Vorteil, da durch das Vorkompilieren, Speichern und die spätere Aufhebung der Zuweisung in der Datenbank Overhead entsteht.

Das Vorkompilieren einer dynamischen SQL-Anweisung für das Ausführen und Speichern im Speicher ist zeit- und ressourcenaufwendig. Wenn eine Anweisung voraussichtlich nicht mehrfach während einer Sitzung verwendet wird, kann es sein, dass die Overheadkosten für ein **prepare** in der Datenbank die möglichen Vorteile überwiegen. Zu berücksichtigen ist auch, dass eine dynamische SQL-Anweisung nach dem "Prepare" in der Datenbank einer gespeicherten Prozedur sehr ähnlich ist. In manchen Fällen kann es daher besser sein, gespeicherte Prozeduren zu erstellen und auf dem Server resident zu halten, anstatt Prepared Statements in der Anwendung zu definieren. Dies wird unter "[Wahl zwischen Prepared Statements und gespeicherte Prozeduren](#)" auf Seite 120 im einzelnen besprochen.

Sie können jConnect benutzen, um die Performance von dynamischen SQL-Anweisungen in einer Sybase-Datenbank zu optimieren:

- Erstellen Sie **PreparedStatement**-Objekte, die vorkompilierte Anweisungen enthalten, wenn eine Anweisung voraussichtlich mehrere Male in einer Sitzung ausgeführt wird.
- Erstellen Sie **PreparedStatement**-Objekte, die nicht kompilierte SQL-Anweisungen enthalten, wenn eine Anweisung in einer Sitzung nur selten ausgeführt wird.

Wie in den folgenden Abschnitten beschrieben wird, hängt der optimale Weg zur Einstellung der Verbindungseigenschaft `DYNAMIC_PREPARE` und zur Erstellung von **PreparedStatement**-Objekten wahrscheinlich davon ab, ob Ihre Anwendung auf andere JDBC-Treiber portierbar sein muss, oder ob Sie eine Anwendung schreiben, die jConnect-spezifische Erweiterungen von JDBC zulässt.

jConnect Version 4.1 und höher bietet Funktionen für die Performanceoptimierung dynamischer SQL-Anweisungen.

Wahl zwischen Prepared Statements und gespeicherte Prozeduren

Wenn Sie ein **PreparedStatement**-Objekt erstellen, das eine nicht kompilierte dynamische SQL-Anweisung enthält, wird die Anweisung nach dem Kompilieren in der Datenbank im wesentlichen eine Stored Procedure, die im Speicher gehalten und der mit der Sitzung verknüpften Datenstruktur zugeordnet wird. Wenn Sie sich entscheiden müssen, ob Sie in Ihrer Anwendung gespeicherte Prozeduren in der Datenbank halten oder **PreparedStatement**-Objekte erstellen wollen, die kompilierte SQL-Anweisungen enthalten, müssen Sie Überlegungen zum Ressourcenbedarf und zur Pflege der Anwendung einbeziehen:

- Wenn eine gespeicherte Prozedur kompiliert ist, steht sie global allen Verbindungen zur Verfügung. Eine dynamische SQL-Anweisung in einem **PreparedStatement**-Objekt muss hingegen in jeder Sitzung, in der sie verwendet wird, kompiliert und deallokiert werden.
- Wenn Ihre Anwendung auf mehrere Datenbanken zugreift, bedeutet der Einsatz von gespeicherten Prozeduren, dass dieselben gespeicherten Prozeduren in allen Zieldatenbanken verfügbar sein müssen. Dies kann ein Problem bei der Datenbankwartung bewirken. Wenn Sie **PreparedStatement**-Objekte für dynamische SQL-Anweisungen verwenden, können Sie dieses Problem umgehen.
- Wenn Ihre Anwendung **CallableStatement**-Objekte erstellt, um gespeicherte Prozeduren aufzurufen, können Sie SQL-Code und Tabellenreferenzen in den gespeicherten Prozeduren verkapseln. Ohne Eingriff in die Anwendung können dann Änderungen der Basis-Datenbank oder des SQL-Codes vorgenommen werden.

Prepared Statements in portierbaren Anwendungen

Wenn Ihre Anwendung auf Datenbanken unterschiedlicher Anbieter laufen soll und Sie wollen, dass einige **PreparedStatement**-Objekte vorkompilierte Anweisungen enthalten, andere aber nicht kompilierte, gehen Sie wie folgt vor:

- Wenn Sie auf eine Sybase-Datenbank zugreifen, sorgen Sie dafür, dass die Verbindungseigenschaft `DYNAMIC_PREPARE` auf "true" gesetzt ist.
- Um **PreparedStatement**-Objekte zurückzugeben, die vorkompilierte Anweisungen enthalten, verwenden Sie **Connection.prepareStatement()** auf die Standardweise:

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

- Um **PreparedStatement**-Objekte zurückzugeben, die unkompliierte Anweisungen enthalten, verwenden Sie **Connection.prepareCall()**.

Connection.prepareCall() gibt ein **CallableStatement**-Objekt zurück, aber da **CallableStatement** eine Unterklasse von **PreparedStatement** ist, können Sie ein **CallableStatement**-Objekt an ein **PreparedStatement**-Objekt anbinden, wie dies im folgenden Beispiel geschieht:

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

Das **PreparedStatement**-Objekt *ps_uncomp* enthält garantiert eine nicht kompilierte Anweisung, da nur **Connection.prepareStatement()** so eingerichtet ist, dass **PreparedStatement**-Objekte zurückgegeben werden, die vorkompilierte Anweisungen enthalten.

Prepared Statements in Anwendungen mit jConnect-Erweiterungen

Wenn die Portierbarkeit auf andere Treiber für Sie kein Problem darstellt, können Sie einen Programmcode schreiben, der **SybConnection.prepareStatement()** benutzt, um festzulegen, ob ein **PreparedStatement**-Objekt vorkompilierte oder nicht kompilierte Anweisungen enthält. In diesem Fall richtet sich die Programmierung der Prepared Statements in der Regel danach, ob die meisten dynamischen Anweisungen in einer Anwendung voraussichtlich mehrfach aufgerufen werden oder nicht.

Wenn die meisten dynamischen Anweisungen nur selten ausgeführt werden

In einer Anwendung, deren dynamische SQL-Anweisungen in der Regel nur einmal oder zweimal in einer Sitzung ausgeführt werden, gilt folgendes:

- Setzen Sie die Verbindungseigenschaft `DYNAMIC_PREPARE` auf `"false"`.
- Um **PreparedStatement**-Objekte zurückzugeben, die nicht kompilierte Anweisungen enthalten, verwenden Sie **Connection.prepareStatement()** auf die Standardweise:

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

- Um **PreparedStatement**-Objekte zurückzugeben, die vorkompilierte Anweisungen enthalten, verwenden Sie **SybConnection.prepareStatement()** mit dem Parameter *dynamic* auf `"true:"`

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

Wenn die meisten dynamischen Anweisungen mehrfach ausgeführt werden

Wenn die meisten dynamischen Anweisungen in einer Anwendung im Verlauf einer Sitzung voraussichtlich mehrfach ausgeführt werden, gehen Sie wie folgt vor:

- Setzen Sie die Verbindungseigenschaft `DYNAMIC_PREPARE` auf `"true"`.
- Um **PreparedStatement**-Objekte zurückzugeben, die vorkompilierte Anweisungen enthalten, verwenden Sie **Connection.prepareStatement()** auf die Standardweise:

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

- Um **PreparedStatement**-Objekte zurückzugeben, die nicht kompilierte Anweisungen enthalten, können Sie entweder **Connection.prepareCall()** (siehe dritten Aufzählungspunkt unter [Prepared Statements in portierbaren Anwendungen](#)) oder **SybConnection.prepareStatement()**, mit dem Parameter *dynamic* auf `"false"` verwenden:

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

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

Connection.prepareStatement()

jdbcConnect benutzt **Connection.prepareStatement()**, so dass festgelegt werden kann, dass entweder vorkompilierte SQL-Anweisungen oder nicht kompilierte SQL-Anweisungen in **PreparedStatement**-Objekten zurückgegeben werden können. Wenn **Connection.prepareStatement()** so eingestellt ist, dass vorkompilierte SQL-Anweisungen in **PreparedStatement**-Objekten zurückgegeben werden, werden dynamische SQL-Anweisungen an die Datenbank gesendet, um sie so zu kompilieren und zu speichern, wie dies durch das direkte Ausführen des Befehls **prepare** erfolgen würde. Wenn **Connection.prepareStatement()** so eingestellt ist, dass nicht kompilierte SQL-Anweisungen zurückgegeben werden, erfolgt die Rückgabe in **PreparedStatement**-Objekten, ohne sie an die Datenbank zu senden.

Welche Art von SQL-Anweisungen **Connection.prepareStatement()** zurückgibt, wird von der Verbindungseigenschaft **DYNAMIC_PREPARE** festgelegt und gilt für eine ganze Sitzung.

Bei Sybase-spezifischen Anwendungen stellt jdbcConnect 5.0 eine **prepareStatement()**-Methode in der Klasse **SybConnection** bereit. Mit **SybConnection.prepareStatement()** können Sie angeben, ob eine bestimmte dynamische SQL-Anweisung vorkompiliert werden soll, unabhängig von der Einstellung der Verbindungseigenschaft **DYNAMIC_PREPARE** auf Sitzungsniveau.

DYNAMIC_PREPARE, Verbindungseigenschaft

DYNAMIC_PREPARE ist eine mit Booleschen Werten operierende Verbindungseigenschaft für die Aktivierung von dynamischen SQL-Prepared Statements:

- Wenn **DYNAMIC_PREPARE** auf "true" gesetzt ist, versucht jeder Aufruf von **Connection.prepareStatement()** während einer Sitzung, eine vorkompilierte Anweisung in einem **PreparedStatement**-Objekt zurückzugeben.

In diesem Fall gilt: Wenn ein **PreparedStatement** ausgeführt wird, ist die darin enthaltene Anweisung in der Datenbank bereits mit Platzhaltern für dynamisch zugeteilte Parameter vorkompiliert, und die Anweisung braucht daher nur ausgeführt zu werden.

- Wenn **DYNAMIC_PREPARE** für eine Verbindung auf "false" gesetzt ist, enthält das von **Connection.prepareStatement()** zurückgegebene **PreparedStatement** keine vorkompilierte Anweisung.

In diesem Fall gilt: Jedes Mal, wenn ein **PreparedStatement** ausgeführt wird, muss die darin enthaltene dynamische SQL-Anweisung zum Kompilieren und Ausführen an die Datenbank geschickt werden.

Der Standardwert für `DYNAMIC_PREPARE` ist "false".

Im folgenden Beispiel ist `DYNAMIC_PREPARE` auf "true" gesetzt, um die Vorkompilierung der dynamischen SQL-Anweisungen zu ermöglichen. In diesem Beispiel steht **props** für ein **Properties**-Objekt, in dem die Verbindungseigenschaften festgelegt werden.

```
...
props.put("DYNAMIC_PREPARE", "true")
Connection conn = DriverManager.getConnection(url, props);
```

Wenn `DYNAMIC_PREPARE` auf "true" gesetzt ist, müssen Sie Folgendes beachten:

- Nicht alle dynamischen Anweisungen können mit dem Befehl **prepare** kompiliert werden. Der SQL-92-Standard enthält einige Einschränkungen für die Anweisungen, die mit dem Befehl **prepare** verwendet werden können, und für Datenbanken von Drittanbietern können weitere Beschränkungen gelten.
- Wenn die Datenbank eine Fehlermeldung ausgibt, weil sie eine Anweisung, die ihr durch **Connection.prepareStatement()** gesendet wurde, nicht vorkompilieren und speichern kann, fängt `jdbc` die Fehlermeldung ab und gibt ein **PreparedStatement**-Objekt zurück, das eine nicht kompilierte dynamische SQL-Anweisung enthält. Jedes Mal, wenn das **PreparedStatement**-Objekt ausgeführt wird, wird die Anweisung an die Datenbank zurückgesandt, um kompiliert und ausgeführt zu werden.
- Eine vorkompilierte Anweisung bleibt im Speicher der Datenbank und wird dort entweder bis zum Ende der Sitzung oder bis zum expliziten Schließen ihres **PreparedStatement**-Objekts resident. Die Abfalldatensammlung in einem **PreparedStatement**-Objekt entfernt das Prepared Statement nicht aus der Datenbank.

Als Faustregel gilt: Schließen Sie **PreparedStatement**-Objekte explizit nach ihrer letzten Verwendung, damit sich Prepared Statements nicht im Arbeitsspeicher des Servers ansammeln und die Performance verschlechtern.

SybConnection.prepareStatement()

Wenn Ihre Anwendung jConnect-spezifische Erweiterungen von JDBC zulässt, können Sie die **SybConnection.prepareStatement()**-Erweiterungsmethode verwenden, um dynamische SQL-Anweisungen in **PreparedStatement**-Objekten zurückzugeben.

```
PreparedStatement SybConnection.prepareStatement(String sql_stmt,  
        boolean dynamic) throws SQLException
```

SybConnection.prepareStatement() kann je nach der Einstellung des *dynamic*-Parameters **PreparedStatement**-Objekte zurückgeben, die vorkompilierte oder nicht kompilierte SQL-Anweisungen enthalten. Wenn *dynamic* auf "true" ist, gibt **SybConnection.prepareStatement()** ein **PreparedStatement**-Objekt mit einer vorkompilierten SQL-Anweisung zurück. Wenn *dynamic* auf "false" gesetzt ist, gibt die Methode ein **PreparedStatement**-Objekt mit einer nicht kompilierten SQL-Anweisung zurück.

Das folgende Beispiel zeigt die Verwendung von **SybConnection.prepareStatement()** zur Rückgabe eines **PreparedStatement**-Objekts, das eine vorkompilierte Anweisung enthält:

```
PreparedStatement precomp_stmt =  
    ((SybConnection) conn).prepareStatement( "SELECT * FROM  
    authors WHERE au_fname LIKE ?", true);
```

In diesem Beispiel ist das Verbindungsobjekt *conn* in ein **SybConnection**-Objekt eingebunden, damit **SybConnection.prepareStatement()** verwendet werden kann. Beachten Sie, dass die SQL-Zeichenfolge, die an **SybConnection.prepareStatement()** übergeben wird, in der Datenbank vorkompiliert wird, auch wenn die Verbindungseigenschaft **DYNAMIC_PREPARE** auf "false" gesetzt ist.

Wenn die Datenbank eine Fehlermeldung ausgibt, weil sie eine Anweisung, die ihr durch **SybConnection.prepareStatement()** gesendet wurde, nicht vorkompilieren und speichern kann, generiert jConnect eine **SQLException**-Fehlermeldung, und der Aufruf kann die Rückgabe eines **PreparedStatement**-Objekts nicht ausführen. Dies ist anders als bei **Connection.prepareStatement()**: In diesem Fall fängt jConnect SQL-Fehler ab und gibt ein **PreparedStatement**-Objekt zurück, das eine nicht kompilierte Anweisung enthält.

Cursor-Performance

Wenn Sie die Methode **Statement.setCursorName()** oder **setFetchSize()** in der Klasse **SybCursorResultSet** verwenden, erstellt jConnect einen Cursor in der Datenbank. Andere Methoden veranlassen jConnect, einen Cursor zu öffnen, abzufragen und zu aktualisieren.

Die jConnect-Versionen vor 4.0 sind so eingerichtet, dass die Erstellung und Manipulation eines Cursors nur durch das Senden von SQL-Anweisungen mit expliziten Cursorbefehlen für die syntaktische Analyse und Kompilation an die Datenbank erfolgt.

Die Versionen 4.0 und höher von jConnect sind so eingerichtet, dass die Erstellung und Manipulation eines Cursors entweder durch das Senden von SQL-Anweisungen an die Datenbank oder durch die Codierung von Cursorbefehlen als Token innerhalb des Kommunikationsprotokolls Tabular Data Stream (TDS) erfolgt. Ein Cursor der ersten Art ist der "Language-Cursor", ein Cursor der zweiten Art ist der "Protocol-Cursor".

Ein Protocol-Cursor bietet bessere Performance als ein Language-Cursor. Außerdem unterstützen nicht alle Datenbanken den Language-Cursor. Beispiel: Datenbanken in Adaptive Server Anywhere unterstützen den Language-Cursor nicht.

In jConnect ist jeder Cursor standardmäßig ein Protocol-Cursor. Mit der Verbindungseigenschaft **LANGUAGE_CURSOR** können Sie aber veranlassen, dass der Cursor durch Language-Befehle in der Datenbank erstellt und manipuliert wird.

LANGUAGE_CURSOR, Verbindungseigenschaft

LANGUAGE_CURSOR ist eine durch Boolesche Werte gesteuerte Verbindungseigenschaft in jConnect, mit der Sie festlegen können, ob ein Cursor als Protocol-Cursor oder als Language-Cursor erstellt wird:

- Wenn **LANGUAGE_CURSOR** auf "false" gesetzt ist, ist jeder Cursor, der während einer Sitzung erstellt wird, ein Protocol-Cursor, was die Performance verbessert. jConnect erstellt und manipuliert den Cursor, indem Cursorbefehle als Token im TDS-Protokoll übermittelt werden.

Standardmäßig steht **LANGUAGE_CURSOR** auf "false".

- Wenn `LANGUAGE_CURSOR` auf "true" gesetzt ist, wird ein während einer Sitzung erstellter Cursor als Language-Cursor erstellt. jConnect erstellt und manipuliert den Cursor, indem SQL-Anweisungen für die syntaktische Analyse und Kompilation an die Datenbank geschickt werden.

Es ist kein Grund bekannt, warum `LANGUAGE_CURSOR` besser auf "true" gesetzt werden sollte, die Option steht aber zur Verfügung, falls eine Anwendung ungewöhnliches Verhalten zeigt, wenn `LANGUAGE_CURSOR` auf "false" gesetzt ist.

jConnect-Anwendungen migrieren

In diesem Kapitel wird erläutert, wie Anwendungen, die Sybase-Erweiterungen verwenden, von jConnect Version 4.0 oder früher auf jConnect Version 4.1 oder höher migriert werden können.

Folgende Themen werden behandelt:

Name	Seite
Anwendungen auf jConnect 4.1 migrieren	130
Anwendungen auf jConnect 5.x migrieren	130
Anwendungen auf jConnect 4.2 und 5.2 migrieren	130
Änderungen bei Sybase-Erweiterungen	133

jConnect-Anwendungen migrieren

Anwendungen auf jConnect 4.1 migrieren

jConnect 4.1 ist rückwärtskompatibel mit früheren Versionen von jConnect. Alle bestehenden Anwendungen müßten ohne Neukompilierung weiterhin funktionieren.

Wenn Sie neue Anwendungen entwickeln, die Sybase-Erweiterungen verwenden, benutzen Sie die Schnittstellen in **com.sybase.jdbcx**.

Diese einheitliche Schnittstelle ermöglicht Ihnen ein Upgrade von Anwendungen auf jConnect Version 4.1 und höher mit nur geringfügigen Änderungen. Unter ["Änderungen bei Sybase-Erweiterungen" auf Seite 133](#) finden Sie die Änderungen, die an den Sybase-Erweiterungen vorgenommen wurden.

Anwendungen auf jConnect 5.x migrieren

Das Paket mit den Klassen für den Sybase jConnect 5.x-Treiber ist *com.sybase.jdbc2.jdbc.SybDriver*. Es unterscheidet sich von früheren jConnect-Versionen. Der Quellcode von Anwendungen muss an der Stelle geändert werden, an der sie SybDriver laden. Danach muss eine Neukompilierung mit der Java™ 2-Plattform erfolgen.

Nachstehend finden Sie ein Beispiel für das Laden des Treibers in jConnect 5.0:

```
Driver d =  
(Driver)Class.forName( "com.sybase.jdbc2.jdbc.SybDriver" ).newInstance() ;  
DriverManager.registerDriver(d) ;
```

Wenn Ihre Anwendungen Sybase-Erweiterungen der JDBC API verwenden, ändern Sie die Importanweisungen von **com.sybase.jdbc** bzw. **com.sybase.utils** auf **com.sybase.jdbcx**. Unter ["Änderungen bei Sybase-Erweiterungen" auf Seite 133](#) finden Sie eine Angabe der Änderungen, die an den Sybase-Erweiterungen vorgenommen wurden.

Weitere Beispiele für den Einsatz von Sybase-Erweiterungen finden Sie in den Programmcodebeispielen, die mit jConnect geliefert wurden.

Anwendungen auf jConnect 4.2 und 5.2 migrieren

Wenn Sie eine Aktualisierung auf jConnect 4.2 oder 5.2 von früheren Versionen durchführen, zeigt Ihnen die folgende Tabelle, welche Aktualisierungspfade eine Änderung und ein Rekompilieren des Quellcodes erfordern.

Legende:

- A** Änderung auf das **com.sybase.jdbcx**-Paket empfohlen.
- B** Ändern Sie CLASSPATH auf die neue Installationsstruktur.
- C** Rekompilieren Sie, um den neuen jConnect 5.x-Treiber zu verwenden.

Zusätzliche Details finden Sie unterhalb.

Upgrade von jConnect Version	Auf jConnect Version			
	4.1	4.2	5.0	5.2
4.0 & früher	A	AB	AC	ABC
4.1	-	AB	AC	ABC
4.2	-	-	kein unterstützter Pfad	AC
5.0	-	-	-	B

❖ A. Verwenden Sie die neuen Sybase-Erweiterungen.

- 1 Ändern Sie die folgenden Paket-Importe:

```
import com.sybase.jdbc.*
```

Ändern Sie folgendermaßen:

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

- 2 Verwenden Sie die neuen Sybase-Erweiterungs-APIs. Siehe ["Änderungen bei Sybase-Erweiterungen" auf Seite 133](#).

❖ B. Ändern Sie CLASSPATH auf die neue JDBC_HOME Installationsstruktur.

Setzen Sie JDBC_HOME auf das Hauptverzeichnis des jConnect-Treibers, den Sie installiert haben.

Beispiel:

- Für jConnect 4.2:
JDBC_HOME=jConnect-4_2
- Für jConnect 5.0:

JDBC_HOME=<jConnect-Installationsverzeichnis>

Weitere Informationen zum Setzen von JDBC_HOME finden Sie unter "Umgebungsvariable setzen" im Kapitel 1 der Dokumentation *jConnect for JDBC Installationshandbuch*.

Änderung der Version		CLASSPATH umfasst
Von	4.1	JDBC_HOME/classes
Auf	5.2	JDBC_HOME/jconn2.jar
Von	4.1	JDBC_HOME/classes
Auf	4.2	JDBC_HOME/classes
Von	5.0	JDBC_HOME/classes/jconn2.jar
Auf	5.2	JDBC_HOME/classes/jconn2.jar

❖ **C. Rekompilieren Sie, um den neuen jConnect 5.x-Treiber zu verwenden.**

- Ändern Sie den Quellcode, aus dem der Treiber geladen wird:

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

Ändern Sie folgendermaßen:

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


Änderungen bei Sybase-Erweiterungen

Ein neues Paket **com.sybase.jdbcx**, wurde den jConnect-Versionen 4.1, 4.2 und 5.x hinzugefügt. Das Paket enthält alle Sybase-Erweiterungen von JDBC. In früheren Versionen von jConnect standen diese Versionen in den Paketen **com.sybase.jdbc** und **com.sybase.utils** zur Verfügung.

com.sybase.jdbcx bietet eine einheitliche Schnittstelle in allen Versionen von jConnect. Alle Sybase-Erweiterungen sind als Java-Schnittstellen definiert, so dass die dadurch betroffenen Implementierungen ohne Beeinträchtigung der Anwendungen geändert werden können, die mit diesen Schnittstellen erstellt wurden.

Wenn Sie neue Anwendungen entwickeln, die Sybase-Erweiterungen benutzen, verwenden Sie **com.sybase.jdbcx**. Die Schnittstellen in diesem neuen Paket ermöglichen ein Upgrade der Anwendungen auf Versionen von jConnect nach Version 4.0 mit nur geringfügigen Änderungen.

Hinweis Anwendungen, die früher mit den Sybase-Erweiterungen der JDBC API erstellt wurden (verfügbar über **com.sybase.jdbc** und **com.sybase.utils**), funktionieren auch unter jConnect 4.x. Alle Sybase-Erweiterungen in **com.sybase.jdbc** und **com.sybase.utils** wurden allerdings als "deprecated" markiert.

Einige Sybase-Erweiterungen wurden geändert, um die neue **com.sybase.jdbcx**-Schnittstelle zu berücksichtigen.

Änderungsbeispiel

Wenn eine Anwendung **SybMessageHandler** benutzt, werden folgende Unterschiede im Programmcode aktuell:

- **jConnect 4.0** Code:

```
import com.sybase.jdbc.SybConnection;
import com.sybase.jdbc.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setMessageHandler(new ConnectionMsgHandler());
```

- **jConnect 4.1**-Code und höher:

Änderungen bei Sybase-Erweiterungen

```
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setSybMessageHandler(new ConnectionMsgHandler());
```

Weitere Beispiele für den Einsatz von Sybase-Erweiterungen finden Sie in den Programmcodbeispielen, die mit jConnect geliefert wurden.

Geänderte Methodennamen

In der folgenden Tabelle wird gezeigt, wie die Methoden in der neuen Schnittstelle umbenannt wurden.

Klasse	Alter Name	Neuer Name
SybConnection	getCapture()	createCapture()
SybConnection	setMessageHandler()	setSybMessageHandler()
SybConnection	getMessageHandler()	getSybMessageHandler()
SybStatement	setMessageHandler()	setSybMessageHandler()
SybStatement	getMessageHandler()	getSybMessageHandler()

Debug-Klasse einrichten

Direkte statische Referenzen auf die **Debug**-Klasse werden nicht mehr unterstützt, sind aber im Paket **com.sybase.utils** mit der Markierung "deprecated" noch enthalten. Um die Fehlersuchfunktionen von jConnect zu benutzen, verwenden Sie die Methode **getDebug()** der Klasse **SybDriver**, um eine Referenz auf die Klasse **Debug** zu erhalten. Beispiel:

```
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);
```

Eine vollständige Liste der Sybase-Erweiterungen befindet sich in der jConnect javadoc Dokumentation im Verzeichnis *docs/* des jConnect-Installationsverzeichnis.

Webserver Gateways

In diesem Kapitel wird das Konzept des Webserver Gateways beschrieben und erklärt, wie ein solches Gateway in jConnect verwendet wird.

Das Kapitel ist in folgende Abschnitte unterteilt:

Name	Seite
Hinweise zu Webserver Gateways	138
Cascade Gateway verwenden	141
TDS-Tunnelling Servlet benutzen	147

Hinweise zu Webserver Gateways

Wenn Ihr Datenbankserver auf einem anderen Host untergebracht ist als Ihr Webserver, oder wenn Sie Internet-Anwendungen entwickeln, die sich über eine Firewall mit einem sicheren Datenbankserver verbinden wollen, brauchen Sie ein Gateway, das als Proxy fungiert und einen Pfad zum Datenbankserver bereitstellt.

Um die Verbindung mit Servern über das Protokoll Secure Sockets Layer (SSL) aufzunehmen, bietet jConnect ein Java-Servlet, das Sie auf jedem Webserver installieren können, der das Interface **javax.servlet** unterstützt. Dieses Servlet richtet jConnect so ein, dass Verschlüsselungsfunktionen mit dem Webserver als Gateway unterstützt werden.

Hinweis jConnect enthält auch Unterstützung für SSL auf dem Client-System. Hinweise zur clientseitigen Unterstützung von SSL durch jConnect finden Sie unter ["Benutzerdefinierte Socket-Plug-Ins implementieren"](#) auf Seite 28.

Die jConnect-Installation enthält ein Cascade-Gateway, eine leicht veränderte Version des Cascade Webservers. Das Cascade Gateway unterstützt die Methode CONNECT HTTP, mit der Daten im Format Tabular Data Stream (TDS) durch das HTTP-Protokoll geführt werden können. Sie können das Cascade Gateway benutzen, um sich mit einem Datenbankserver zu verbinden, der auf einem anderen Host läuft als der Webserver, aber nicht durch eine Firewall gehen muss. Siehe ["Cascade Gateway verwenden"](#) auf Seite 141.

TDS-Tunnelling

jConnect benutzt TDS zur Kommunikation mit Datenbankservern. HTTP-tunnelled TDS ist für die Weitergabe von Anforderungen nützlich. Anforderungen von einem Client zu einem Backend-Server, die durch das Gateway gehen, enthalten TDS im Daten-Hauptteil der Anforderung. Der Kopfteil der Anforderung gibt die Länge der TDS-Daten an, die im Anforderungspaket enthalten sind.

TDS ist ein verbindungsorientiertes Protokoll, während HTTP dies nicht ist. Um Sicherheitsfunktionen wie die Verschlüsselung für Internet-Anwendungen zu unterstützen, benutzt jConnect ein TDS-tunneling-Servlet zur Aufrechterhaltung einer logischen Verbindung über HTTP-Anforderungen. Das Servlet generiert eine Sitzungs-ID während der ersten Login-Anforderung, und die Sitzungs-ID ist im Kopfteil jeder nachfolgenden Anforderung enthalten. Durch den Einsatz von Sitzungs-IDs können Sie aktive Sitzungen identifizieren und sogar eine Sitzung wiederaufnehmen, wenn das Servlet eine offene Verbindung mit dieser Sitzungs-ID behält.

Die logische Verbindung, die vom TDS-tunneling-Servlet bereitgestellt wird, versetzt jConnect in die Lage, verschlüsselte Kommunikation zwischen zwei Systemen zu unterstützen. Beispiel: Ein jConnect-Client, bei dem die Verbindungseigenschaft `CONNECT_PROTOCOL` auf "https" gesetzt ist und der sich mit einem Webserver verbindet, auf dem das TDS-tunnelling Servlet läuft.

jConnect und Gateway-Konfiguration

Es gibt mehrere Optionen für die Einrichtung Ihrer Webserver und Adaptive Server. Vier gemeinsame Konfigurationen werden nachstehend beschrieben. Diese Beispiele zeigen, wo der jConnect-Treiber zu installieren ist, und wann das Cascade-Gateway oder ein Gateway mit dem TDS-tunneling-Servlet verwendet werden müssen.

Webserver und Adaptive Server auf einem Host

In dieser zweischichtigen Konfiguration werden der Webserver und Adaptive Server beide auf demselben Host installiert.

- Installieren Sie jConnect auf dem Webserver-Host.
- Kein Gateway erforderlich.

Dedizierter JDBC Webserver und Adaptive Server auf einem Host

Mit dieser Konfiguration haben Sie einen eigenen Host für Ihren Hauptserver. Ein zweiter Host wird von einem Webserver gemeinsam genutzt, der speziell für den Zugriff auf Adaptive Server eingerichtet wurde. Links vom Hauptserver leiten direkte Anforderungen, die SQL-Zugriff verlangen, auf den dedizierten Webserver.

- Installieren Sie jConnect auf dem zweiten Host (Adaptive Server).

- Kein Gateway erforderlich.

Webserver und Adaptive Server auf getrennten Hosts

In dieser dreischichtigen Konfiguration ist Adaptive Server auf einem vom Webserver getrennten Host untergebracht. jConnect benötigt ein Gateway, das als Proxy zu Adaptive Server fungieren soll.

- Installieren Sie jConnect auf dem Webserver-Host.
- Dazu ist entweder ein TDS-tunnelling Servlet oder das Cascade Gateway erforderlich.

Verbindung mit einem Server über eine Firewall

Zur Verbindung mit einem Server, der durch eine Firewall geschützt ist, müssen Sie einen Webserver mit dem TDS-tunneling-Servlet benutzen, um die Übertragung der Antworten auf Datenbankankorderungen über das Internet zu unterstützen.

- Installieren Sie jConnect auf dem Webserver-Host.
- Dafür ist ein Webserver erforderlich, der das **javax.servlet**-Interface unterstützt.

Cascade Gateway verwenden

Wenn Ihr Datenbankserver auf einem anderen Host läuft als der Webserver, brauchen Sie auch ohne Firewall ein Gateway, das als Proxy einen Weg zum Datenbankserver bereit stellt.

Die jConnect-Installation enthält ein Cascade-Gateway, eine leicht veränderte Version des Cascade Webservers, der von David Wilkerson in Java programmiert wurde. (email: davidw@cascade.org.uk; Web site: <http://www.cascade.org.uk/>).

Dieses Gateway empfängt Pakete und leitet sie weiter, wobei von HTTP auf TDS umgeschaltet wird, wenn eine Anforderung an eine Adaptive Server-Installation geht, und von TDS auf HTTP, wenn die Ergebnisse zurückgegeben werden.

Hinweis Das Cascade-Gateway unterstützt keine Verschlüsselung, und kann daher bei Internet-Anwendungen nicht eingesetzt werden, die mit einem Backend-Server über eine Firewall verbunden sind.

Syntaxanforderungen

- Wenn Sie jConnect nicht im Standard-Installationsverzeichnis installiert haben, müssen Sie *www.dos* (DOS) oder *www.template* (UNIX) bearbeiten, um alle Verweise auf das Standard-Installationsverzeichnis auf das tatsächliche jConnect-Installationsverzeichnis umzuleiten.
- Das Cascade Gateway und Ihr Webserver müssen auf demselben Hostrechner ausgeführt werden. Auf diese Weise verbinden sich Applets mit demselben Hostrechner wie der Webserver, allerdings am Port, der vom Cascade Gateway kontrolliert wird. Das Gateway routet die Anforderung an die geeignete Datenbank. Wenn Sie sehen wollen, wie dies programmiert wird, sehen Sie sich den Quellcode von *Isql.java* und *gateway.html* im Unterverzeichnis *sample* (jConnect 4.x) oder *sample2* (jConnect 5.x) des jConnect-Installationsverzeichnisses an. Suchen Sie nach "proxy".

Wenn Ihr Datenbankserver auf demselben Hostrechner läuft wie Ihr Webserver, brauchen Sie das Cascade Gateway nicht.

Cascade Gateway installieren

Das Cascade Gateway wird installiert, wenn Sie eine volle jConnect-Installation durchführen, indem Sie entweder das Installationsprogramm von jConnect oder die Dateien *install.bat* bzw. *install.sh* benutzen. Sie können das jConnect-Installationsprogramm auch verwenden, um nur die Dateien des Cascade Gateways zu installieren, wenn dies erforderlich ist. Hinweise finden Sie in der Dokumentation *Sybase jConnect for JDBC Installationsanleitung und Versionshinweise*.

Cascade Gateway starten

Befolgen Sie die nachstehend angeführten Anweisungen für die jeweilige Plattform, um das Cascade Gateway zu testen.

Windows NT und Windows 95

- 1 Öffnen Sie ein DOS-Fenster und wechseln Sie in das jConnect-Installationsverzeichnis.

Ihre Umgebungsvariable JDBC_HOME muss auf dieses Verzeichnis verweisen.

- 2 Starten Sie das Cascade Gateway mit folgender Eingabe:

```
httpd
```

Wenn der Befehl erfolgreich ausgeführt wird, erscheint die Ausgabe von *httpd.bat* mit folgenden abschließenden Kommentaren:

```
HTTPDServer www.dos
```

UNIX

Wechseln Sie in das Verzeichnis, in dem Sie jConnect installiert haben (das JDBC_HOME-Verzeichnis). Geben Sie folgenden Befehl ein:

```
sh httpd.sh &
```

Fehlersuche

- Wenn Sie nach der Eingabe des **httpd**-Befehls keine Meldung sehen, läuft der Server nicht. Versuchen Sie die Eingabe des Befehls im Verbose-Modus nochmals.

Unter Windows rufen Sie ein DOS-Fenster auf und geben ein:

```
httpd -Dverbose=1 > Dateiname
```

Unter UNIX geben Sie ein:

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

In diesen Befehlen ist *Dateiname* die Ausgabedatei für Debug-Meldungen.

- Eventuell erscheint folgende Fehlermeldung:

```
HTTPServer: IOException: getRequest() Adresse wird  
bereits benutzt
```

Das bedeutet, dass auf der in der Datei *www.dos* (DOS) oder *www.template* (UNIX) (im Verzeichnis *JDBC_HOME*) angegebenen Portnummer bereits ein Prozess läuft. Dieser Fehler tritt auf, wenn Sie das Gateway starten.

Sie haben folgende Möglichkeiten:

- Sie können den Prozess stoppen, der gerade auf dem angegebenen Port läuft. Nachdem Sie überprüft haben, ob der Prozess wirklich beendet wurde, versuchen Sie nochmals, das Gateway zu starten;
oder
- Ändern Sie die Portnummer in der Datei *www.dos* oder *www.template* und modifizieren Sie dann die Datei *gateway.html* im Unterverzeichnis *sample* (jConnect 4.x) oder *sample2* (jConnect 5.x) von *JDBC_HOME*, indem Sie den Parameter *proxy* in "localhost: *Neuer_Port*" ändern.

Wenn Ihr Host nicht "localhost" ist (was der Fall sein wird, wenn Ihr Cascade HTTP Server und Browser auf verschiedenen Hosts untergebracht sind), müssen Sie darauf achten, dass der Parameter *proxy* den Namen des entfernten Hostrechners verwendet, und nicht "localhost".

Cascade Gateway testen

Um Ihre Systemeinstellung zu prüfen und das Cascade Gateway zu testen, können Sie ein Validierungsprogramm laufen lassen, das sich mit der Sybase-Demodatenbank verbindet.

Hinweis Das Validierungsprogramm "Validate" benutzt "localhost:8000", um das Gateway zu testen.

Aus dem DOS-Fenster unter Windows NT oder Windows 95 oder an der UNIX-Eingabeaufforderung wechseln Sie auf das Verzeichnis JDBC_HOME.

Für jConnect 4.x geben Sie Folgendes ein:

```
java sample.SybSample Validate
```

Für jConnect 5.x geben Sie Folgendes ein:

```
java sample2.SybSample Validate
```

Wenn die Validierung erfolgreich verläuft, sehen Sie die jConnect-Versionsnummer und die Meldung "Connected successfully" (Verbindung erfolgreich) im Ausgabefenster.

Fehlersuche

Wenn der Fehler "Befehl oder Dateiname nicht gefunden" (Windows 95) oder "Der Befehl ist entweder falsch geschrieben oder konnte nicht gefunden werden. Bitte überprüfen Sie die Schreibweise und die Umgebungsvariable 'PATH'" (Windows NT) ausgegeben wird, vergewissern Sie sich, dass die Path-Variable das Unterverzeichnis *\bin* des JDK-Home-Verzeichnisses enthält.

Die Datei *index.html* lesen

Benutzen Sie Ihren Webbrowser, um die Datei *index.html* im jConnect-Installationsverzeichnis aufzurufen. *index.html* bietet Links zur jConnect-Dokumentation und zu Programmcodebeispielen.

Hinweis Wenn Sie Netscape auf demselben Rechner verwenden, auf dem Sie jConnect installiert haben, müssen Sie sicherstellen, dass Ihr Browser keinen Zugriff auf die CLASSPATH-Umgebungsvariable hat. Siehe "Einschränkungen beim Setzen von CLASSPATH bei der Verwendung von Netscape" im Kapitel 3 der Dokumentation *Sybase jConnect for JDBC Installationsanleitung und Versionshinweise*.

- 1 Öffnen Sie den Webbrowser.
- 2 Geben Sie den URL ein, der zu Ihrer Systemeinstellung passt. Beispiel: Wenn Ihr Browser und das Cascade Gateway auf demselben Hostrechner laufen, geben Sie ein:

```
http://localhost:8000/index.html
```

Wenn der Browser und das Cascade Gateway auf unterschiedlichen Hostrechnern laufen, geben Sie ein:

```
http://Host:Port/index.html
```

Dabei gilt: *Host* ist der Name des Hostrechners, auf dem das Cascade Gateway läuft, und *Port* ist der Listening-Port.

Fehlersuche

Wenn Sie den richtigen Host, den richtigen Port und die richtige Dateiinformation eingegeben haben, der Browser aber diesen Link nicht öffnen kann, läuft das Cascade Gateway nicht. Siehe "[Cascade Gateway starten](#)" auf Seite 142.

Muster-Isql-Applet ausführen

Nachdem Sie die Datei *index.html* in Ihrem Browser geladen haben, gehen Sie wie folgt vor:

- 1 Klicken Sie auf "Run Sample JDBC Applets".
Damit wechseln Sie auf die jConnect Sample Programs Seite.

Cascade Gateway verwenden

- 2 Rollen Sie die Sample Programs Seite ab, bis Sie die Tabelle unter "Executable Samples" finden.
- 3 Suchen Sie "Isql.java" in der Tabelle und klicken Sie auf "Run" am Ende der Zeile.

Das Beispiel-Applet **Isql.java** führt eine einfache Abfrage in einer Beispiel-Datenbank durch und zeigt die Ergebnisse an. Das Applet zeigt einen Standard-Adaptive-Server-Hostnamen, eine Portnummer, einen Benutzernamen (*guest*), ein Kennwort (*sybase*) eine Datenbank und eine Abfrage. Mit den Standardwerten verbindet sich das Applet mit der Sybase-Demodatenbank. Es gibt Ergebnisse zurück, nachdem Sie auf "Go" geklickt haben.

Fehlersuche

Unter UNIX können Sie die Bildschirmdimensionen des Applets ändern, falls es nicht richtig angezeigt wird:

- 1 Bearbeiten Sie folgende Dateien mit einem Texteditor:

Für jConnect 4.x

\$JDBC_HOME/sample/gateway.html

Für jConnect 5.x

\$JDBC_HOME/sample2/gateway.html

- 2 Ändern Sie den Höhenparameter in Zeile 7 auf 650. Sie können mit unterschiedlichen Höheneinstellungen experimentieren.
- 3 Laden Sie die Webseite in Ihrem Browser nochmals.

Verbindung zum Cascade Gateway definieren

Um eine Verbindung in Ihrer Anwendung zu definieren, die das Cascade Gateway verwendet, setzen Sie den Namen des Hosts, auf dem das Cascade Gateway läuft, in den URL:

Host:Port

Dabei gilt: *Host* ist der Name des Hostrechners, auf dem das Cascade Gateway läuft, und *Port* ist der Listening-Port.

TDS-Tunnelling Servlet benutzen

Um das TDS-tunneling-Servlet zu benutzen, benötigen Sie einen Webserver, der die **javax.servlet**-Interfaces unterstützt, wie z.B. den Java Webserver von Sun Microsystems, Inc.™. Wenn Sie den Webserver installieren, nehmen Sie das jConnect TDS-tunneling-Servlet in die Liste der aktiven Servlets auf. Sie können auch Parameter für das Servlet setzen, um Verbindungs-Zeitüberschreitungen und maximale Paketgrößen festzulegen.

Mit dem TDS-tunneling-Servlet enthalten Anforderungen von einem Client an den Backend-Server, die durch das Gateway gehen, einen GET- oder POST-Befehl, die TDS-Sitzungs-ID (nach der ersten Anforderung), die Backend-Adresse und den Status der Anforderung.

TDS ist im Hauptteil der Anforderung enthalten. Zwei Felder im Header geben die Länge des TDS-Streams, sowie die Sitzungs-ID an, die durch das Gateway zugeordnet wird.

Wenn der Client eine Anforderung sendet, gibt das Header-Feld Content-Length die Größe des TDS-Inhalts an, und der Anforderungsbefehl ist POST. Wenn in der Anforderung keine TDS-Daten vorhanden sind, weil der Client entweder den nächsten Abschnitt der Antwortdaten aus dem Server abrufen, oder aber die Verbindung schließt, ist der Anforderungsbefehl GET.

Das folgende Beispiel zeigt, wie die Informationen zwischen dem Client und einem HTTPS-Gateway im TDS-tunneled-HTTPS-Protokoll fließen. Dieses Beispiel zeigt eine Verbindung mit einem Backend-Server namens DBSERVER und einer Portnummer "1234".

Tabelle 6-1: Login-Anforderung Client zu Gateway. Keine Sitzungs-ID.

<i>Abfrage</i>	POST/tds?ServerHost=dbserver&ServerPort=1234&Operation=more HTTP/1.0
<i>Header</i>	Content-Length: 605
<i>Inhalt</i> (TDS)	Login-Anforderung

Tabelle 6-2: Gateway an Client. Der Header enthält die Sitzungskennung, die vom TDS-Servlet zugewiesen wurde.

<i>Abfrage</i>	200 SUCCESS HTTP/1.0
<i>Header</i>	Content-Length: 210 TDS-Session: TDS00245817298274292
<i>Inhalt</i> (TDS)	Login-Bestätigung EED

Tabelle 6-3: Client an Gateway. Header für alle nachfolgenden Anforderungen enthalten die Sitzungs-ID.

Abfrage	POST/tds?TDS-Session=TDS00245817298274292&Operation=more HTTP/1.0
Header	Content-Length: 32
Inhalt (TDS)	Abfrage "SELECT * from authors"

Tabelle 6-4: Gateway an Client. Header für alle nachfolgenden Antworten enthalten die Sitzungs-ID.

Abfrage	200 SUCCESS HTTP/1.0
Header	Content-Length: 2048 TDS-Session: TDS00245817298274292
Inhalt (TDS)	Zeilenformat und einige Zeilen aus der Abfrageantwort

TDS-tunnelling-Servlet, erforderliche Systemausstattung

Um das jConnect-Servlet für TDS-tunneled-HTTP verwenden zu können, sind folgende Voraussetzungen zu erfüllen:

- Sie brauchen einen Webserver, der die **javax.servlet**-Interfaces unterstützt. Um den Server zu installieren, befolgen Sie die Anweisungen, die mit ihm mitgeliefert werden.
- Sie brauchen einen Webbrowser, der JDK 1.1 unterstützt, z.B. Netscape 4.0, Internet Explorer 4.0 oder HotJava.

Servlet installieren

Ihre jConnect-Installation enthält ein Unterverzeichnis *gateway* (jConnect 4.x) oder *gateway2* (jConnect 5.x) unter dem Verzeichnis *classes*. Das Unterverzeichnis enthält Dateien, die für das TDS-tunneling-Servlet benötigt werden.

Kopieren Sie das jConnect **gateway**-Paket in das Unterverzeichnis *gateway* (jConnect 4.x) oder *gateway2* (jConnect 5.x) unter dem Verzeichnis "Servlets" Ihres Webserver. Wenn Sie die Servlets kopiert haben, befolgen Sie die für Ihren Webserver geltenden Anweisungen zur ihrer Aktivierung.

Servlet-Argumente setzen

Wenn Sie das Servlet Ihrem Webserver hinzufügen, können Sie optionale Argumente eingeben, um die Performance anzupassen:

- *SkipDoneProc* [*true|false*] – Sybase-Datenbanken geben oft als Zwischeninformationen bei der Ausführung einer Abfrage Daten zur Zeilenanzahl zurück. Normalerweise werden diese Daten von Clientanwendungen ignoriert. Indem Sie *SkipDoneProc* auf "true" setzen, entfernt das Servlet diese Zusatzinformationen aus den Antworten "während der Verarbeitung", so dass das Netzwerk weniger belastet und der Verarbeitungsaufwand auf dem Client reduziert wird. Dies ist besonders sinnvoll, wenn Sie HTTPS/SSL verwenden, weil die unerwünschten Daten vor ihrer Ausfilterung nicht erst entschlüsselt/verschlüsselt werden müssen.
- *TdsResponseSize* – Benutzen Sie diesen Parameter, um die maximale TDS-Paketgröße für tunneled-HTTPS einzustellen. Eine größere *TdsResponseSize* ist effizienter, wenn nur wenige Benutzer mit großen Datenvolumina arbeiten. Benutzen Sie eine kleinere *TdsResponseSize*, wenn viele Benutzer kleinere Transaktionen durchführen.
- *TdsSessionIdleTimeout* – Benutzen Sie diesen Parameter, um in Millisekunden festzulegen, wie lange eine Serververbindung im Leerlauf bleiben kann, bevor die Verbindung automatisch geschlossen wird. Die Standardeinstellung für *TdsSessionIdleTimeout* ist 600,000 (10 Minuten).

Wenn Sie über interaktive Client-Programme verfügen, die längere Zeiten im Leerlauf bleiben können, die Verbindung aber nicht abgebrochen werden soll, erhöhen Sie *TdsSessionIdleTimeout*.

Sie können den Zeitüberschreitungswert für die Verbindung auch vom jConnect-Client beziehen, indem Sie die Verbindungseigenschaft `SESSION_TIMEOUT` benutzen. Dies ist sinnvoll, wenn Sie bestimmte Anwendungen haben, die möglicherweise längere Zeit im Leerlauf bleiben. In diesem Fall sollten Sie mit der Verbindungseigenschaft `SESSION_TIMEOUT` einen höheren Zeitüberschreitungswert für die Verbindungen setzen, und nicht für das Servlet.

- *Debug* – Fehlersuchmodus aktivieren. Siehe ["Debugging mit jConnect" auf Seite 102](#).

Geben Sie die Servlet-Argumente in einer Zeichenfolge getrennt durch Kommas ein. Beispiel:

```
TdsResponseSize=[size],TdsSessionIdleTimeout=[timeout],Debug=true
```

Vollständige Hinweise zur Eingabe von Servlet-Argumenten entnehmen Sie der Dokumentation des Webservers.

Servlet aufrufen

jConnect ermittelt, wann das Gateway zu verwenden ist, in dem das TDS-tunneling-Servlet installiert wurde, indem die Path-Erweiterung der Verbindungseigenschaft *proxy* abgefragt wird. jConnect erkennt die Servlet-Path-Erweiterung für *proxy* und ruft das Servlet auf dem bezeichneten Gateway auf.

Definieren Sie den Verbindungs-URL mit folgendem Format:

```
http://Host:Port/TDS-servlet-path
```

jConnect ruft das TDS-tunneling-Servlet auf dem Webserver auf, um die TDS-Daten durch HTTP durchzuführen. Der Wert für den TDS-Servlet-Suchpfad muss dem Suchpfad-Wert entsprechen, den Sie in der Aliasliste der Servlets Ihres Webservers definiert haben.

Überwachung aktiver TDS-Sitzungen

Sie können Informationen über aktive TDS-Sitzungen anzeigen, unter anderem auch die Serververbindungen für jede Sitzung. Benutzen Sie Ihren Webbrowser, um den administrativen URL zu öffnen:

```
http://Host:Port/TDS-Servlet-Suchpfad?Operation=list
```

Wenn beispielsweise Ihr Server HAUPTSERVER ist und der TDS-Servlet-Path */tds* lautet, geben Sie ein:

```
http://MeinWebserver:8080/tds?Operation=list
```

Damit wird eine Liste aller aktiven TDS-Sitzungen angezeigt. Sie können auf eine Sitzung klicken, um mehr Informationen anzuzeigen, darunter auch die Serververbindung.

TDS-Sitzungen beenden

Sie können den oben beschriebenen URL verwenden, um eine aktive TDS-Sitzung zu beenden. Klicken Sie auf eine aktive Sitzung in der Liste der Sitzungen auf der ersten Seite und benutzen Sie dann den Befehl "Terminate This Session" (Diese Sitzung beenden).

TDS-Sitzung wieder aufnehmen

Sie können die Verbindungseigenschaft `SESSION_ID` so einrichten, dass sie erforderlichenfalls eine bestehende offene Verbindung wieder aufnehmen können. Wenn Sie eine `SESSION_ID` liefern, überspringt `jConnect` die Login-Phase des Protokolls und nimmt die Verbindung mit dem Gateway mit der bezeichneten Sitzungs-ID wieder auf. Wenn die Sitzungs-ID im Servlet nicht mehr vorhanden ist, gibt `jConnect` eine SQL-Ausnahmebedingung aus, wenn Sie zum ersten Mal versuchen, diese Verbindung zu benutzen.

TDS-tunnelling und Netscape Enterprise Server 3.5.1 auf Solaris verwenden

Netscape Enterprise Server 3.5.1 unterstützt die Methoden `javax.servlet.ServletConfig.getInitParameters()` oder `javax.servlet.ServletConfig.getInitParameterNames()` nicht. Um die erforderlichen Parameterwerte zu liefern, müssen Sie Aufrufe von `getInitParameter()` und `getInitParameterNames()` durch hartkodierte Parameterwerte in `TDSTunnelServlet.java` ersetzen.

Um die erforderlichen Parameterwerte in `TDSTunnelServlet.java` einzugeben und TDS-tunnelling mit Netscape Enterprise Server 3.5.1 auf Solaris zu verwenden, gehen Sie wie folgt vor:

- 1 Hartkodieren Sie die Parameterwerte in `TDSTunnelServlet.java`.
- 2 Erstellen Sie `.class`-Dateien aus den Klassendeklarationen in `TDSTunnelServlet.java`.

Dabei sollten folgende Dateien entstehen:

- `TDSTunnelServlet.class`
- `TdsSession.class`
- `TdsSessionManager.class`

- 3 Erstellen Sie ein Verzeichnis für die `.class`-Dateien in Ihrem Netscape Enterprise Server 3.5.1 (NSE_3.5.1) Installationsverzeichnis wie folgt:

```
mkdir NSE_3.5.1_Install_Verz/plugins/java/servlets/gateway
```

- 4 Kopieren Sie die `.class`-Dateien, die aus `TDSTunnelServlet.java` abgeleitet wurden, in das gerade erstellte Verzeichnis.
- 5 Kopieren Sie die Klassen unter `$JDBC_HOME/classes/com/sybase` auf `NSE_3.5.1_install_dir/docs/com/sybase`.

Eine einfache Methode dafür ist das wiederholte Kopieren aller Dateien unter `$JDBC_HOME/classes` auf `NSE_3.5.1_install_dir/docs`, z.B.:

```
cp -r $JDBC_HOME/classes  
NSE_3.5.1_Install_Verz/docs
```

Damit wird eine Anzahl von Dateien und Verzeichnissen, die sich nicht in `$JDBC_HOME/classes/com/sybase` befinden, kopiert. Die zusätzlichen Dateien und Verzeichnisse stören nicht, belegen aber Festplattenspeicher. Sie können sie löschen, um den Festplattenspeicher wieder freizumachen.

- 6 Setzen Sie den *proxy*-URL auf das TDS-tunnelling Servlet.

Beispiel: In `$JDBC_HOME/sample/gateway.html`, bearbeiten Sie den Parameter *proxy* wie folgt:

```
<param name=proxy value="http://hostname/servlet/  
gateway_name.TDSTunnelServlet_name">
```

SQLExceptions und SQLWarnings Meldungsübersicht

Die folgende Tabelle enthält eine Liste der SQL-Meldungen über Ausnahmefehler und Warnungen, die beim Einsatz von jConnect auftreten könnten.

SQLState	Meldung/Beschreibung/Maßnahme
010DP	<p>Mehrfach angegebene Property ____ wurde ignoriert</p> <p><i>Beschreibung:</i> Eine Verbindungseigenschaft wurde doppelt definiert. Sie ist in der Liste der Treiber-Verbindungseigenschaften möglicherweise zweimal mit unterschiedlicher Groß- und Kleinschreibung definiert, z.B. "password" und "PASSWORD". Namen von Verbindungseigenschaften beachten keine Groß/Kleinschreibung. Deshalb unterscheidet jConnect nicht zwischen Eigenschaftsnamen mit demselben Namen, aber unterschiedlicher Schreibweise.</p> <p>Die Verbindungseigenschaft kann auch sowohl in der Liste der Verbindungseigenschaften, als auch im URL definiert sein. In diesem Fall hat der Eigenschaftswert in der Liste der Verbindungseigenschaften Vorrang.</p> <p><i>Maßnahme:</i> Achten Sie darauf, dass Ihre Anwendung die Verbindungseigenschaften nur einmal definiert. Es kann aber vorkommen, dass Sie die Regel, dass die in der Liste der Verbindungseigenschaften definierten Verbindungseigenschaften über diejenigen Vorrang haben, die in einem URL definiert sind, für Ihre Anwendung nutzen wollen. In diesem Fall können Sie die Warnung ruhig ignorieren.</p>
010HA	<p>Der Server hat die angeforderten High-Availability-Funktionen zurückgewiesen. Konfigurieren Sie Ihre Datenbank neu oder fordern Sie keine Hochverfügbarkeits-Sitzung an.</p> <p><i>Beschreibung:</i> Die Verbindungseigenschaft REQUEST_HA_SESSION wurde nicht auf "true" gesetzt und der Server, mit dem sich jConnect verbinden wollte, hat die Verbindung nicht zugelassen.</p> <p><i>Maßnahme:</i> Konfigurieren Sie den Server neu, damit er die Hochverfügbarkeits-Notumschaltung unterstützt, oder setzen Sie REQUEST_HA_SESSION nicht auf "true."</p>
010HD	<p>Sybase High-Availability-Notumschaltung wird von diesem Datenbankservertyp nicht unterstützt</p> <p><i>Beschreibung:</i> Die Datenbank, mit der sich jConnect verbinden wollte, unterstützt keine Hochverfügbarkeits-Notumschaltung.</p> <p><i>Maßnahme:</i> Verbinden Sie sich nur mit Datenbankservern, die die Hochverfügbarkeits-Notumschaltung unterstützen.</p>
010MX	<p>'Metadata-Accessor'-Informationen in dieser Datenbank nicht gefunden. Installieren Sie die erforderlichen Tabellen wie in der jConnect-Dokumentation beschrieben. Fehler beim versuchten Abruf von Metadata-Informationen: ____</p> <p><i>Beschreibung:</i> Der Server verfügt möglicherweise nicht über die erforderlichen gespeicherten Prozeduren für die Rückgabe von Metadata-Informationen.</p> <p><i>Maßnahme:</i> Sorgen Sie dafür, dass gespeicherte Prozeduren für die Bereitstellung der Metadaten auf dem Server installiert sind. Siehe "Gespeicherte Prozeduren installieren" im Kapitel 3 der Dokumentation <i>jConnect for JDBC Installationshandbuch</i>.</p>

SQLState	Meldung/Beschreibung/Maßnahme
010P4	<p>Ein Ausgabeparameter wurde empfangen und ignoriert.</p> <p><i>Beschreibung:</i> Die Abfrage, die Sie ausgeführt haben, hat einen Ausgabeparameter zurückgegeben, aber der Code der Anwendung für die Verarbeitung des Ergebnisses konnte ihn nicht abfragen. Er wurde daher ignoriert.</p> <p><i>Maßnahme:</i> Wenn Ihre Anwendung die Daten des Ausgabeparameters braucht, müssen sie die Anwendung umprogrammieren, so dass sie einen Fetch auf diesen Ausgabeparameter durchführt. Dies kann ein CallableStatement erfordern, um die Abfrage auszuführen. Möglicherweise müssen Aufrufe von registerOutputParameter() und getXXX() hinzugefügt werden.</p>
010PF	<p>Mindestens eine in der Verbindungseigenschaft PRELOAD_JARS angegebene Jar-Datei konnte nicht geladen werden.</p> <p><i>Beschreibung:</i> Dies ist der Fall, wenn der DynamicClassLoader verwendet wird, wobei die Verbindungseigenschaft PRELOAD_JARS auf eine Komma-begrenzte Liste von JAR-Namen festgelegt ist. Wenn der DynamicClassLoader seine Verbindung zum Server öffnet, von dem die Klassen geladen werden sollen, versucht er, alle in dieser Verbindungseigenschaft angegebenen JARs "vorzuladen". Falls einer oder mehrere JAR-Namen auf dem Server nicht vorhanden sind, entsteht die obige Fehlermeldung.</p> <p><i>Maßnahme:</i> Stellen Sie sicher, dass alle in der Verbindungseigenschaft PRELOAD_JARS angegebenen JAR-Dateien auf dem Server vorhanden und verfügbar sind.</p>
010RC	<p>Angeforderter Typ und Parallelität für ResultSet wird nicht unterstützt. Sie wurden konvertiert.</p> <p><i>Beschreibung:</i> Sie haben eine Kombination aus Typ und Parallelität für das ResultSet angefordert, die nicht unterstützt wird. Die angeforderten Werte mussten konvertiert werden.</p> <p><i>Maßnahme:</i> Fordern Sie eine Kombination aus Typ und Parallelität für das ResultSet an, die unterstützt wird.</p>
010SJ	<p>'Metadata-Accessor'-Informationen in dieser Datenbank nicht gefunden. Installieren Sie die erforderlichen Tabellen wie in der jConnect-Dokumentation beschrieben.</p> <p><i>Beschreibung:</i> Die Metadaten-Information ist auf dem Server nicht konfiguriert.</p> <p><i>Maßnahme:</i> Wenn Ihre Anwendung Metadaten erfordert, installieren Sie die gespeicherten Prozeduren für die Rückgabe der Metadaten, die mit jConnect geliefert werden (siehe "Gespeicherte Prozeduren installieren" im Kapitel 3 der Dokumentation <i>jConnect for JDBC Installationshandbuch</i>). Wenn Sie keine Metadaten benötigen, setzen Sie die Eigenschaft USE_METADATA auf "false".</p>
010SK	<p>Datenbank kann Option ____ nicht setzen.</p> <p><i>Beschreibung:</i> Ihre Anwendung hat versucht, eine Operation auszuführen, die die Datenbank, mit der Sie verbunden sind, nicht unterstützt.</p> <p><i>Maßnahme:</i> Sie müssen eventuell ein Upgrade Ihrer Datenbank durchführen oder sicherstellen, dass die jüngste Version der Metadaten-Informationen in ihr installiert ist.</p>

SQLState	Meldung/Beschreibung/Maßnahme
010SN	<p>Keine Berechtigung zum Schreiben der Datei. Datei: ____.</p> <p>Fehlermeldung: ____</p> <p><i>Beschreibung:</i> Die Berechtigung zum Schreiben in eine Datei, die in der Verbindungseigenschaft <code>PROTOCOL_CAPTURE</code> festgelegt wurde, wird zurückgewiesen, weil eine Sicherheitsverletzung in der VM vorliegt. Dies kann vorkommen, wenn ein Applet versucht, in die angegebene Datei zu schreiben.</p> <p><i>Maßnahme:</i> Wenn Sie versuchen, in die Datei aus einem Applet zu schreiben, müssen Sie dafür sorgen, dass das Applet Zugang zum betreffenden Dateisystem hat.</p>
010SP	<p>Datei konnte nicht zum Schreiben geöffnet werden. Datei: ____.</p> <p>Fehlermeldung: ____</p> <p><i>Maßnahme:</i> Achten Sie darauf, dass der Dateiname richtig geschrieben und die Datei nicht schreibgeschützt ist.</p>
010TP	<p>Der ursprüngliche Zeichensatz der Verbindung, ____, konnte vom Server nicht konvertiert werden. Der vom Server vorgeschlagene Zeichensatz, ____, wird verwendet, Konvertierungen werden von <code>jdbcConnect</code> vorgenommen.</p> <p><i>Beschreibung:</i> Der Server kann den ursprünglich von <code>jdbcConnect</code> verlangten Zeichensatz nicht benutzen und hat mit einem anderen Zeichensatz geantwortet. <code>jdbcConnect</code> akzeptiert die Änderung und führt die erforderlichen Zeichensatzkonvertierungen durch.</p> <p>Diese Meldung hat rein informativen Charakter und ist ansonsten folgenlos.</p> <p><i>Maßnahme:</i> Um diese Meldung zu vermeiden, setzen Sie die Verbindungseigenschaft <code>CHARSET</code> auf einen Zeichensatz, der vom Server unterstützt wird.</p>
010UF	<p>Versuchter Befehl 'use database' fehlgeschlagen. Fehlermeldung: ____</p> <p><i>Beschreibung:</i> <code>jdbcConnect</code> konnte sich mit der Datenbank nicht verbinden, die in dem Verbindungs-URL angegeben wurde. Es gibt zwei Möglichkeiten:</p> <ul style="list-style-type: none"> • Der Name wurde im URL falsch eingegeben. • <code>USE_METADATA</code> ist "true" (Standardwert), aber die gespeicherten Prozeduren für die Rückgabe der Metadaten wurden nicht installiert. Als Ergebnis hat <code>jdbcConnect</code> versucht, den Befehl <code>use database</code> mit der Datenbank in der URL auszuführen, der Befehl ist aber fehlgeschlagen. Dies kann darauf zurückzuführen sein, dass Sie versuchten, auf eine SQL-Anywhere-Datenbank zuzugreifen. SQL-Anywhere-Datenbanken unterstützen den Befehl <code>use database</code> nicht. <p><i>Maßnahme:</i> Achten Sie darauf, ob der Datenbankname im URL richtig ist. Achten Sie darauf, dass die gespeicherten Prozeduren für die Rückgabe der Metadaten auf dem Server installiert sind (siehe "Gespeicherte Prozeduren installieren" im Kapitel 3 der Dokumentation <i>Sybase jdbcConnect for JDBC Installationsanleitung und Versionshinweise</i>). Wenn Sie versuchen, auf eine SQL-Anywhere-Datenbank zuzugreifen, geben Sie entweder keinen Datenbanknamen im URL an oder setzen Sie <code>USE_METADATA</code> auf "false".</p>

SQLState	Meldung/Beschreibung/Maßnahme
010UP	<p>Property ____ ignoriert, weil nicht bekannt.</p> <p><i>Beschreibung:</i> Sie haben versucht, eine Verbindungseigenschaft im URL zu setzen, die jConnect derzeit nicht erkennt. jConnect ignoriert die nicht erkannte Property.</p> <p><i>Maßnahme:</i> Prüfen Sie die URL-Definition in Ihrer Anwendung, um sicherzustellen, dass sie nur auf gültige Verbindungseigenschaften des jConnect-Treibers verweist.</p>
0100V	<p>Die Version des verwendeten TDS-Protokolls ist zu alt. Version: ____</p> <p><i>Beschreibung:</i> Der Server unterstützt die erforderliche Version des TDS-Protokolls nicht. jConnect erfordert Version 5.0 oder später.</p> <p><i>Maßnahme:</i> Verwenden Sie einen Server, der die erforderliche Version von TDS unterstützt. Details finden Sie im Abschnitt "Erforderliche Systemausstattung" in der Dokumentation "jConnect Installationshandbuch".</p>
JZ010	<p>I/O-Schicht: Thread-Operation fehlgeschlagen.</p> <p><i>Beschreibung:</i> Ein interner Fehler ist bei einem zeitbegrenzten I/O-Stream aufgetreten.</p> <p><i>Maßnahme:</i> Verbindung schließen und wieder öffnen.</p>
JZ001	<p>Benutzername '____' zu lang. Maximale Länge ist 30 Zeichen.</p> <p><i>Maßnahme:</i> Maximale Größe von 30 Byte nicht überschreiten.</p>
JZ002	<p>Kennwort '____' zu lang. Maximale Länge ist 30 Zeichen.</p> <p><i>Maßnahme:</i> Maximale Größe von 30 Byte nicht überschreiten.</p>
JZ003	<p>Falsches URL-Format. URL: ____</p> <p><i>Maßnahme:</i> Überprüfen Sie das URL-Format. Siehe "Parameter für die Verbindungseigenschaft im URL" auf Seite 20.</p> <p>Wenn Sie die Verbindungseigenschaft PROXY verwenden, kann ein Ausnahmefehler JZ003 gemeldet werden, sobald Sie versuchen, eine Verbindung herzustellen und das Format der PROXY-Property falsch ist.</p> <p>Das PROXY-Format für den Cascade-Proxy ist:</p> <p style="padding-left: 40px;"><i>ip_address:port_number</i></p> <p>Das PROXY-Format für das TDS Tunneling Servlet ist:</p> <p style="padding-left: 40px;"><i>http[s]://Host:Port/tunneling_servlet_alias</i></p>
JZ004	<p>Die Angabe des Benutzernamens fehlt in <code>DriverManager.getConnection(..., Properties)</code></p> <p><i>Maßnahme:</i> Liefern Sie die verlangte Benutzer-Property.</p>
JZ006	<p>IOException: ____ wurde abgefangen</p> <p><i>Beschreibung:</i> Ein unerwarteter I/O-Fehler wurde in einer niedrigeren Schicht entdeckt. Solche I/O-Ausnahmefehler werden mit <code>ERR_IO_EXCEPTION JZ006 sqlstate</code> als SQL-Ausnahmebedingungen ausgegeben. Diese Fehler treten oft auf, wenn es zu Kommunikationsproblemen im Netzwerk kommt.</p> <p><i>Maßnahme:</i> Versuchen Sie, die Statement-Cachegröße zu erhöhen.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ008	<p>Ungültiger Spalten-Indexwert ____.</p> <p><i>Beschreibung:</i> Sie haben einen Spalten-Indexwert von weniger als 1 oder größer als dem höchsten verfügbaren Wert angefordert.</p> <p><i>Maßnahme:</i> Überprüfen Sie den Aufruf der Methode getXXX() und den Text der ursprünglichen Abfrage oder rufen Sie auf jeden Fall rs.next() auf.</p>
JZ009	<p>Fehler beim Konvertieren aufgetreten. Fehlermeldung: ____</p> <p><i>Beschreibung:</i> Folgende Möglichkeiten sind gegeben:</p> <ul style="list-style-type: none"> • Eine Konvertierung zwischen zwei unvereinbaren Datentypen wurde versucht, z.B. date zu int. • Es wurde versucht, eine Zeichenfolge, die ein nicht-numerisches Zeichen enthielt, in einen numerischen Typ zu konvertieren. • Ein Formatierungsfehler ist aufgetreten, etwa eine falsch formatierte Zeit/Datums-Zeichenfolge. <p><i>Maßnahme:</i> Vergewissern Sie sich, dass die JDBC-Spezifikation die versuchte Zeichenumwandlung unterstützt. Vergewissern Sie sich, dass die Zeichenfolgen korrekt formatiert sind. Wenn eine Zeichenfolge nicht-numerische Zeichen enthält, versuchen Sie nicht, sie in einen numerischen Typ zu konvertieren.</p>
JZ00B	<p>Numerischer Überlauf.</p> <p><i>Beschreibung:</i> Sie haben versucht, einen BigInteger als TDS-numerisch zu senden, und der Wert war zu groß oder Sie haben versucht, ein Java long als int zu senden und der Wert war zu groß.</p> <p><i>Maßnahme:</i> Diese Werte können in Sybase nicht gespeichert werden. Für den Typ long versuchen Sie den Datentyp "Sybase numeric". Es gibt keine Behelfslösung für Bignum.</p>
JZ00E	<p>Versuchter Aufruf von execute() oder executeUpdate() für ein Statement, in dem setCursorName() aufgerufen wurde.</p> <p><i>Maßnahme:</i> Versuchen Sie nicht, execute oder executeUpdate für eine Anweisung aufzurufen, die einen festgelegten Cursornamen hat. Verwenden Sie ein eigenes Statement, um einen Cursor zu löschen oder zu aktualisieren. Weitere Hinweise finden Sie unter "Cursor mit Ergebnismengen verwenden" auf Seite 48.</p>
JZ00F	<p>Cursorname wurde bereits durch setCursorName() gesetzt.</p> <p><i>Maßnahme:</i> Setzen Sie den Cursornamen nicht zweimal für ein Statement. Schließen Sie das ResultSet des aktuellen Cursor-Statements.</p>
JZ00G	<p>Für diese Zeilenaktualisierung wurden keine Spaltenwerte gesetzt.</p> <p><i>Beschreibung:</i> Sie haben versucht, eine Zeile zu aktualisieren, in der keine Spaltenwerte geändert wurden.</p> <p><i>Maßnahme:</i> Um Spaltenwerte in einer Zeile zu ändern, rufen Sie updateXX()-Methoden auf, bevor Sie updateRow() aufrufen.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ00H	<p>ResultSet kann nicht aktualisiert werden. Verwenden Sie <code>Statement.setResultSetConcurrencyType()</code>.</p> <p><i>Maßnahme:</i> Um eine Ergebnismenge von schreibgeschützt in aktualisierbar zu ändern, verwenden Sie die Methode <code>Statement.setResultSetConcurrencyType()</code>, oder fügen Sie eine <code>for update</code>-Klausel zu Ihrer SQL-select-Anweisung hinzu.</p>
JZ00L	<p>Login fehlschlagen. Überprüfen Sie SQLWarnings zu diesem Fehler, um die Ursache zu sehen.</p> <p><i>Maßnahme:</i> Siehe Meldungstext. Gehen Sie so vor, wie es gemäß den angegebenen Ursachen für den Login-Fehlschlag angemessen ist.</p>
JZ010	<p>Objektwert konnte nicht de-serialisiert werden. Fehlertext: _____</p> <p><i>Maßnahme:</i> Stellen Sie sicher, dass das Java-Objekt von der Datenbank die Schnittstelle <code>Serializable</code> implementiert und sich in Ihrer lokalen Variablen <code>CLASSPATH</code> befindet.</p>
JZ011	<p>Zahlenformat-Ausnahmebedingung bei der syntaktischen Analyse der numerischen Verbindungseigenschaft _____ festgestellt.</p> <p><i>Beschreibung:</i> Ein Nicht-Ganzzahl-Wert wurde für eine numerische Verbindungseigenschaft festgelegt.</p> <p><i>Maßnahme:</i> Geben Sie einen Ganzzahlwert für die Verbindungseigenschaft an.</p>
JZ012	<p>Interner Fehler. Bitte den Technischen Kundendienst von Sybase verständigen. Falscher Typ für Property _____ angegeben.</p> <p><i>Maßnahme:</i> Verständigen Sie den Technischen Kundendienst von Sybase.</p>
JZ013	<p>Fehler beim Bezug des JNDI-Eintrags: _____</p> <p><i>Maßnahme:</i> Korrigieren Sie den JNDI URL oder fügen Sie einen neuen Eintrag in den Verzeichnisdienst ein.</p>
JZ0BD	<p>Es wurde ein ungültiger oder unzulässiger Wert als Methodenparameter verwendet.</p> <p><i>Maßnahme:</i> Prüfen Sie den Parameterwert in der Methode auf Richtigkeit.</p>
JZ0BE	<p>BatchUpdateException: Fehler beim Ausführen einer Batch-Anweisung: _____.</p>
JZ0BP	<p>Ausgabeparameter sind in Batch-Update-Anweisungen unzulässig.</p> <p><i>Maßnahme:</i></p>
JZ0BR	<p>Der Cursor steht nicht auf einer Zeile, die die Methode _____ unterstützt.</p> <p><i>Beschreibung:</i> Sie haben versucht, eine <code>ResultSet</code>-Methode aufzurufen, die für die aktuelle Zeilenposition nicht zulässig ist (z.B. <code>insertRow()</code>), wenn der Cursor nicht in der Einfügezeile steht).</p> <p><i>Maßnahme:</i> Versuchen Sie nicht, eine <code>ResultSet</code>-Methode aufzurufen, die für die aktuelle Zeilenposition nicht zulässig ist.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0BS	Batch-Anweisungen werden nicht unterstützt.
JZ0BT	<p>Die Methode _____ wird für ResultSets des Typs _____ nicht unterstützt.</p> <p><i>Beschreibung:</i> Sie versuchen, eine ResultSet-Methode aufzurufen, die für den ResultSet-Typ nicht zulässig ist.</p> <p><i>Maßnahme:</i> Versuchen Sie nicht, eine ResultSet-Methode aufzurufen, die für den ResultSet-Typ nicht zulässig ist.</p>
JZ0C0	<p>Verbindung wurde bereits geschlossen.</p> <p><i>Beschreibung:</i> Die Anwendung hat für dieses Verbindungsobjekt Connection.close() bereits aufgerufen. Es kann nicht mehr verwendet werden.</p> <p><i>Maßnahme:</i> Berichtigen Sie den Anwendungscode, so dass Verweise auf Verbindungsobjekte genullt werden, wenn eine Verbindung geschlossen ist.</p>
JZ0D0	<p>Diese jConnect-Installation ist noch nicht registriert worden. Sie müssen die geeigneten SybDriverKey-Klassen installieren.</p> <p><i>Maßnahme:</i> Gehen Sie auf die jConnect-Website, um Ihre jConnect-Software zu registrieren: bei http://www.sybase.com/products/internet/jconnect/).</p> <p>Sobald Sie sie registriert haben, können Sie die SybDriverKey-Klassen herunterladen, die zum Aktivieren des jConnect-Treibers benötigt werden.</p>
JZ0D2	<p>Ihre Sybase-JDBC-Lizenz lief am _____ ab. Besorgen Sie sich eine neue Lizenz.</p> <p><i>Maßnahme:</i> Besorgen Sie sich bei Sybase eine neue Lizenz für Ihren jConnect-Treiber.</p>
JZ0D3	<p>Ihre Sybase-JDBC-Lizenz wird bald ablaufen. Sie sollten sich eine neue Lizenz besorgen. Diese läuft am _____ ab.</p> <p><i>Maßnahme:</i> Besorgen Sie sich bei Sybase eine neue Lizenz für Ihren jConnect-Treiber.</p>
JZ0D4	<p>Das angegebene Protokoll in der Sybase JDBC URL wurde nicht erkannt: _____.</p> <p><i>Beschreibung:</i> Sie haben einen Verbindungs-URL mit einem anderen Protokoll als TDS angegeben. TDS ist aber das einzige Protokoll, das derzeit von jConnect unterstützt wird.</p> <p><i>Maßnahme:</i> Überprüfen Sie die URL-Definition. Wenn der URL TDS als Subprotokoll festlegt, achten Sie darauf, dass der Eintrag das folgende Format und die entsprechende Schreibweise verwendet:</p> <p>jdbc:sybase:Tds:Host:Port</p> <p>Wenn der URL JNDI als Subprotokoll festlegt, achten Sie darauf, dass der Eintrag mit folgender Zeichenfolge beginnt:</p> <p>jdbc:sybase:jndi:</p>
JZ0D5	<p>Fehler beim Laden des Protokolls _____.</p> <p><i>Maßnahme:</i> Prüfen Sie die Einstellungen für die Systemvariable CLASSPATH.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0D6	<p>Die angegebene Versionsnummer _____ in setVersion ist nicht korrekt. Achten Sie darauf, für SybDriver.VERSION_* einen Wert zu wählen, der nicht größer als die verwendete jConnect-Version ist.</p> <p><i>Maßnahme:</i> Siehe Meldungstext.</p>
JZ0D7	<p>Fehler beim Laden des URL-Providers _____. Fehlermeldung: _____</p> <p><i>Maßnahme:</i> Prüfen Sie den JNDI URL, um sich zu vergewissern, dass er richtig ist.</p>
JZ0D8	<p>Fehler beim Initialisieren des URL-Providers: _____</p> <p><i>Maßnahme:</i> Prüfen Sie den JNDI URL, um sich zu vergewissern, dass er richtig ist.</p>
JZ0EM	<p>Ende der Daten.</p> <p><i>Maßnahme:</i> Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p>
JZ0H0	<p>Thread für Event-Handler konnte nicht gestartet werden; Name des Events: _____.</p> <p><i>Maßnahme:</i> Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p>
JZ0H1	<p>Eine Event-Notification wurde empfangen, der Event-Handler wurde aber nicht gefunden; Name des Events: _____.</p> <p><i>Maßnahme:</i> Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p>
JZ0HC	<p>Unzulässiges Zeichen '_____' beim Durchsuchen einer hexadezimalen Zahl gefunden.</p> <p><i>Beschreibung:</i> Eine Zeichenfolge, die einen Binärwert darstellen sollte, enthält ein Zeichen, das sich nicht in dem Bereich befindet (0–9, a–f) der für eine hexadezimale Zahl erforderlich ist.</p> <p><i>Maßnahme:</i> Prüfen Sie die Zeichenwerte in der Zeichenfolge, um sicherzugehen, dass sie im erforderlichen Bereich liegt.</p>
JZ0I1	<p>I/O-Schicht: Fehler beim Lesen des Streams.</p> <p><i>Beschreibung:</i> Die Verbindung konnte die verlangte Menge nicht einlesen. Wahrscheinlich wurde die Dauer der Zeitüberschreitung in der Anweisung überschritten, und die Verbindung ist abgelaufen.</p> <p><i>Maßnahme:</i> Erhöhen Sie den Wert der Zeitüberschreitung in der Anweisung.</p>
JZ0I2	<p>I/O-Schicht: Fehler beim Schreiben des Streams.</p> <p><i>Beschreibung:</i> Die Verbindung konnte die verlangte Ausgabe nicht schreiben. Wahrscheinlich wurde die Dauer der Zeitüberschreitung in der Anweisung überschritten, und die Verbindung ist abgelaufen.</p> <p><i>Maßnahme:</i> Erhöhen Sie den Wert der Zeitüberschreitung in der Anweisung.</p>
JZ0I3	<p>Unbekannte Eigenschaft. Wahrscheinlich ein internes Produktproblem. Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p> <p><i>Maßnahme:</i> Die Meldung wird wahrscheinlich durch ein internes Produktproblem verursacht. Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ015	<p>Unbekannte CHARSET-Eigenschaft angegeben: ____.</p> <p><i>Beschreibung:</i> Sie haben einen Zeichensatzcode für die Verbindungseigenschaft CHARSET eingegeben, der nicht unterstützt wird.</p> <p><i>Maßnahme:</i> Geben Sie einen gültigen Zeichensatzcode für die Verbindungseigenschaft ein. Siehe "Zeichensatzkonverter von jConnect" auf Seite 34.</p>
JZ016	<p>Fehler beim Konvertieren von UNICODE in den vom Server benutzten Zeichensatz. Fehlermeldung: ____</p> <p><i>Maßnahme:</i> Wählen Sie einen anderen Zeichensatzcode für die Verbindungseigenschaft CHARSET auf dem jConnect-Client, der alle Zeichen unterstützt, die Sie an den Server senden müssen. Sie müssen unter Umständen einen anderen Zeichensatz auf dem Server installieren.</p>
JZ017	<p>Keine Antwort vom Proxy-Gateway.</p> <p><i>Beschreibung:</i> Das Cascade- oder Sicherheits-Gateway antwortet nicht.</p> <p><i>Maßnahme:</i> Vergewissern Sie sich, dass das Gateway richtig installiert ist und läuft.</p>
JZ018	<p>Verbindung mit Proxy-Gateway abgelehnt. Antwort von Gateway: ____</p> <p><i>Beschreibung:</i> Der Webserver oder das Gateway, die von der Verbindungseigenschaft PROXY festgelegt wurden, haben die Verbindungsanforderung abgelehnt.</p> <p><i>Maßnahme:</i> Prüfen Sie die Zugriffs- und Fehler-Logs auf dem Proxy, um zu ermitteln, warum die Verbindung abgelehnt wurde. Vergewissern Sie sich, dass der Proxy ein JDBC-Gateway ist.</p>
JZ019	<p>Dieser InputStream wurde bereits geschlossen.</p> <p><i>Beschreibung:</i> Sie haben versucht, einen von getAsciiStream(), getUnicodeStream() oder getBinaryStream() erhaltenen InputStream zu lesen, der InputStream war aber bereits geschlossen. Der Stream wurde vielleicht geschlossen, weil Sie eine andere Spalte verschoben oder die Ergebnismenge annulliert haben und nicht genügend Ressourcen vorhanden waren, um die Daten in den Cache zu legen.</p> <p><i>Maßnahme:</i> Erhöhen Sie den Cachespeicher oder lesen Sie die Spalten in der Reihenfolge.</p>
JZ01A	<p>Fehler beim Senden von _____. Der Stream wurde unterbrochen.</p> <p><i>Beschreibung:</i> Es ist ein Kürzungsfehler bei der Zeichensatzkonvertierung aufgetreten, bevor eine Zeichenfolge gesendet werden konnte. Die konvertierte Zeichenfolge ist länger als der Speicherplatz, der für sie reserviert wurde.</p> <p><i>Maßnahme:</i> Wählen Sie einen anderen Zeichensatzcode für die Verbindungseigenschaft CHARSET auf dem jConnect-Client, der alle Zeichen unterstützt, die Sie an den Server senden müssen. Sie müssen unter Umständen einen anderen Zeichensatz auf dem Server installieren.</p>
JZ0IS	<p>getXXXStream kann nach der Aktualisierung in der Ergebnismenge für eine Spalte nicht aufgerufen werden.</p> <p><i>Beschreibung:</i> Nachdem eine Spalte in einem ResultSet aktualisiert wurde, haben Sie versucht, den aktualisierten Spaltenwert zu lesen, indem Sie eine der folgenden SybResultSet-Methoden benutzten: getAsciiStream(), getUnicodeStream(), getBinaryStream(). jConnect unterstützt diese Syntax nicht.</p> <p><i>Maßnahme:</i> Versuchen Sie nicht, einen Fetch von InputStreams aus Spalten durchzuführen, die Sie aktualisieren.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0J0	<p>Die Werte für Offset und Länge liegen über der aktuellen Bild/Text-Größe.</p> <p><i>Maßnahme:</i> Prüfen Sie die Werte für Offset bzw. Länge, die Sie benutzten, auf Richtigkeit.</p>
JZ0NC	<p>wasNull ohne vorherigen Aufruf einer Spalte aufgerufen.</p> <p><i>Beschreibung:</i> Sie können wasNull() nur nach einem Get-Aufruf einer Spalte aufrufen, etwa getInt() oder getBinaryStream().</p> <p><i>Maßnahme:</i> Ändern Sie den Programmcode, um den Aufruf auf wasNull() zu verschieben.</p>
JZ0NE	<p>Falsches URL-Format. URL: _____. Fehlermeldung: _____</p> <p><i>Maßnahme:</i> Prüfen Sie das Format des URLs. Vergewissern Sie sich, dass die Portnummer nur aus numerischen Zeichen besteht.</p>
JZ0NF	<p>SybSocketFactory konnte nicht geladen werden. Prüfen Sie die Schreibweise des Klassennamens, die volle Angabe des Pakets, die Verfügbarkeit der Klasse im Class-Path und dass die Klasse einen 'public zero-argument constructor' hat.</p> <p><i>Maßnahme:</i> Siehe Meldungstext.</p>
JZ0P1	<p>Unerwarteter Ergebnistyp.</p> <p><i>Beschreibung:</i> Die Datenbank hat ein Ergebnis zurückgegeben, das das Statement nicht an die Anwendung zurückgeben kann, oder das die Anwendung an diesem Punkt nicht erwartet. Damit wird im Allgemeinen darauf hingewiesen, dass die Anwendung JDBC fehlerhaft verwendet, um die Abfrage oder gespeicherte Prozeduren auszuführen. Wenn die JDBC-Anwendung mit einer Open-Server-Anwendung verbunden ist, kann dies einen Fehler in der Open-Server-Anwendung vermuten lassen, der den Open Server veranlasst, unerwartete Sequenzen von Ergebnissen auszusenden.</p> <p><i>Maßnahme:</i> Benutzen Sie die Debug-Tools com.sybase.utils.Debug(true, "ALL"), um herauszufinden, welche unerwarteten Ergebnisse auftreten, und welche Gründe dafür maßgebend sind.</p>
JZ0P4	<p>Protokollfehler. Wahrscheinlich ein internes Produktproblem. Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p> <p><i>Maßnahme:</i> Siehe Meldungstext.</p>
JZ0P7	<p>Spalte nicht im Cache; RE-READABLE_COLUMNS-Eigenschaft verwenden.</p> <p><i>Beschreibung:</i> Mit der Verbindungseigenschaft REPEAT_READ auf "false" wurde ein Versuch unternommen, eine Spalte nochmals zu lesen oder eine Spalte in der falschen Reihenfolge zu lesen.</p> <p>Wenn REPEAT_READ auf "false" gesetzt wurde, können Sie den Spaltenwert für eine Zeile nur einmal einlesen, und Sie können Spalten nur in aufsteigender Spalten-Index-Reihenfolge lesen. Beispiel: Nachdem Sie Spalte 3 für eine Zeile gelesen haben, können Sie ihren Wert nicht ein zweites Mal lesen, und Sie können auch nicht Spalte 2 für diese Zeile lesen.</p> <p><i>Maßnahme:</i> Entweder setzen Sie REPEAT_READ auf "true", oder Sie versuchen nicht, einen Spaltenwert zweimal bzw. Spalten außerhalb der Reihenfolge zu lesen.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0P8	<p>Der Spaltentypname RSMDB ist unbekannt.</p> <p><i>Beschreibung:</i> jConnect konnte den Namen eines Spaltentyps in der Methode <code>ResultSetMetaData.getColumnTypeName()</code> nicht ermitteln.</p> <p><i>Maßnahme:</i> Prüfen Sie, ob Ihre Datenbank die letzten gespeicherten Prozeduren für Metadaten hat.</p>
JZ0P9	<p>Eine Abfrage <code>COMPUTE BY</code> wurde erkannt. Diese Art von Ergebnis wird nicht unterstützt. Die Abfrage wurde abgebrochen.</p> <p><i>Beschreibung:</i> Die Abfrage, die Sie ausgeführt haben, gab <code>COMPUTE</code>-Ergebnisse zurück, die von jConnect nicht unterstützt werden.</p> <p><i>Maßnahme:</i> Ändern Sie Ihre Abfrage oder gespeicherten Prozedur so, dass sie <code>COMPUTE BY</code> nicht verwendet.</p>
JZ0PA	<p>Abfrage wurde storniert und Antwort verworfen.</p> <p><i>Beschreibung:</i> Stornierung (Aufruf einer <code>cancel</code>-Methode) wurde möglicherweise von einem anderen Statement in dieser Verbindung (Connection) ausgegeben.</p> <p><i>Maßnahme:</i> Prüfen Sie die Kette der <code>SQLExceptions</code> und <code>SQLWarnings</code> auf diesem und anderen Statements, um die Ursache zu ermitteln.</p>
JZ0PB	<p>Der Server unterstützt eine angeforderte Operation nicht.</p> <p><i>Beschreibung:</i> Wenn jConnect eine Verbindung mit einem Server erstellt, wird der Server informiert, welche Funktionen unterstützt werden sollen, und der Server informiert jConnect, welche Funktionen er unterstützen kann. Diese Fehlermeldung wird gesendet, wenn eine Anwendung einen Vorgang anfordert, der in der anfänglichen Dialogverhandlung über die unterstützten Funktionen nicht bereitgestellt wurde.</p> <p>Beispiel: Wenn die Datenbank die Vorkompilierung von dynamischen SQL-Anweisungen nicht unterstützt, und Ihr Programmcode <code>SybConnection.prepareStatement(sql_stmt, dynamic)</code> aufruft und <code>dynamic</code> auf "true" gesetzt ist, erzeugt jConnect diese Meldung.</p> <p><i>Maßnahme:</i> Ändern Sie Ihren Programmcode, so dass er keine nicht unterstützte Funktion anfordert.</p>
JZ0R0	<p><code>ResultSet</code> wurde bereits geschlossen.</p> <p><i>Beschreibung:</i> Die <code>ResultSet.close()</code>-Methode wurde bereits für das <code>ResultSet</code>-Objekt aufgerufen. Sie können das <code>ResultSet</code> nicht für einen anderen Zweck benutzen.</p> <p><i>Maßnahme:</i> Berichtigen Sie den Programmcode, so dass die Verweise auf <code>ResultSet</code>-Objekte genullt werden, wenn eine Verbindung geschlossen ist.</p>
JZ0R1	<p><code>ResultSet</code> ist im Zustand <code>IDLE</code>, da Sie derzeit nicht auf eine Zeile zugreifen.</p> <p><i>Beschreibung:</i> Die Anwendung hat eine der <code>ResultSet.getXXX</code>-Methoden für die Abfrage von Spaltendaten aufgerufen, aber es gibt keine aktuelle Zeile. Die Anwendung hat <code>ResultSet.next()</code> nicht aufgerufen, oder <code>ResultSet.next()</code> gab "false" zurück, um anzuzeigen, dass keine Daten vorhanden waren.</p> <p><i>Maßnahme:</i> Prüfen Sie, ob <code>rs.next()</code> auf "true" gesetzt ist, bevor Sie <code>rs.getXXX</code> aufrufen.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0R2	<p>Keine Ergebnisse für diese Abfrage.</p> <p><i>Beschreibung:</i> Sie haben <code>Statement.executeQuery()</code> verwendet, die Anweisung gab aber keine Zeile zurück.</p> <p><i>Maßnahme:</i> Verwenden Sie <code>executeUpdate</code> für Anweisungen, die keine Zeilen zurückgeben.</p>
JZ0R3	<p>Spalte ist im Zustand DEAD. Dies ist ein interner Fehler. Bitte den Technischen Kundendienst von Sybase verständigen.</p> <p><i>Maßnahme:</i> Siehe Meldungstext.</p>
JZ0R4	<p>Spalte hat keinen TextPointer. Sie ist entweder keine Text/Image-Spalte, oder die Spalte ist NULL.</p> <p><i>Beschreibung:</i> Sie können eine text/image-Spalte nicht aktualisieren, wenn Sie NULL ist. Eine NULL-text/image-Spalte hat keinen Textzeiger.</p> <p><i>Maßnahme:</i> Sorgen Sie dafür, dass Sie nicht versuchen, einen Textzeiger für eine Spalte zu aktualisieren oder zu setzen, die keine Text- und Bilddaten unterstützt. Achten Sie darauf, dass Sie nicht versuchen, eine text/image-Spalte zu aktualisieren, die NULL ist. Fügen Sie erst die Daten ein und aktualisieren Sie dann.</p>
JZ0RM	<p><code>refreshRow</code> kann nach <code>updateRow</code> oder <code>deleteRow</code> nicht aufgerufen werden.</p> <p><i>Beschreibung:</i> Nachdem eine Zeile in der Datenbank mit <code>SybCursorResult.updateRow()</code> aktualisiert oder mit <code>SybCursorResult.deleteRow()</code> gelöscht wurde, haben Sie <code>SybCursorResult.refreshRow()</code> verwendet, um die Zeile aus der Datenbank nochmals einzulesen.</p> <p><i>Maßnahme:</i> Versuchen Sie nicht, eine Zeile neu einzulesen, nachdem sie in der Datenbank aktualisiert oder gelöscht wurde.</p>
JZ0S0	<p>Statement ist BUSY. Dieses Statement ist gerade aktiv.</p> <p><i>Beschreibung:</i> Dieser Fehler wird nur aus der <code>Statement.setCursorname()</code>-Methode ausgegeben, sobald die Anwendung versucht, den Cursornamen zu senden, wenn das Statement bereits verwendet wird und Non-Cursor-Ergebnisse hat, die gelesen werden müssen.</p> <p><i>Maßnahme:</i> Setzen Sie den Cursornamen auf ein Statement, bevor Sie Abfragen darauf durchführen, oder rufen Sie <code>Statement.cancel()</code> auf, bevor Sie den Cursornamen einrichten, um sicherzugehen, dass das Statement nicht im Zustand BUSY ist.</p>
JZ0S1	<p>Statement ist im Zustand IDLE. Ein FETCH ist daher derzeit nicht möglich.</p> <p><i>Beschreibung:</i> Ein interner Fehler ist im Statement aufgetreten.</p> <p><i>Maßnahme:</i> Schließen Sie das Statement und öffnen Sie ein anderes.</p>
JZ0S2	<p>Statement wurde bereits geschlossen.</p> <p><i>Beschreibung:</i> Die <code>Statement.close()</code>-Methode wurde für das Statement-Objekt bereits aufgerufen. Sie können das Statement nicht mehr für etwas anderes verwenden.</p> <p><i>Maßnahme:</i> Berichten Sie die Anwendung, so dass Verweise auf Statement-Objekte genullt werden, wenn ein Statement geschlossen ist.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0S3	<p>Die geerbte Methode _____ kann in dieser Klasse (SubClass) nicht benutzt werden.</p> <p><i>Beschreibung:</i> PreparedStatement unterstützt executeQuery(String), executeUpdate(String) oder execute(String) nicht.</p> <p><i>Maßnahme:</i> Wenn Sie eine Abfrage-Zeichenfolge weitergeben wollen, verwenden Sie Statement, nicht PreparedStatement.</p>
JZ0S4	<p>Leere Abfrage (Länge 0) kann nicht ausgeführt werden.</p> <p><i>Maßnahme:</i> Führen Sie keine leere Abfrage aus ("").</p>
JZ0S8	<p>Escape-Sequenz in einer SQL-Abfrage fehlerhaft formatiert: '_____'. <i>Beschreibung:</i> Dieser Fehler entsteht durch eine fehlerhafte Escape-Syntax. <i>Maßnahme:</i> Ermitteln Sie in der JDBC-Dokumentation die richtige Syntax.</p>
JZ0S9	<p>Leere Abfrage (Länge 0) kann nicht ausgeführt werden.</p> <p><i>Maßnahme:</i> Führen Sie keine leere Abfrage aus ("").</p>
JZ0SA	<p>Prepared Statement: Eingabeparameter nicht vollständig, Index: _____.</p> <p><i>Maßnahme:</i> Sorgen Sie dafür, dass jeder Eingabeparameter einen Wert hat.</p>
JZ0SB	<p>Anzahl der Parameter ist außerhalb des gültigen Bereichs: _____. <i>Beschreibung:</i> Sie haben versucht, einen Parameter zu holen, zu setzen oder zu registrieren, der die zulässige Anzahl von Parametern überschreitet. <i>Maßnahme:</i> Prüfen sie die Anzahl von Parametern in Ihrer Abfrage.</p>
JZ0SC	<p>Callable Statement: Es ist nicht erlaubt, den Returnstatus als Eingabeparameter zu setzen. <i>Beschreibung:</i> Sie haben einen Aufruf einer gespeicherten Prozedur vorbereitet, der einen Status zurückgibt, versuchen nun aber, den Parameter 1 zu setzen, der den Returnstatus bezeichnet. <i>Maßnahme:</i> Setzbare Parameter starten bei dieser Art von Programmaufruf mit 2.</p>
JZ0SD	<p>Kein registrierter Parameter für diesen Ausgabeparameter gefunden. <i>Beschreibung:</i> Dies weist auf einen Fehler in der Anwendungslogik hin. Sie haben versucht, getXXX() oder wasNull() für einen Parameter aufzurufen, haben aber noch keine Parameter gelesen, oder es gibt keine Ausgabeparameter. <i>Maßnahme:</i> Stellen Sie fest, ob die Anwendung registrierte Parameter im CallableStatement hat, ob das Statement ausgeführt wurde, und ob die Ausgabeparameter gelesen wurden.</p>
JZ0SE	<p>Ungültiger Objekttyp für setObject() angegeben. <i>Beschreibung:</i> Unzulässiges Typargument an PreparedStatement.setObjectübergeben. <i>Maßnahme:</i> Schlagen Sie in der JDBC-Dokumentation nach. Das Argument muss eine Konstante von java.sql.Types sein.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0SF	<p>Keine Parameter erwartet. Wurde eine Abfrage gesendet?</p> <p><i>Beschreibung:</i> Sie haben versucht, einen Parameter für ein Statement ohne Parameter zu setzen.</p> <p><i>Maßnahme:</i> Sorgen Sie dafür, dass die Abfrage abgesandt wurde, bevor Sie die Parameter gesetzt haben.</p>
JZ0SG	<p>Eine RPC gab nicht so viele Ausgabeparameter zurück, wie die Anwendung für sie registriert hat.</p> <p><i>Beschreibung:</i> Dieser Fehler tritt auf, wenn Sie CallableStatement.registerOutParam() für mehr Parameter aufrufen, als Sie als "OUTPUT"-Parameter in der gespeicherten Prozedur deklariert haben. Weitere Hinweise finden Sie unter "RPC gibt weniger Ausgabeparameter zurück als registriert" auf Seite 111.</p> <p><i>Maßnahme:</i> Prüfen Sie Ihre gespeicherten Prozeduren und registerOutParameter-Aufrufe. Achten Sie darauf, dass alle erforderlichen Parameter als "OUTPUT" deklariert werden. Suchen Sie folgende Programmcodezeile:</p> <pre>create procedure yourproc (@p1 int OUTPUT, ...</pre> <hr/> <p>Hinweis Wenn Sie diesen Fehler erhalten, während Sie Adaptive Server Anywhere (früher SQL Anywhere) verwenden, führen Sie ein Upgrade auf Adaptive Server Anywhere Version 5.5.04 durch.</p>
JZ0SH	<p>Statische Funktions-Escape-Sequenz verwendet, aber die 'Metadata-Accessor'-Informationen wurden auf diesem Server nicht gefunden.</p> <p><i>Maßnahme:</i> Installieren Sie die Metadata-Accessor-Informationen, bevor Sie statische Funktions-Escape-Sequenzen verwenden.</p>
JZ0SI	<p>Eine statische Funktions-Escape-Sequenz _____ wurde benutzt, die auf diesem Server nicht unterstützt wird.</p> <p><i>Maßnahme:</i> Verwenden Sie diese Escape-Sequenz nicht.</p>
JZ0SJ	<p>'Metadata-Accessor'-Informationen in dieser Datenbank nicht gefunden.</p> <p><i>Maßnahme:</i> Installieren Sie die Metadata-Informationen, bevor Sie Metadata-Aufrufe durchführen.</p>
JZ0SM	<p>Dieser SQL-Typ _____ wird nicht unterstützt.</p> <p><i>Maßnahme:</i> Verwenden Sie Types.NULL, Types.OTHER oder PreparedStatement.setObject(null) nicht.</p>
JZ0SN	<p>setMaxFieldSize: Feldgröße kann nicht negativ sein.</p> <p><i>Maßnahme:</i> Benutzen Sie einen positiven Wert oder Null (unbegrenzt), wenn Sie setMaxFieldSize aufrufen.</p>
JZ0T2	<p>Listener-Thread-Lesefehler.</p> <p><i>Maßnahme:</i> Prüfen Sie die Netzwerkverbindungen.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0T3	<p>Lesevorgang dauerte zu lange (time out).</p> <p><i>Beschreibung:</i> Die für das Lesen der Antwort auf eine Abfrage zugeteilte Zeitspanne wurde überschritten.</p> <p><i>Maßnahme:</i> Erhöhen Sie die Zeitablaufperiode durch den Aufruf von <code>Statement.setQueryTimeout()</code>.</p>
JZ0T4	<p>Schreibvorgang dauerte zu lange. Zeitüberschreitung in Millisekunden: ____.</p> <p><i>Beschreibung:</i> Die für das Senden einer Abfrage reservierte Zeitspanne wurde überschritten.</p> <p><i>Maßnahme:</i> Erhöhen Sie die Zeitablaufperiode durch den Aufruf von <code>Statement.setQueryTimeout()</code>.</p>
JZ0T5	<p>Der für das Speichern der Antworten benutzte Cache ist voll.</p> <p><i>Maßnahme:</i> Benutzen Sie den Standardwert oder einen höheren Wert für die Verbindungseigenschaft <code>STREAM_CACHE_SIZE</code>.</p>
JZ0T6	<p>Fehler beim Lesen der 'Tunnelled TDS'-URL.</p> <p><i>Beschreibung:</i> Das "tunneled protocol" schlug beim Lesen des URL-Headers fehl.</p> <p><i>Maßnahme:</i> Prüfen Sie den URL, den Sie für die Verbindung definiert haben.</p>
JZ0T7	<p>Listener-Thread-Lesefehler - ThreadDeath abgefangen. Bitte Netzwerkverbindungen prüfen.</p> <p><i>Maßnahme:</i> Prüfen Sie die Netzwerkverbindungen und versuchen Sie, die Anwendung neuerlich zu starten. Wenn die Threads weiterhin abgebrochen werden, verständigen Sie den Technischen Kundendienst von Sybase.</p>
JZ0T9	<p>Der Versuch, in einen Stream zu schreiben, wurde nicht synchronisiert. Melden Sie diesen Fehler dem Technischen Kundendienst von Sybase.</p> <p><i>Maßnahme:</i> Siehe Meldungstext.</p>
JZ0TC	<p>Versuchte Konvertierung zwischen diesem Typenpaar ist unzulässig.</p> <p><i>Beschreibung:</i> Die Konvertierung zwischen einem Java-Typ und einem SQL-Typ ist fehlgeschlagen.</p> <p><i>Maßnahme:</i> Prüfen Sie die verlangte Typenkonvertierung, um sicherzugehen, dass sie in der JDBC-Spezifikation unterstützt wird.</p>
JZ0TE	<p>Versuchte Konvertierung zwischen diesem Typenpaar ist unzulässig. Gültige Datentypen sind: '_____'.</p> <p><i>Beschreibung:</i> Der Spaltentyp der Datenbank und der im Aufruf <code>ResultSet.getXXX()</code> angeforderte Datentyp sind nicht implizit konvertierbar.</p> <p><i>Maßnahme:</i> Verwenden Sie einen der zulässigen Datentypen, die in der Fehlermeldung aufgelistet sind.</p>

SQLState	Meldung/Beschreibung/Maßnahme
JZ0US	<p>Die SybSocketFactory Verbindungseigenschaft wurde gesetzt, und die PROXY-Verbindungseigenschaft wurde auf den URL eines Servlets gesetzt. Der jConnect-Treiber unterstützt diese Kombination nicht. Wenn Sie sichere HTTP von einem Applet aus einem Browser senden wollen, benutzen Sie einen Proxy-URL, der mit "https://" eingeleitet wird.</p> <p><i>Maßnahme:</i> Siehe Meldungstext.</p>
JZ0CX	<p>_____ ist ein nicht erkannter Transaktionskoordinatortyp.</p> <p><i>Beschreibung:</i> Die Metadaten-Informationen zeigen an, dass der Server verteilte Transaktion unterstützt, jConnect unterstützt aber das verwendete Protokoll nicht.</p> <p><i>Maßnahme:</i> Stellen Sie sicher, dass Sie die neuesten Metadaten-Skripten installiert haben. Wenn der Fehler weiter besteht, wenden Sie sich an den Technischen Kundendienst von Sybase.</p>
JZ0XS	<p>Der Server unterstützt keine XA-Transaktionen. Prüfen Sie, ob die Transaktionsfunktion für den Server aktiviert und lizenziert ist.</p> <p><i>Beschreibung:</i> Der Server, mit dem sich jConnect verbinden wollte, unterstützt verteilte Transaktionen nicht.</p> <p><i>Maßnahme:</i> Verwenden Sie XADatasource bei diesem Server nicht, oder führen Sie eine Aktualisierung durch/konfigurieren Sie den Server für verteilte Transaktionen.</p>
JZ0XU	<p>Der aktuelle Benutzer hat keine Berechtigung für XA-Transaktionen. Der Benutzer benötigt dafür die Rolle _____.</p> <p><i>Beschreibung:</i> Der mit der Datenbank verbundene Benutzer hat keine Berechtigung zum Ausführen von verteilten Transaktionen. Wahrscheinlich hat er nicht die geeignete Rolle (die im Zwischenraum angezeigt wird).</p> <p><i>Maßnahme:</i> Erteilen Sie dem Benutzer die Rolle, die in der Fehlermeldung gezeigt wird, oder lassen Sie einen anderen Benutzer die Transaktion ausführen.</p>
S0022	<p>Ungültiger Spaltenname '_____'. <i>Beschreibung:</i> Sie haben versucht, eine Spalte mit ihrem Namen zu referenzieren, aber es ist keine Spalte mit diesem Namen vorhanden. <i>Maßnahme:</i> Prüfen Sie die Schreibweise des Spaltennamens.</p>
ZZ00A	<p>Methode _____ ist nicht implementiert und sollte nicht aufgerufen werden.</p> <p><i>Beschreibung:</i> Sie haben versucht, eine Methode aufzurufen, die nicht implementiert ist.</p> <p><i>Maßnahme:</i> In den Versionshinweisen, die mit Ihrer Version von jConnect geliefert wurden, finden Sie vielleicht weitere Hinweise. Sie können auch die jConnect-Webseite unter http://www.sybase.com konsultieren, um festzustellen, ob eine jüngere Version von jConnect die Methode implementiert. Wenn nicht, benutzen Sie diese Methode nicht.</p>

jConnect Beispielprogramme

Dieser Anhang führt Sie durch Sybase jConnect-Beispielprogramme.

Er enthält die folgenden Abschnitte:

Name	Seite
IsqlApp ausführen	172
jConnect-Beispielprogramme und -Code ausführen	175

IsqlApp ausführen

Mit **IsqlApp** können Sie **isql**-Befehle von der Befehlszeile ausgeben und jConnect-Beispielprogramme starten.

Die Syntax für **IsqlApp** ist:

```
IsqlApp [-U Benutzername] [-P Kennwort]
        [-S Servername]
        [-G Gateway]
        [-p {http|https}]
        [-D Debug-Klassenliste]
        [-v]
        [-I Eingabe-Befehlsdatei]
        [-c Befehlsabschluss]
        [-C Zeichensatz] [-L Sprache]
        [-T Sitzungskennung]
        [-V <Version {2,3,4,5}>]
```

Parameter	Beschreibung
-U	Die Anmeldekennung, mit der Sie sich mit einem Server verbinden.
-P	Das Kennwort für die angegebene Anmeldekennung.
-S	Der Name des Servers, mit dem Sie sich verbinden.
-G	Die Gateway-Adresse. Für das HTTP-Protokoll lautet der URL folgendermaßen: <code>http://Host:Port</code> . Wenn das HTTPS-Protokoll verwendet werden soll, das die Verschlüsselung unterstützt, benutzen Sie die URL <code>https://Host:Port/servlet_alias</code> .
-p	Legt fest, ob Sie das HTTP-Protokoll oder das HTTPS-Protokoll, das Verschlüsselung unterstützt, verwenden wollen.
-D	Schaltet die Fehlersuche für alle oder nur die durch Kommas getrennt angegebenen Klassen ein. Beispiel: <i>-D ALL</i> Zeigt Ausgaben zur Fehlersuche für alle Klassen. <i>-D SybConnection, Tds</i> Zeigt Ausgaben zur Fehlersuche nur für die SybConnection- und Tds-Klassen.
-v	Schaltet ausführliche Ausgaben zum Anzeigen oder Ausdrucken ein.
-I	Veranlasst IsqlApp, Befehle von einer Datei anstatt von der Tastatur anzunehmen. Nach dem Parameter geben Sie den Namen der Datei an, die für die IsqlApp-Eingabe verwendet werden soll. Die Datei muss Befehlsabschlusszeichen (standardmäßig "go") enthalten.

Parameter	Beschreibung
-c	Damit können Sie ein Schlüsselwort festlegen (z.B. "go"), das den Befehl beendet, wenn es als eigene Zeile eingegeben wird. Mit diesem Parameter können Sie mehrzeilige Befehle eingeben, bevor Sie das Abschluss-Schlüsselwort verwenden. Wenn Sie kein Befehlsabschlusszeichen eingeben, wird mit jeder neuen Zeile ein Befehl abgeschlossen.
-C	Legt den Zeichensatz für Zeichenfolgen fest, die durch TDS durchgegeben werden. Wenn Sie keinen Zeichensatz angeben, verwendet IsqlApp den Standard-Zeichensatz des Servers.
-L	Die Sprache, in der die vom Server zurückgegebenen Fehlermeldungen und jConnect-Meldungen angegeben werden.
-T	Wenn dieser Parameter angegeben wird, nimmt jConnect an, dass eine Anwendung versucht, die Kommunikation in einer bestehenden TDS-Sitzung wieder aufzunehmen, die vom TDS-tunnelling Gateway offengehalten wird. jConnect überspringt die Login-Dialogprozedur und leitet alle Anforderungen von der Anwendung an die angegebene Sitzungskennung weiter.
-V	Bewirkt, dass Versions-spezifische Merkmale verwendet werden. Siehe "JCONNECT_VERSION, Verbindungseigenschaft" auf Seite 9.

Hinweis Sie müssen ein Leerzeichen nach jeder Options-Flag eingeben.

Um eine vollständige Beschreibung der Programm-Startparameter zu erhalten, geben Sie folgenden Befehl ein:

```
java IsqlApp -help
```

Im folgenden Beispiel wird gezeigt, wie Sie sich mit einer Datenbank auf einem Host namens "MeinServer" über den Port "3756" verbinden und ein **isql**-Skript namens "MeinScript" ausführen:

```
java IsqlApp -U sa -P SA-Kennwort
-S jdbc:sybase:Tds:MeinServer:3756
```

```
-I $JDBC_HOME/sp/MeinScript -c run
```

Hinweis Ein Applet, das GUI-Zugriff auf **isql**-Befehle ermöglicht, steht in der folgenden Datei zur Verfügung:

Für jConnect 4.x:

\$JDBC_HOME/sample/gateway.html (UNIX)

%JDBC_HOME%\sample\gateway.html (Windows)

For jConnect 5.x:

\$JDBC_HOME/sample2/gateway.html (UNIX)

%JDBC_HOME%\sample2\gateway.html (Windows)

jConnect-Beispielprogramme und -Code ausführen

jConnect enthält einige Beispielprogramme, um viele Themen dieses Kapitels zu veranschaulichen und um Ihnen zu zeigen, wie jConnect mit verschiedenen JDBC-Klassen und Methoden zusammenarbeitet. Außerdem enthält dieser Abschnitt ein Beispiel für einen Programmcode als Referenz.

Beispielanwendungen

Wenn Sie jConnect installieren, können Sie auch Beispielprogramme installieren. Diese Beispiele enthalten den Quellcode, so dass Sie nachprüfen können, wie jConnect verschiedene JDBC-Klassen und Methoden umsetzt. Vollständige Anweisungen zum Installieren der Beispielprogramme finden Sie in der Dokumentation *jConnect for JDBC Installationshandbuch*.

Hinweis Die jConnect-Beispielprogramme sind nur für Demonstrationszwecke gedacht.

Die Beispielprogramme werden im Unterverzeichnis *sample* (jConnect 4.x) oder *sample2* (jConnect 5.x) unter Ihrem jConnect-Installationsverzeichnis installiert. Die Datei *index.html* im Unterverzeichnis *sample* oder *sample2* enthält eine vollständige Liste der verfügbaren Beispiele und eine Beschreibung für jedes Beispiel. Mit der Datei *index.html* können Sie die Beispielprogramme auch anzeigen und als Applets starten.

Beispiel-Applets ausführen

Mit Ihrem Webbrowser können Sie einige der Beispielprogramme als Applets ausführen. Damit können Sie sich den Quellcode ansehen, während Sie das Ergebnis angezeigt erhalten.

Um die Beispiele als Applets auszuführen, müssen Sie das Cascade Gateway starten. Siehe "[Cascade Gateway verwenden](#)" auf Seite 141.

Öffnen Sie mit Ihrem Web-Browser die Datei *index.html*:

Geben Sie für jConnect 4.x folgendes ein:

<http://localhost:8000/sample/index.html>

Geben Sie für jConnect 5.x folgendes ein:

<http://localhost:8000/sample2/index.html>

Die Beispielprogramme mit Adaptive Server Anywhere ausführen

Alle Beispielprogramme sind zu Adaptive Server Enterprise kompatibel, aber nur eine begrenzte Anzahl ist kompatibel zu Adaptive Server Anywhere. Eine aktuelle Liste der zu Adaptive Server Anywhere kompatiblen Beispielprogramme finden Sie in der Datei *index.html* im Unterverzeichnis *sample* oder *sample2*.

Um die für Adaptive Server Anywhere verfügbaren Beispielprogramme ausführen zu können, müssen Sie das Skript *pubs2_any.sql* auf Ihrem Adaptive Server Anywhere-Server installieren. Dieses Skript befindet sich im Unterverzeichnis *sample* (jConnect 4.1) oder *sample2* (jConnect 5.0) .

Gehen Sie unter Windows in ein DOS-Befehlsfenster und geben Sie folgendes ein:

```
java IsqlApp -U dba -P Kennwort
-S jdbc:sybase:Tds:[Hostname]:[Port]
-I %JDBC_HOME%\sample\pubs2_any.sql -c go
```

Unter UNIX geben Sie ein:

```
java IsqlApp -U dba -P Kennwort
-S jdbc:sybase:Tds:[Hostname]:[Port]
-I $JDBC_HOME/sample/pub2_any.sql -c go
```

Beispiel-Programmcode

Der folgende Beispiel-Code veranschaulicht, wie der jConnect-Treiber aufgerufen, eine Verbindung aufgebaut, eine SQL-Anweisung ausgegeben und Ergebnisse bearbeitet werden.

```
import java.io.*;
import java.sql.*;

public class SampleCode
{
    public static void main(String args[])
    {
        try
        {
            /*
            * Verbindung öffnen. Ergibt eventuell eine SQLException.
            */
            Connection con = DriverManager.getConnection(
                "jdbc:sybase:Tds:myserver:3767", "sa", "");
```

```

/*
 * Statement-Objekt als Behälter für die SQL-Anweisung
 * erstellen. Ergibt eventuell eine SQLException.
 */
    Statement stmt = con.createStatement();
/*
 * Erstelle ResultSet durch Ausführen der Abfrage.
 * Ergibt eventuell eine SQLException.
 */
    ResultSet rs = stmt.executeQuery("Select 1");
/*
 * Verarbeite Ergebnismenge.
 */

    if (rs.next())
    {
        int value = rs.getInt(1);
        System.out.println("Fetched value " + value);
    }
}
/*
 * Verarbeitung von Ausnahmebedingungen.
 */
catch (SQLException sqe)
{
    System.out.println("Unexpected exception : " +
        sqe.toString() + ", sqlstate = " +
        sqe.getSQLState());
    System.exit(1);
}
System.exit(0);
}
}

```


Index

A

- Abweichungen von JDBC-Standards 97
- Adaptive Server
 - Beispiel für eine Verbindung 19
 - Verbindung einrichten 18
- Adaptive Server Anywhere 16
 - Bilddaten senden 63
 - Euro-Symbol 38
 - Java-Objekte speichern und abrufen 76
 - SERVICENAME, Verbindungseigenschaft 19
 - Verbindung einrichten 21
 - Zugriff auf Metadaten 47
- Aktualisieren
 - Datenbank aus der Ergebnismenge einer gespeicherten Prozedur aktualisieren 62
- Aktualisierungen in Form von Anweisungsfolgen 61
 - gespeicherte Prozeduren 60
- Änderungen bei Sybase-Erweiterungen 133
- Anwendungen
 - auf jConnect migrieren 130
 - Ausschalten der Fehlersuchfunktion 103
 - Einschalten der Fehlersuchfunktion 103
 - migrieren auf jConnect 4.2 und 5.2 131
 - migrieren auf jConnect 5.x 130
- Applets 141, 142
- APPLICATIONNAME, Verbindungseigenschaft 12
- Aufruf
 - jConnect 11
- Ausschalten der Fehlersuchfunktion in Ihrer Anwendung 103

B

- Bilddaten
 - Aktualisieren mit TextPointer.sendData 65
 - Aktualisieren von Spalten mit
 - TextPointer.sendData() 64
 - public, Methoden in der TextPointer-Klasse 63

- senden 63
- TextPointer-Objekt holen 64

C

- CANCEL_ALL, Verbindungseigenschaft 6, 10, 13
- Capturing TDS communication 106
- Cascade Gateway 141
 - installieren 142
 - starten 142
 - testen 144
 - Verbindung definieren 146
- CHARSET, Verbindungseigenschaft 6, 13
 - festlegen 35
- CHARSET_CONVERTER, Verbindungseigenschaft 6
- CHARSET_CONVERTER_CLASS, Verbindungseigenschaft 13, 35
- CLASSPATH
 - einstellen für die Fehlersuche 103
- Compute-Anweisungen 98
- CONNECTION_FAILOVER, Verbindungseigenschaft 13, 22
- Cursor 48
 - erstellen 49
 - mit einer vorbereiteten Anweisung verwenden 56
- Cursor-Ergebnismenge
 - Methoden für die Aktualisierung der Datenbank 53
 - Zeile einfügen 55
 - Zeile löschen 55
- Cursor-Ergebnismengen
 - aktualisieren 53
 - löschen 52
 - positionierte Aktualisierungen 52
 - positionierte Aktualisierungen und Löschungen mit JDBC 1.x-Methoden 52
 - positionierte Aktualisierungen und Löschungen mit JDBC 2.0-Methoden 53

Index

Cursor-Performance
 und die Verbindungseigenschaft
 LANGUAGE_CURSOR 126

D

Datenbanken
 JNDI für Benennung 85
Datentypen
 Time, Date und Timestamp 66
Debug-Klasse 102
Deserialisierung 83
Dienstprogramm IsqlApp 172
Dienstprogramme
 IsqlApp 172
DYNAMIC_PREPARE, Verbindungseigenschaft 13
Dynamisches Laden von Klassen 81

E

Eigene-API-Teil-Java 3
Eigenschaften
 Treiber 12
Einschalten der Fehlersuchfunktion in Ihrer Anwendung
 103
Ereignismeldung 68
 Beispiel 69
Erstellen eines Cursors 49
Erweiterte Funktionen 68
Erweiterungsänderungen, Sybase 133
Euro-Währungssymbol 38
EXPIRESTRING, Verbindungseigenschaft 13

F

Fehler
 gespeicherte Prozeduren 111
 Verbindung 108, 109
Fehlerbehandlung 102
Fehlermeldungen
 Beispiel für ein Fehlersteuerungsprogramm 74
 Fehler-Steuerungsprogramm einrichten 74
 handhaben 71

SQL-Ausnahmefehler und Warnungen 153
Sybase-spezifische 71
Verarbeitung anpassen 72

Fehlersuche

CLASSPATH setzen 103
in Ihrer Anwendung ausschalten 103
in Ihrer Anwendung einschalten 103
Instanz der Debug-Klasse erhalten 102
jConnect 102
Methoden 104

Festlegen

jConnect-Verbindungseigenschaften 12

G

Gateways 138
 Konfiguration 139
 Open Server 21
 Verbindung abgelehnt 108
gespeicherte Prozeduren
 ausführen 99
 Datenbank aus der Ergebnismenge aktualisieren
 62
 Fehler 111

H

Handhaben
 Fehlermeldungen 71
HOSTNAME, Verbindungseigenschaft 13
HOSTPROC, Verbindungseigenschaft 13
HTTP 138

I

IGNORE_DONE_IN_PROC, Verbindungseigenschaft
 13
Installation
 Cascade Gateway 142
 Fehlerbehandlung 74
Installieren
 das TDS-Servlet 148
Interface (Java, JDBC) 2

Internationalisierung 34
 Isql-Applet
 Beispiel ausführen 145

J

JARS
 preloading 84
 Java-Objekte
 Daten der Spalte speichern als 76
 in ASA 6.0 speichern und abrufen 76
 Java-Objekte als Spaltendaten in einer Tabelle
 speichern 76
 jConnect
 Anwendungen migrieren auf Version 4.2 und 5.2
 131
 Anwendungen migrieren auf Version 5.x 130
 aufrufen 11
 Cascade Gateway 141
 Cursor verwenden 48
 Definition 4
 einrichten 6
 Fehlersuche 102
 Gateways 138
 Performance verbessern 116
 sample programs 175
 Speicherprobleme in Anwendungen 110
 Verbindungseigenschaften festlegen 12
 jconnect
 Anwendungen auf 4.1 migrieren 130
 jConnect 4.x
 SCROLL_INSENSITIVE Ergebnismengen 57
 JCONNECT_VERSION, Verbindungseigenschaft
 9, 14
 JDBC
 Definition 2
 Einschränkungen, Beschränkungen und
 Abweichungen 97
 Schnittstellen 2
 Treiberarten 2
 JDBC 2.0
 Standarderweiterungen 85
 jdbc.drivers 11
 JDBC-Treiber
 eigene-API-Teil-Java 3

JDBC-ODBC Bridge-Treiber 3
 JDBC-ODBC-Brücke 2
 Native-Protocol-All-Java 3
 Net-Protocol-All-Java 3
 JNDI
 Kontextinformationen 26
 verwenden 22
 JNDI für Datenbankbenennung 85

K

Konventionen 10

L

LANGUAGE, Verbindungseigenschaft 6, 14
 LANGUAGE_CURSOR 126
 LANGUAGE_CURSOR, Verbindungseigenschaft 14
 Lightweight Directory Access Protocol (LDAP) 23
 LITERAL_PARAMS, Verbindungseigenschaft 14
 Lokalisierung 34

M

Mehrbyte-Zeichensätze
 Konverter-Klassen 34
 unterstützte 36
 Metadata
 Zugriff 47
 Metadaten
 serverseitige Installation 48
 USE_METADATA 18
 Multithreading
 Anpassungen machen 97

N

Native-Protocol-Aall-Java 3
 Net-Protocol-All-Java 3

O

Open Server Gateway 21

P

PACKETSIZE, Verbindungseigenschaft 14
Password 14
Performance und Optimierung 116
Positionierte Aktualisierungen und Löschungen
 mit JDBC 1.x-Methoden 52
 mit JDBC 2.0-Methoden 53
Preloading JARS 84
PROTOCOL_CAPTURE, Verbindungseigenschaft 14
PROXY, Verbindungseigenschaft 15
PureConverter-Klasse 34

R

Remote Procedure Calls (RPCs)
 Server-zu-Server 45
REMOTEPWD, Verbindungseigenschaft 15
REPEAT_READ 117
REPEAT_READ, Verbindungseigenschaft 15
REQUEST_HA_SESSION 15
rs.getBytes() 67

S

Sample programs 175
Schreibkonventionen 10
SCROLL_INSENSITIVE Ergebnismengen in jConnect 4.x
 57
SELECT_OPENS_CURSOR, Verbindungseigenschaft
 16
Senden
 Bilddaten 63
SERIALIZE_REQUESTS, Verbindungseigenschaft 16
Server-zu-Server Remote Procedure Calls 45
SERVICENAME, Verbindungseigenschaft 16
Servlet-Argumente
 Debug 149
 SkipDoneProc 149
 TdsResponseSize 149

 TdsSessionIdleTimeout 149
Servlets 138
 TDS 138
SESSION_ID, Verbindungseigenschaft 16
SESSION_TIMEOUT, Verbindungseigenschaft 17
setRemotePassword() 46
Setzen
 TDS-Servlet-Argumente 149
Spalten
 aktualisieren in Cursor-Ergebnismengen 53
 löschen in Cursor-Ergebnismengen 52
Speichern von Java-Objekten als Spaltendaten in einer
 Tabelle
 Java-Objekte an eine Datenbank senden 77
 Java-Objekte von einer Datenbank empfangen 78
 Voraussetzungen 77
Speicherprobleme in jConnect-Anwendungen 110
SQL Exceptions
 siehe SQL-Ausnahmefehler und Warnungen 153
SQL-Ausnahmefehler und Warnungen 153
SQLNITSTRING, Verbindungseigenschaft 17
Statement.cancel() 10
STREAM_CACHE_SIZE, Verbindungseigenschaft
 17
Sybase-Erweiterungsänderungen 133
SybEventHandler 68
SybMessageHandler 72
SYB SOCKET_FACTORY, Verbindungseigenschaft
 17
Syntaxkonventionen 10
Systemeigenschaften
 jdbc.drivers 11

T

TDS 4
 capturing communication 106
 Servlet erforderliche Systemausstattung 148
 Servlet-Argumente setzen 149
 Servlets 138
 Servlets installieren 148
 Sitzungen protokollieren 150
 Sitzungen wieder aufnehmen 151
 Tunnelling 138
TDS-Sitzungen protokollieren 150

Technical Library 8
 Time, Date, und Timestamp Datentypen 66
 Treiber
 Eigenschaften 12
 JDBC-Arten 2
 TruncationConverter-Klasse 34, 39
 Tunnelling
 TDS 138
 TYPE_SCROLL_INSENSITIVE
 Einschränkungen 57

U

Unterstützung für die Erweiterungen des JDBC 2.0
 Optionspakets 85
 Unterstützung für Hochverfügbarkeit (HA) 40
 Unterstützung für verteilte Transaktionen 92
 URL
 Parameter der Verbindungseigenschaft 20
 Syntax 19
 USE_METADATA, Verbindungseigenschaft 18
 User 18

V

Verbinden
 Adaptive Server Anywhere 21
 Verbindung
 Fehler 108, 109
 Gateway-Verbindung abgelehnt 108
 mit Adaptive Server 18
 mit einem Server über JNDI 22
 Pools 89
 Verbindung zum Cascade Gateway 146
 Verbindungseigenschaften
 APPLICATIONNAME 12
 CANCEL_ALL 6, 10, 13
 CHARSET 6, 13
 CHARSET_CONVERTER 6
 CHARSET_CONVERTER_CLASS 13, 35
 CONNECTION_FAILOVER 13, 22
 DYNAMIC_PREPARE 13
 EXPIRESTRING 13
 festlegen 12

HOSTNAME 13
 HOSTPROC 13
 IGNORE_DONE_IN_PROC 13
 im URL setzen 20
 JCONNECT_VERSION 9, 14
 LANGUAGE 6, 14
 LANGUAGE_CURSOR 14, 126
 LANGUAGE_CURSOR und Cursor-Performance
 126
 LITERAL_PARAMS 14
 PACKETSIZE 14
 password 14
 PROTOCOL_CAPTURE 14
 PROXY 15
 REMOTEPWD 15
 REPEAT_READ 15, 117
 REQUEST_HA_SESSION 15
 SELECT_OPENS_CURSOR 16
 SERIALIZE_REQUESTS 16
 SVCENAME 16
 SESSION_ID 16
 SESSION_TIMEOUT 17
 SQLINITSTRING 17
 STREAM_CACHE_SIZE 17
 SYB_SOCKET_FACTORY 17
 USE_METADATA 18
 user 18
 VERSIONSTRING 18
 Verbindungspools 89
 VERSIONSTRING, Verbindungseigenschaft 18
 Vorbereitete Anweisung
 verwenden mit Cursorn 56

W

Währungssymbol, Euro 38
 Webserver Gateways 138
 Weitere Handbücher 7
 Wieder aufnehmen
 TDS-Sitzungen 151

X

XAServer 92

Z

Zeichensätze

einstellen 35

Konverter-Klassen 34

unterstützte 36

Zeichensatz-Konverterklasse

auswählen 35

Zeichensatzkonvertierung

Performance des Treibers verbessern 118

Performance verbessern 36

Zeichensatz-Konvertierungsklassen

PureConverter 34

TruncationConverter 34

Zeilen

aus einer Cursor-Ergebnismenge löschen 55

in eine Cursor-Ergebnismenge einfügen 55

Zielgruppe 7