Java dans Adaptive Server Enterprise

Adaptive Server Enterprise

12.5

Réf. du document : 37452-01-1250-01
Dernière mise à jour : Juin 2001

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

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

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

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

Table des matières

Préface ..... **xi**

CHAPITRE 1 **Introduction au langage Java dans la base de données** **1**
- Avantages offerts par Java dans la base de données 1
- Fonctionnalités Java dans la base de données 2
  - Appel des méthodes Java dans la base de données 2
  - Stockage des classes Java comme types de données 3
  - Stockage et interrogation de XML dans la base de données 4
- Normes 5
- Java dans la base de données : questions et réponses 6
  - Quelles sont les caractéristiques clés ? 6
  - Comment stocker des instructions Java dans la base de données ? 7
  - Comment les instructions Java sont-elles exécutées dans la base de données ? 7
  - Comment utiliser Java et SQL ensemble ? 8
  - Qu'est-ce que l'API Java ? 9
  - Comment accéder à l'API Java à partir du système SQL ? 9
  - Quelles sont les classes Java supportées par l'API Java ? 9
  - Puis-je installer mes propres classes Java ? 9
  - Puis-je accéder aux données à l'aide de Java ? 10
  - Puis-je utiliser les mêmes classes sur le client et le serveur ? 10
  - Comment utiliser les classes Java dans SQL ? 10
  - Où puis-je trouver des informations relatives à Java dans la base de données ? 11
  - Que ne permet pas de faire Java dans la base de données ? 11
- Exemples de classes Java 12

CHAPITRE 2 **Préparation et maintenance de Java dans la base de données** **13**
- Environnement d'exécution Java 13
  - Classes Java dans la base de données 13
  - Pilotes JDBC 14
  - Machine virtuelle Java 14

Configuration de la mémoire pour Java dans la base de données 15
Activation de la gestion des objets Java par le serveur 15
Désactivation de la gestion des objets Java par le serveur 15
Création de classes Java et de fichiers JAR 16
Ecriture du code Java 16
Compilation du code Java 16
Sauvegarde des classes dans un fichier JAR 17
Installation de classes Java dans la base de données 17
Utilisation de installjava 18
Référencement d'autres classes Java-SQL 20
Visualisation des informations sur les classes installées et les fichiers JAR 21
Téléchargement des informations sur les classes installées et les fichiers JAR 22
Suppression de classes et de fichiers JAR 22
Enregistrement des classes 23

CHAPITRE 3 Utilisation des classes Java avec le langage SQL 25
Concepts généraux 26
Considérations relatives à Java 26
Noms Java-SQL 27
Utilisation des classes Java comme types de données 27
Création et modification des tables contenant des colonnes Java-SQL 28
Sélection, insertion, mise à jour et suppression des objets Java 30
Appel des méthodes Java en SQL 33
Exemples de méthodes 34
Exceptions dans les méthodes Java-SQL 35
Représentation des instances Java 35
Propriétés de l'affectation des éléments de données Java-SQL 36
Mappage de type de données entre les champs Java et SQL 38
Jeux de caractères pour les données et les identificateurs 40
Sous-types Java-SQL 40
Conversion élargissante 41
Conversion rétrécissante 41
Types de données d'exécution et de compilation 42
Traitement des valeurs NULL dans les données Java-SQL 43
Références aux champs et aux méthodes des instances NULL 43
Valeurs NULL utilisées comme arguments des méthodes Java-SQL 45
Valeurs NULL lors de l'utilisation de la fonction SQL convert . 46
Données de type String Java-SQL 47

Chaîne longueur nulle 47
Méthodes typées et void 48
Méthodes d'instance Java void 49
Méthodes statiques Java Void 50
Opérations d'égalité et d'ordre 51
Ordre d'évaluation et appels de méthode Java 52
Colonnes 52
Variables et paramètres 53
Variables statiques dans les classes Java-SQL 53
Classes Java dans plusieurs bases de données 54
Portée 54
Références entre les bases de données 55
Transferts interclasse 56
Transmission d'arguments interclasse 57
Bases de données temporaires et de travail 57
Classe Java 58

CHAPITRE 4 Accès aux données à l'aide de JDBC 63
Présentation 63
Concepts JDBC et terminologie 64
Différences entre JDBC client et JDBC serveur 65
Autorisations 66
Utilisation de JDBC pour accéder aux données 66
Présentation de la classe JDBCExamples 66
Méthodes JDBC main( ) et serverMain( ) 67
Etablissement d'une connexion JDBC : méthode JDBC Connecter( ) 69
Acheminement de l'action vers d'autres méthodes : méthode doAction( ) 70
Exécution des opérations SQL impératives : méthode doSQL( ) 70
Exécution d'une instruction update : méthode UpdateAction( ) 70
Exécution d'une instruction select : méthode selectAction( ) .. 71
Appel d'une procédure stockée SQL : méthode callAction( ) .. 73
Erreur de gestion du pilote JDBC natif 74
Classe JDBCExamples 76
Méthode main( ) 76
Méthode internalMain( ) 77
Méthode connecter( ) 77
Méthode doAction( ) 78
Méthode doSQL( ) 79
Méthode updateAction( ) 79
Méthode selectAction( ) 80
Méthode callAction( ) 80

CHAPITRE 5 Procédures stockées et fonctions SQLJ 83
Présentation 83
Conformité avec les spécifications de SQLJ Partie 1 84
Aspects généraux 85
Sécurité et autorisations 85
SQLJExamples 86
Appel des méthodes Java dans Adaptive Server 87
Utilisation de Sybase Central pour la gestion des fonctions et les procédures SQLJ 89
Fonctions SQLJ définies par l'utilisateur 90
Gestion des valeurs d'argument NULL 93
Suppression d'un nom de fonction SQLJ 96
Procédures stockées SQLJ 96
Modification de données SQL 98
Utilisation de paramètres d'entrée et de sortie 100
Retour de jeux de résultats 103
Visualisation des informations sur les fonctions et les procédures SQLJ 107
Fonctions avancées 108
Mappage de type de données Java et SQL 108
Utilisation de la méthode de commande main 112
Implémentation SQLJ et Sybase : comparaison 113
Classe SQLJExamples 116

CHAPITRE 6 Introduction à XML dans la base de données 121
Introduction 122
Code source et Javadoc 122
Références 123
Présentation du langage XML 123
Exemple de document XML 124
Types de document XML 128
XSL : Présentation des informations XML 130
Jeux de caractères et données XML 131
Analyseur XML 132

CHAPITRE 7 Sélection de données avec XQL 133
Accès à l'analyseur XML 134
Définition de la variable d'environnement CLASSPATH 134
Installation de XQL dans Adaptive Server 134
Conversion d'un document XML brut en version analysée 135
Insertion de documents XML 136
Mise à jour de documents XML 136
Suppression de documents XML 136

Besoins en mémoire pour l'exécution du moteur de requêtes dans Adaptive Server 137
Utilisation de XQL 137
Structures de requête ayant une influence sur les performances 140
Exemples 140
Autres applications du package XQL 142
Syntaxe com.sybase.xml.xql.XqlDriver 143
Validation de votre document 145
Utilisation de XQL dans le développement d'applications autonomes 145
Méthodes XQL 148
Méthodes dans com.sybase.xml.xql.Xql 149
parse(String xmlDoc) 149
parse(InputStream xml_document, boolean validate) 150
query(String query, String xmlDoc) 150
query(String query, InputStream xmlDoc) 151
query(String query, SybXmlStream xmlDoc) 151
query(String query, JXml jxml) 152
sybase.aseutils.SybXmlStream 152
com.sybase.xml.xql.store.SybMemXmlStream 153
com.sybase.xml.xql.store.SybFileXmlStream 153
setParser(String parserName) 153
reSetParser 154

CHAPITRE 8 Traitement XML spécialisé 155
Classe OrderXml des documents Commande 155
Constructeur OrderXml(String) 156
OrderXml(date, customerid, server) 156
void order2Sql(String ordersTableName, String server) 157
static void createOrderTable (String ordersTableName, String server) 157
void setOrderElement (String elementName, String newValue) 157
Chaîne getItemElement (int itemNumber, String elementName) 158
void setItemElement (int itemNumber, String elementName, String newValue) 158
Chaîne getItemAttribute (int itemNumber, elementName, attributeName) 158
void setItemAttribute (int itemNumber, elementName, attributeName, newValue) 159
void appendItem (newItemid, newItemName, newQuantity, newUnit) 159

void deleteItem(int itemNumber) 159
Stockage des documents XML 160
Mappage et stockage 160
Avantages et inconvénients des options de stockage..... 161
Considérations relatives au client et au serveur..... 162
Création et remplissage des tables SQL avec les données Commande 162
Tables de stockage d'éléments 162
Tables de stockage de documents et tables de stockage hybride..... 164
Utilisation de la technique de stockage d'éléments..... 164
Assemblage de documents Commande à partir de données SQL 164
Conversion d'un document XML Commande en données SQL..... 166
Utilisation de la technique de stockage de documents 168
Stockage des documents XML Commande dans des colonnes SQL 168
Accès aux éléments des documents XML Commande stockés..... 169
Accès serveur aux éléments Commande..... 172
Ajout et suppression d'articles dans le document XML..... 172
Utilisation de la technique de stockage hybride 173

CHAPITRE 9 XML pour les jeux de résultats SQL 175
Classe ResultSetXML 175
ResultSetXml(String) 176
Constructeur : ResultSetXml (query, cdataColumns, colNames, server) 176
Exemple de ResultXml 176
String toSqlScript (resultTableName, columnPrefix, goOption) 177
String getColumn(int rowNumber, int columnNumber)..... 177
String getColumn(int rowNumber, String columnName) 177
void setColumn (int rowNumber, int columnNumber, newValue)..... 178
void setColumn (int rowNumber, String columnName, newValue) 178
Boolean allString (int ColumnNumber, String compOp, String comparand) 178
Boolean someString (int ColumnNumber, String compOp, String comparand) 179
Exemple personnalisable pour des jeux de résultats différents ... 179
Type de document ResultSet 180

Utilisation de la technique de stockage de documents 184
Génération d'un document ResultSet dans le client 185
Génération d'un jeu de résultats dans Adaptive Server 186
Traduction du document XML ResultSet dans le client 186
Traduction du document XML ResultSet dans Adaptive Server .. 188
Utilisation de la technique de stockage de documents 188
Stockage d'un document XML ResultSet dans une colonne SQL 188
Accès aux colonnes des documents ResultSet stockés 189
Comparaisons quantifiées dans les documents ResultSet stockés 192

CHAPITRE 10 Débogage de Java dans la base de données 197
Introduction au débogage de Java 197
Fonctionnement du débogueur 197
Conditions requises pour utiliser le débogueur Java 198
Applications possibles du débogueur 198
Utilisation du débogueur 198
Démarrage du débogueur et connexion à la base de données 199
Compilation des classes à déboguer 199
Réquisition d'une machine virtuelle Java 200
La fenêtre Source 200
Options 201
Définition des points d'arrêt 202
Déconnexion de la base de données 205
Didacticiel de débogage 205
Notes préliminaires 205
Démarrage du débogueur Java et connexion à la base de données 206
Réquisition d'une machine virtuelle Java 206
Chargement du code source dans le débogueur 207
Passage en revue du code source 208
Vérification et modification des variables 209

CHAPITRE 11 Accès au réseau avec java.net 211
Présentation 211
Classes java.net 212
Configuration de java.net 212
Exemple d'utilisation 213
Utilisation des classes socket 213
Utilisation de la classe URL 216
Remarques à l'attention de l'utilisateur 219
Complément d'information 219

CHAPITRE 12 Rubriques de référence 221
Affectation 221
Règles d'affectation pour la compilation 222
Règles d'affectation pour l'exécution 222
Conversions autorisées 223
Transfert d'objets Java-SQL vers les clients 223
Packages, classes et méthodes d'API Java supportés 224
Les packages et les classes Java supportés 224
Packages et classes Java non supportés 225
Méthodes et interfaces java.sql non supportées 226
Appel de SQL à partir de Java 227
Considérations spéciales 227
Commandes Transact-SQL à partir des méthodes Java 228
Mappage de type de données entre Java et SQL 232
Identificateurs Java-SQL 233
Noms de classe et de package Java-SQL 234
Déclarations de colonnes Java-SQL 235
Déclarations de variables Java-SQL 235
Références de colonnes Java-SQL 236
Références de membres Java-SQL 237
Appels de méthode Java-SQL 238

Glossaire 241

Index 247

Préface

Le présent document explique comment installer et utiliser les classes et les méthodes Java de la base de données Sybase® Adaptive Server® Enterprise.

A qui s'adresse ce manuel ?

Ce manuel s'adresse aux administrateurs système Sybase, aux propriétaires de bases de données et aux utilisateurs familiarisés avec les langages de programmation Java et Transact-SQL®, la version Sybase de SQL (Structured Query Language). Les utilisateurs de JDBC (Java Database Connectivity), XML (eXtensible Markup Language) et XQL (Extensible Query Language) sont supposés connaître les fonctionnalités de ces langages.

Comment utiliser ce manuel ?

Ce manuel est conçu pour vous aider à installer, configurer et utiliser les classes et les méthodes Java dans la base de données Adaptive Server. Il comprend les chapitres suivants :

- Le chapitre 1, "Introduction au langage Java dans la base de données", présente succinctement Java dans Adaptive Server, avec à l'appui une section "Questions et réponses" destinée aux utilisateurs novices et aux utilisateurs chevronnés de Java.
- Le chapitre 2, "Préparation et maintenance de Java dans la base de données", décrit l'environnement d'exécution Java et des étapes à suivre pour activer Java sur le serveur et installer les classes Java.
- Le chapitre 3, "Utilisation des classes Java avec le langage SQL", explique comment utiliser Java-SQL dans votre base de données Adaptive Server.
- Le chapitre 4, "Accès aux données à l'aide de JDBC", explique comment utiliser un pilote JDBC (sur le serveur ou sur le client) pour exécuter des opérations SQL en Java.
- Le chapitre 5, "Procédures stockées et fonctions SQLJ", explique comment inclure et utiliser les méthodes Java dans les encapsulations SQL.

- Le chapitre 6, "Introduction à XML dans la base de données", présente XML, ainsi que les méthodes de stockage de documents XML dans Adaptive Server et de génération à partir des données SQL.

  Les chapitres 7, 8 et 9 traitent d'autres applications de XML dans la base de données Adaptive Server.

- Le chapitre 7, "Sélection de données avec XQL", explique comment sélectionner les données brutes d'Adaptive Server à l'aide du langage XQL et les afficher sous forme de document XML.

- Le chapitre 8, "Traitement XML spécialisé", décrit la classe OrderXML, conçue pour une application exemple utilisant des documents XML pour les données Commande ; ce chapitre est consacré spécifiquement au traitement des documents XML correspondant aux données de la commande.

- Le chapitre 9, "XML pour les jeux de résultats SQL", décrit la classe ResultSetXML qui vous permet de générer un document XML représentant un jeu de résultats SQL, d'accéder, et de mettre à jour ce document XML.

- Le chapitre 10, "Débogage de Java dans la base de données", explique comment utiliser le débogueur Sybase avec Java.

- Le chapitre 11, "Accès au réseau avec java.net", explique comment utiliser java.net, un package qui vous permet de créer des applications de réseau via TCP/IP. Les classes s'exécutant dans Adaptive Server peuvent accéder tout type de serveur.

- Le chapitre 12, "Rubriques de référence", fournit des informations sur le mappage de type de données, la syntaxe Java-SQL et d'autres instructions utiles.

En outre, un glossaire donne la définition des termes Java et Java-SQL utilisés dans ce manuel.

Documentation

La documentation Sybase Adaptive Server Enterprise comprend les manuels suivants :

- Les Notes de mise à jour pour votre plate-forme contiennent les informations de dernière minute qui ne figurent pas dans les manuels.

  Une version plus récente des Notes de mise à jour est disponible sur le Web. Pour rechercher des informations ultérieures à la commercialisation du CD-ROM du produit, consultez le site Sybase Technical Library.

- Le *guide d'installation* d'Adaptive Server pour votre plate-forme décrit les procédures d'installation, de mise à niveau et de configuration pour tous les produits Adaptive Server et Sybase associés.
- Le *Manuel de configuration d'Adaptive Server Enterprise* pour votre plate-forme fournit des instructions sur les tâches de configuration particulières pour Adaptive Server.
- *Adaptive Server Enterprise - Nouvelles fonctionnalités* : décrit les nouvelles fonctionnalités d'Adaptive Server version 12.5, les modifications système introduites pour les supporter et les changements qui risquent d'avoir une influence sur les applications existantes.
- Le *Guide de l'utilisateur Transact-SQL* présente Transact-SQL, version enrichie du langage de base de données relationnelle de Sybase. Ce manuel sert de référence pour les utilisateurs qui découvrent le système de gestion de bases de données. Il décrit également les bases de données exemple pubs2 et pubs3.
- Le *Guide d'administration système* fournit des informations détaillées sur l'administration des serveurs et des bases de données. Ce manuel contient des instructions relatives à la gestion des ressources physiques, de la sécurité et des bases de données système et utilisateur, ainsi qu'au paramétrage de la conversion de caractères, la langue et l'ordre de tri.
- Le *Manuel de référence* contient des informations détaillées sur toutes les commandes, fonctions, procédures et types de données Transact-SQL. Ce manuel fournit également la liste des mots réservés Transact-SQL et les définitions des tables système.
- Le document *Performances et optimisation* explique comment optimiser les performances d'Adaptive Server. Il contient des informations sur les aspects de la conception des bases de données qui conditionnent les performances, sur l'optimisation des requêtes, sur le paramétrage d'Adaptive Server pour des bases de données volumineuses, sur la configuration des caches et des disques et sur l'impact du verrouillage et des curseurs sur les performances.
- Le manuel *Utilitaires* décrit les utilitaires d'Adaptive Server tels qu'isql et bcp, qui sont exécutés au niveau du système d'exploitation.
- Le *Guide de référence rapide* est un petit fascicule qui répertorie les noms et syntaxes des commandes, fonctions, procédures système, procédures système étendues, types de données et utilitaires. Il est uniquement disponible sur papier.

- Le *Diagramme des tables système* est un poster qui illustre les tables système selon le modèle entités-relations. Il est uniquement disponible sur papier.
- Les documents *Error Messages et Troubleshooting Guide* expliquent comment résoudre les conditions d'erreur les plus courantes et donnent les solutions aux problèmes système souvent rencontrés par les utilisateurs.
- Le *Guide de l'utilisateur de Component Integration Services* explique comment utiliser la fonctionnalité Component Integration Services d'Adaptive Server pour se connecter à des bases de données distantes Sybase et non Sybase.
- Le document *Utilisation de Sybase Failover en environnement haute disponibilité* traite de l'utilisation de Sybase Failover pour configurer Adaptive Server comme serveur compagnon dans un environnement haute disponibilité.
- Le document *Utilisation des fonctionnalités DTM* traite de la configuration et de l'utilisation des fonctionnalités DTM d'Adaptive Server ainsi que de la résolution des éventuels problèmes dans les environnements de traitement des transactions distribuées.
- Le *Guide de l'utilisateur d'EJB Server* explique comment utiliser EJB Server pour déployer et exécuter Enterprise JavaBeans dans Adaptive Server.
- Le document *XA Interface Integration Guide for CICS, Encina, and TUXEDO* fournit des instructions sur l'utilisation de l'interface DTM XA de Sybase avec les gestionnaires de transactions X/Open XA.
- Le *Glossaire* définit les termes techniques utilisés dans la documentation Adaptive Server.
- Le document *Sybase jConnect for JDBC Programmer's Reference* décrit le produit jConnect for JDBC et explique comment l'utiliser pour accéder aux données stockées dans des systèmes de gestion de bases de données relationnelles.
- Le document *Full-Text Search Specialty Data Store – Guide de l'utilisateur* explique comment utiliser la fonction Full-Text Search avec Verity afin d'effectuer des recherches dans les données d'Adaptive Server Enterprise.
- Le *Guide de l'utilisateur de Monitor Historical Server* explique comment utiliser Historical Server afin d'obtenir des statistiques de performances de SQL Server et Adaptive Server.

- Le *Guide de l'utilisateur de Monitor Server* explique comment utiliser Monitor Server afin d'obtenir des statistiques de performances de SQL Server et Adaptive Server.
- Le document *Monitor Client Library Programmer's Guide* explique comment écrire des applications Monitor Client Library accédant aux données de performances d'Adaptive Server.

Autres sources d'information

Utilisez le CD Sybase Technical Library ainsi que le site Web Technical Library Product Manuals pour obtenir davantage d'informations sur les produits :

- Le CD-ROM Technical Library, qui contient des manuels sur les produits et des documents techniques, est livré avec le logiciel. L'explorateur DynaText (téléchargeable à partir du site Product Manuals (http://www.sybase.com/detail/1,3693,1010661,00.html)) vous permet d'accéder aux informations techniques sur les produits dans un format facile à utiliser.

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

- Le site Web Technical Library Product Manuals est une version HTML du CD Technical Library, à laquelle vous pouvez accéder à l'aide d'un navigateur Web standard. Outre les manuels sur les produits, vous trouverez également des liens vers le site Web Technical Documents (anciennement Tech Info Library), la page Solved Cases et des forums Sybase/Powersoft.

  Pour accéder à Technical Library Product Manuals, rendez-vous sur le site Product Manuals (http://www.sybase.com/support/manuals/).

Certifications Sybase sur le Web

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

❖ **Pour accéder aux informations les plus récentes sur les certifications de produit :**

1. Cliquez sur Technical Documents (http://www.sybase.com/support/techdocs/).
2. Sélectionnez des produits dans la barre de navigation située à gauche.
3. Sélectionnez un nom de produit dans la liste des produits.
4. Sélectionnez le filtre Certification Report, choisissez une période de temps dans la liste Time Frame, puis cliquez sur Go.
5. Cliquez sur le titre du rapport de certification que vous voulez consulter.

❖ **Pour accéder aux informations les plus récentes sur les recueils de correctifs de bugs et les mises à jour**

1 Cliquez sur Technical Documents (http://www.sybase.com/support/techdocs/).

2 Sélectionnez EBFs/Updates. Saisissez vos nom d'utilisateur et mot de passe si vous y êtes invité (pour les comptes Web existants) ou créez un compte (service gratuit).

3 Sélectionnez une période de temps dans la liste Time Frame, puis cliquez sur Go.

4 Sélectionnez un produit.

5 Cliquez sur l'EBF ou la mise à jour souhaité.

❖ **Pour créer une vue personnalisée du site Web de Sybase (y compris des pages de support technique)**

Créez un profil MySybase. MySybase est un service gratuit qui vous permet de créer une vue personnalisée des pages Web de Sybase.

1 Cliquez sur Technical Documents (http://www.sybase.com/support/techdocs/).

2 Cliquez sur MySybase, puis créez un profil MySybase.

Conventions syntaxiques pour les instructions Java

Ce manuel utilise les polices et les conventions syntaxiques suivantes pour les éléments Java :

- Les classes, les interfaces, les méthodes et les packages apparaissent en Helvetica. Par exemple :

  interface SybEventHandler

  méthode setBinaryStream( )

  package com.Sybase.jdbx

- Les objets, les instances et les noms de paramètre sont indiqués en italique. Par exemple :

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

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

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

- Les noms Java tiennent compte des majuscules/minuscules. Par exemple, si un nom de méthode Java s'affiche sous la forme Misc.stripLeadingBlanks( ), vous devez saisir le nom de la méthode tel qu'il est affiché.

Conventions syntaxiques pour les isntructions Transact-SQL

Ce manuel utilise les mêmes polices et conventions syntaxiques pour Transact-SQL que pour les autres documents Adaptive Server :

- Les noms de commande, d'option de commande, d'utilitaire, de drapeau d'utilitaire et autres mots-clés apparaissent en Helvetica. Par exemple :

  commande select

  utilitaire isql

  argument -f

- Les variables (termes se substituant aux valeurs à insérer) apparaissent en italiques. Par exemple :

  nom_utilisateur

  nom_serveur

- Les fragments de code sont indiqués dans une police à espacement non proportionnel. Les variables dans les fragments de code termes se substituant aux valeurs à insérer apparaissent en italique. Par exemple :

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

- Les mots-clés Transact-SQL ne tiennent pas compte des majuscules et des minuscules. Par exemple, SELECT, Select et select sont identiques.

Le tableau 1 indique les conventions syntaxiques utilisées pour les instructions. Des exemples illustrant chaque convention sont disponibles dans le *Guide d'administration système*.

Tableau 1 : Conventions syntaxiques des instructions

Clé	Définition
{ }	Les accolades indiquent que vous devez choisir au moins une des options énumérées. N'insérez pas les accolades dans l'option.
[ ]	Les crochets indiquent que les options citées sont facultatives. N'incluez pas les crochets à votre option.
( )	Les parenthèses font partie de la commande et doivent être saisies.
|	La barre verticale indique que vous ne pouvez choisir qu'une seule des options énumérées.
,	La virgule vous permet de choisir autant d'options que vous le souhaitez, à condition de les séparer par une virgule dans la commande.

Si vous avez besoin d'aide

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

CHAPITRE 1

Introduction au langage Java dans la base de données

Ce chapitre fournit une présentation succincte des classes Java dans Adaptive Server Enterprise.

Rubrique	Page
Avantages offerts par Java dans la base de données	1
Fonctionnalités Java dans la base de données	2
Normes	5
Java dans la base de données : questions et réponses	6
Exemples de classes Java	12

Avantages offerts par Java dans la base de données

Adaptive Server fournit un environnement d'exécution Java permettant d'exécuter le code Java dans le serveur. La création d'un environnement d'exécution Java sur le serveur de base de données offre des solutions puissantes et novatrices pour gérer et stocker les données et la logique.

- Vous pouvez utiliser le langage de programmation Java comme partie intégrante de Transact-SQL.
- Vous pouvez réutiliser le code Java dans les différentes couches de votre application (client, niveau intermédiaire ou serveur) et utiliser celles-ci chaque fois que vous le jugez nécessaire.
- Java dans Adaptive Server est un langage plus puissant que les procédures stockées pour créer des flux de contrôle dans la base de données.
- Les classes Java deviennent des types de données utilisateur complexes.
- Les méthodes des classes Java offrent de nouvelles fonctions accessibles à partir du système SQL.

- Le langage Java peut être utilisé dans la base de données sans que l'intégrité, la sécurité et la robustesse de cette dernière soient compromises. L'utilisation du langage Java n'influe pas sur le comportement des instructions SQL existantes ni sur d'autres aspects du fonctionnement des bases de données relationnelles non-Java.

Fonctionnalités Java dans la base de données

Java dans Adaptive Server vous permet :

- d'appeler des méthodes Java dans la base de données ;
- de stocker les classes Java en tant que types de données ;
- de stocker et d'interroger XML dans la base de données.

Appel des méthodes Java dans la base de données

Vous pouvez installer des classes Java dans Adaptive Server, puis appeler leurs méthodes statiques de deux manières différentes :

- Vous pouvez appeler les méthodes Java directement depuis le système SQL.
- Vous pouvez encapsuler les méthodes dans des noms SQL et les appeler comme s'il s'agissait de procédures stockées Transact-SQL standard.

Appel des méthodes Java directement depuis le système SQL

Les méthodes d'un langage orienté objet correspondent aux fonctions d'un langage procédural. Vous pouvez appeler les méthodes stockées dans la base de données en les référençant, par qualification du nom, sur des instances pour les méthodes d'instance et sur des instances ou des classes pour les méthodes statiques (de classe). Vous pouvez appeler la méthode directement dans les listes de sélection Transact-SQL et les clauses where, par exemple.

Vous pouvez utiliser les méthodes statiques qui renvoient une valeur au demandeur comme fonctions utilisateur (UDF).

L'utilisation des méthodes Java est alors soumise à certaines restrictions :

- Si une méthode Java accède à la base de données via JDBC, les valeurs des jeux de résultats sont disponibles uniquement pour la méthode Java et non pour l'application cliente.
- Les paramètres de sortie ne sont pas supportés. Une méthode peut manipuler les données qu'elle reçoit d'une connexion JDBC, mais la seule valeur qu'elle peut renvoyer à son demandeur est une valeur de retour unique déclarée dans le cadre de sa définition.

Appel des méthodes Java en tant que fonctions et procédures stockées SQLJ

Vous pouvez encapsuler les méthodes statiques Java dans SQL et les utiliser comme s'il s'agissait de fonctions intégrées et de procédures stockées Transact-SQL. Cette fonctionnalité :

- permet aux méthodes Java de renvoyer des paramètres de sortie et des jeux de résultats à l'environnement demandeur ;
- vous permet d'exploiter des fonctionnalités traditionnelles comme la syntaxe SQL, les métadonnées et l'autorisation ;
- vous permet d'appeler les fonctions SQLJ sur les bases de données ;
- vous permet d'utiliser les méthodes Java existantes en tant que fonctions et procédures SQLJ sur le serveur, sur le client et sur toute base de données tierce compatible SQLJ ;
- est conforme aux dispositions de la Section 1 de la norme ANSI. Pour plus d'informations, reportez-vous à la section "Normes", page 5.

Stockage des classes Java comme types de données

Avec Java dans la base de données, vous pouvez installer des classes Java pures (Pure Java) dans un système SQL, puis les utiliser naturellement comme types de données dans une base de données SQL. Cette fonctionnalité utilise un modèle universellement interprétable et un langage portable largement répandu pour ajouter dans SQL un mécanisme d'extension de type de données entièrement orienté objet. Les objets créés et stockés à l'aide de cette fonctionnalité sont prêts à être transférés vers n'importe quel environnement compatible Java, soit dans un autre système SQL, soit dans un environnement Java autonome.

L'emploi de classes Java dans la base de données est utile pour deux raisons différentes mais complémentaires :

- Il fournit un mécanisme d'extension de type pour SQL, utilisable sur les données créées et traitées dans SQL.
- Il fournit une fonctionnalité de stockage permanent des données pour Java, permettant de stocker en SQL des données créées et traitées (principalement) en Java. Java dans Adaptive Server présente un net avantage sur les fonctionnalités SQL traditionnelles : vous n'avez pas besoin de mapper les objets Java sur des types de données SQL scalaires ni de stocker les objets Java sous forme de chaînes binaires sans type.

Stockage et interrogation de XML dans la base de données

Proche du langage HTML (Hypertext Markup Language), le langage XML (eXtensible Markup Language) vous permet de définir vos propres balises de marquage orientées application, propriété qui le rend particulièrement adapté pour l'échange de données.

Adaptive Server vous permet :

- de générer des documents au format XML à partir de données brutes stockées dans Adaptive Server ;
- de stocker les documents XML, ainsi que les données qui en sont extraites, dans Adaptive Server ;
- d'interroger les documents XML diffusés sur le Web.

Adaptive Server emploie le langage XQL (XML Query Language) pour effectuer des recherches dans les documents XML. Un programme de traitement de requêtes XQL compatible Java est livré avec Adaptive Server. Nombre d'outils généralement utilisés pour le traitement des données XML sont écrits en Java, Adaptive Server est un excellent tremplin pour les applications XML compatibles SQL.

Normes

Le consortium SQLJ de revendeurs SQL développe actuellement des spécifications sur l'utilisation de Java avec SQL. Ces spécifications sont soumises à l'organisme ANSI pour homologation de leur normalisation. Les normes sont consultables sur le Web, à l'adresse http://www.ansi.org. Dans ce document, SQLJ fait référence aux fonctionnalités conformes aux dispositions de la Section 1 SQLJ de la norme.

La conformité aux normes SQLJ des fonctionnalités Sybase leur garantit une compatibilité avec toutes les bases de données relationnelles normalisées tierces.

La norme se divise en trois sections :

- *Section 0* – "Langage de base de données SQL – Section 10 : Object Language Bindings (SQL/OLB)", ANSI X3.135.10-1998.

  Il s'agit des spécifications relatives à l'encapsulation des instructions SQL dans les méthodes Java, similaires aux fonctionnalités SQL traditionnelles pour l'encapsulation des données SQL dans les langages COBOL, C ou autres. Les classes Java contenant des instructions SQL encapsulées sont précompilées en classes Java pures avec des appels JDBC.

- *Section 1* – "SQLJ – Section 1 : Routines SQL qui utilisent le langage de programmation Java", ANSI NCITS N331.1.

  Il s'agit des spécifications relatives à l'installation des classes Java dans un système SQL et à l'appel de méthodes statiques Java en tant que procédures stockées et fonctions SQL.

- *Section 2* – "SQLJ – Section 2 : Types SQL qui utilisent le langage de programmation Java", ANSI NCITS N331.2.

  Il s'agit des spécifications relatives à l'utilisation des classes Java en tant que types de données SQL.

Sybase supporte les dispositions de la Section 1 de la norme. En outre, Sybase complète les fonctionnalités spécifiées dans la norme. Par exemple, Adaptive Server vous permet de référencer les méthodes et les classes Java directement en SQL, tandis que les Sections 1 et 2 de la norme SQLJ stipulent l'utilisation d'alias SQL.

Java dans la base de données : questions et réponses

Même pour le lecteur familiarisé avec le langage Java, il y a beaucoup à apprendre sur l'emploi de Java dans une base de données. Sybase étend les fonctionnalités de la base de données avec Java, mais aussi celles de Java avec la base de données. Reporter-vous à la section ou au document.

Il est recommandé aux utilisateurs, chevronnés comme novices, du langage Java, de lire cette section. Son format "question-réponse" vous aidera à vous familiariser avec les rudiments de Java dans Adaptive Server.

Quelles sont les caractéristiques clés ?

Tous ces points sont traités en détail dans les sections qui suivent. Java dans Adaptive Server permet :

- d'exécuter Java dans le serveur de base de données à l'aide d'une machine virtuelle Java interne ;
- d'appeler des fonctions Java (méthodes) directement à partir d'instructions SQL ;
- d'encapsuler les méthodes Java dans des alias SQL et les appeler comme s'il s'agissait de procédures stockées et de fonctions intégrées SQL ;
- d'accéder aux données SQL à partir du système Java à l'aide d'un pilote JDBC interne ;
- d'utiliser les classes Java en tant que types de données SQL ;
- de sauvegarder des instances de classes Java dans des tables ;
- de générer des documents formatés en XML à partir de données brutes stockées dans des bases de données Adaptive Server et, inversement, de stocker des documents XML et des données extraites de ces documents dans des bases de données Adaptive Server ;
- de déboguer les instructions Java dans la base de données.

Comment stocker des instructions Java dans la base de données ?

Java est un langage orienté objet dont les instructions (code source) se présentent sous la forme de classes. Les instructions Java sont écrites et compilées à l'extérieur de la base de données dans des classes compilées (code octet), c'est-à-dire des fichiers binaires contenant les instructions Java.

Après quoi, les classes compilées sont installées dans la base de données, pour être exécutées par le serveur de base de données.

Adaptive Server est un environnement d'exécution Java. Vous avez besoin d'un environnement de développement Java, tel que Sybase PowerJ™ ou Java Development Kit (JDK) de Sun Microsystems, pour écrire et compiler les instructions Java.

Comment les instructions Java sont-elles exécutées dans la base de données ?

Pour supporter les instructions Java dans la base de données, Adaptive Server :

- est livré avec sa propre machine virtuelle Java, spécialement conçue pour gérer les traitements Java sur le serveur ;
- exécute son propre pilote JDBC sur le serveur pour accéder à une base de données.

La machine virtuelle Java de Sybase est exécutée dans l'environnement de base de données. Elle interprète les instructions Java compilées et les exécute sur le serveur de base de données.

La machine virtuelle Java de Sybase est conforme aux spécifications JCM de Java Software ; elle est conçue pour fonctionner avec la version 2.0 de l'API Java. Elle supporte les méthodes de classe et d'instance public, les classes qui héritent d'autres classes, l'API Java et l'accès aux champs protected, public et private. Certaines fonctions de l'API Java sont incompatibles avec un environnement serveur. Par exemple, les éléments de l'interface utilisateur ne sont pas reconnus. Toutes les classes et tous les packages d'API Java supportés sont fournis avec Adaptive Server.

La machine virtuelle Java d'Adaptive Server est disponible en permanence pour exécuter une opération Java chaque fois qu'elle est nécessaire à l'exécution d'une instruction SQL. Le serveur de base de données démarre automatiquement la machine virtuelle Java lorsqu'elle est requise ; vous n'avez pas besoin d'exécuter une action explicite pour démarrer ou arrêter la machine virtuelle Java.

JDBC client et serveur

JDBC est l'API standard pour exécuter les instructions SQL dans le système Java.

Adaptive Server fournit un pilote JDBC natif. Ce pilote est conçu pour optimiser les performances lorsqu'il est exécuté dans le serveur car il n'a pas besoin de communiquer via le réseau. Il permet aux classes Java installées dans une base de données d'utiliser les classes JDBC qui exécutent des instructions SQL.

Lorsque des classes JDBC sont utilisées au sein d'une application cliente, vous devez généralement utiliser jConnect™ for JDBC™, le pilote de base de données JDBC client de Sybase, pour fournir les classes nécessaires à l'établissement d'une connexion de base de données.

Comment utiliser Java et SQL ensemble ?

Le principe fondamental de Java dans la base de données est de fournir une extension naturelle et ouverte à la fonctionnalité SQL existante.

- *Les opérations Java sont appelées à partir de SQL* : Sybase a étendu la série d'expressions SQL pour y inclure des champs et des méthodes d'objets Java, de sorte que les opérations Java peuvent être incluses dans une instruction SQL.
- *Méthodes Java utilisées en tant que fonctions et procédures stockées SQLJ* : vous créez un alias SQLJ pour les méthodes statiques Java, en vue de les appeler en tant que procédures stockées et fonctions utilisateurs (UDF) SQL standard.
- *Les classes Java deviennent des types de données définis par l'utilisateur* : vous stockez les instances des classes Java à l'aide des mêmes instructions SQL que celles utilisées pour les types de données SQL traditionnels.

Vous pouvez utiliser des classes faisant partie de l'API Java et des classes créées et compilées par des développeurs Java.

Qu'est-ce que l'API Java ?

L'API (Application Programming Interface) Java est un ensemble de classes défini par Sun Microsystems. Cet ensemble offre une série de fonctionnalités de base qui peuvent être utilisées et enrichies par des développeurs Java. Il s'agit de la "boîte à outils" Java.

L'API Java offre des fonctionnalités considérables. Une grande partie de l'API Java est intégrée aux bases de données compatibles avec le code Java, notamment la majorité des classes non visuelles de l'API Java déjà connues des développeurs qui utilisent le JDK Sun Microsystems.

Comment accéder à l'API Java à partir du système SQL ?

Vous pouvez utiliser l'API Java dans des procédures stockées, des fonctions utilisateur et des instructions SQL comme des extensions des fonctions intégrées fournies par SQL.

Par exemple, la fonction SQL PI(*) renvoie la valeur de Pi. La classe d'API Java java.lang.Math possède un champ parallèle nommé PI qui renvoie la même valeur. Cependant, java.lang.Math possède également un champ nommé E qui renvoie la base du logarithme naturel, ainsi qu'une méthode qui calcule l'opération "reste" sur deux arguments comme le préconise la norme IEE754.

Quelles sont les classes Java supportées par l'API Java ?

Toutes les classes d'API Java ne sont pas supportées par la base de données. Certaines classes, par exemple le package java.awt, qui contient les composants d'interface utilisateur pour les applications, ne sont pas utilisables à l'intérieur d'un serveur de base de données. D'autres classes, notamment une partie de java.io, concernent l'écriture d'informations sur un disque, ce qui n'est pas non plus supporté dans l'environnement serveur de base de données. Reportez-vous au chapitre 12, "Rubriques de référence", pour obtenir la liste des classes supportées ou non.

Puis-je installer mes propres classes Java ?

Vous pouvez installer vos propres classes Java dans la base de données comme, par exemple, une classe Employee créé par l'utilisateur ou une classe Inventory conçue, écrite et compilée par un développeur à l'aide d'un compilateur Java.

Les classes Java définies par l'utilisateur peuvent contenir des informations et des méthodes. Une fois installé dans une base de données, Adaptive Server permet d'utiliser ces classes dans toutes les parties et les opérations de la base de données et d'exécuter leurs fonctionnalités (sous forme de méthodes de classe ou d'instance).

Puis-je accéder aux données à l'aide de Java ?

L'interface JDBC est une norme technique permettant d'accéder aux systèmes de base de données. Les classes JDBC sont conçues pour établir une connexion à une base de données, demander des données à l'aide d'instructions SQL et renvoyer des résultats pouvant être traités dans l'application cliente.

Vous pouvez vous connecter à Adaptive Server Enterprise à partir d'une application cliente via JDBC, à l'aide de jConnect ou d'une passerelle JDBC/ODBC. Adaptive Server intègre également un pilote JDBC interne qui permet aux classes Java installées dans une base de données d'utiliser des classes JDBC exécutant des instructions SQL.

Puis-je utiliser les mêmes classes sur le client et le serveur ?

Vous pouvez créer des classes Java pouvant être utilisées à différents niveaux d'une application d'entreprise. La même classe Java peut être intégrée à l'application cliente, au niveau intermédiaire ou à la base de données.

Comment utiliser les classes Java dans SQL ?

L'utilisation des classes Java, qu'elles soient définies par l'utilisateur ou à partir de l'API Java, s'effectue en trois étapes :

1 Écrivez ou récupérez un ensemble de classes Java que vous souhaitez utiliser en tant que types de données SQL ou en tant qu'alias SQL pour les méthodes statiques.

2 Installez ces classes dans la base de données Adaptive Server.

3 Utilisez ces classes en code SQL :

- Appelez les méthodes (statiques) de ces classes en tant que fonctions utilisateur.

- Déclarez les classes Java comme types de données de colonnes, de variables et de paramètres SQL. Dans ce manuel, ces éléments sont nommés colonnes, variables et paramètres Java-SQL.
- Référencez les colonnes Java-SQL, leurs champs et leurs méthodes.
- Encapsulez les méthodes statiques dans des alias SQL et utilisez-les comme s'il s'agissait de fonctions ou de procédures stockées.

Où puis-je trouver des informations relatives à Java dans la base de données ?

De nombreux manuels traitent de Java en général et de Java dans la base de données. Deux sont particulièrement utiles :

- James Gosling, Bill Joy, Guy Steele et Gilad Bracha, *The Java™ Language Specification, Second Edition*, Addison-Wesley, 2000.
- Seth White, Maydene Fisher, Rick Cattell, Graham Hamilton et Mark Hapner, *JDBC™ API Tutorial and Reference*, Second Edition, Addison-Wesley, 1999.

Que ne permet pas de faire Java dans la base de données ?

Adaptive Server est un environnement d'exécution Java et non un environnement de développement Java.

Ainsi, vous ne pouvez pas exécuter les tâches suivantes dans la base de données :

- Éditer les fichiers source des classes (fichiers **.java*).
- Compiler les fichiers source des classes Java (fichiers **.java*).
- Exécuter des API Java non supportées, comme des applets et des classes visuelles.
- Utiliser des threads Java. Adaptive Server ne supporte ni java.lang.Thread, ni java.lang.ThreadGroup. Si vous tentez de générer un thread, Adaptive Server déclenche une exception java.lang.UnsupportedOperationException.
- Utiliser des objets Java en tant que paramètres transmis vers ou depuis un appel de procédure à distance. Ils ne sont pas convertis correctement.

- Sybase vous déconseille d'utiliser des variables statiques dans des méthodes référencées par les fonctions Java-SQL, les fonctions SQLJ ou les procédures stockées SQLJ. Les valeurs renvoyées par ces variables risquent d'être peu fiables dans la mesure où la portée d'une variable statique dépend de la mise en œuvre.

Exemples de classes Java

Dans le présent manuel, des classes Java simples ont été utilisées pour illustrer les principes de base de l'emploi de Java dans la base de données. Vous trouverez des copies de ces classes dans les chapitres qui les décrivent et dans le répertoire d'installation de Sybase sous *$SYBASE/$SYBASE_ASE/sample/JavaSql* (UNIX) ou *%SYBASE%\Ase-12_5\sample\JavaSql* (Windows NT). Ce sous-répertoire contient également des utilitaires Javadoc pour visualiser des spécifications sur des classes et des méthodes exemples à l'aide de votre explorateur Web.

CHAPITRE 2

Préparation et maintenance de Java dans la base de données

Ce chapitre décrit l'environnement d'exécution Java, explique comment activer Java sur le serveur et fournit les procédures d'installation et de maintenance des classes Java dans la base de données.

Rubrique	Page
Environnement d'exécution Java	13
Activation de la gestion des objets Java par le serveur	15
Création de classes Java et de fichiers JAR	16
Installation de classes Java dans la base de données	17
Visualisation des informations sur les classes installées et les fichiers JAR	21
Téléchargement des informations sur les classes installées et les fichiers JAR	22
Suppression de classes et de fichiers JAR	22

Environnement d'exécution Java

L'environnement d'exécution Java d'Adaptive Server requiert une machine virtuelle Java, qui est intégrée au serveur de base de données, et les classes Java nécessaires à l'installation Sybase, ou l'API Java. Si vous exécutez des applications Java sur le client, vous aurez probablement besoin du pilote JDBC de Sybase, jConnect, sur le client.

Classes Java dans la base de données

Vous pouvez utiliser des classes Java provenant de l'une des sources suivantes :

- des classes Java nécessaires à l'installation Sybase ;
- des classes définies par l'utilisateur.

Classes Java nécessaires à l'installation Sybase

La machine virtuelle Java de Sybase supporte un sous-ensemble de classes et de packages de JDK version 2.0 (UNIX et Windows NT).

Les classes Java nécessaires à l'installation Sybase sont des classes de bas niveau installées pour permettre à une base de données de contenir des objets Java. Elles sont copiées automatiquement lors de l'installation d'Adaptive Server et sont ensuite disponibles sous *$SYBASE/$SYBASE_ASE/lib/runtime.zip* (UNIX) ou *%SYBASE%\%SYBASE_ASE%\lib\runtime.zip* (Windows NT). Vous n'avez pas besoin de définir la variable d'environnement CLASSPATH pour Java dans Adaptive Server.

Sybase ne supporte pas les packages et les classes d'exécution Java qui nécessitent un affichage à l'écran, interviennent dans les interconnexions réseau ou les communications à distance ou gèrent la sécurité. Reportez-vous au chapitre 12, "Rubriques de référence" pour obtenir la liste des packages et des classes supporté.

Classes Java définies par l'utilisateur

Installez les classes définies par l'utilisateur dans la base de données avec l'utilitaire installjava. Une fois installées, ces classes sont mises à la disposition des autres classes de la base de données et du système SQL en tant que types de données définis par l'utilisateur.

Pilotes JDBC

Le pilote JDBC interne de Sybase, livré avec Adaptive Server supporte JDBC version 1.2. Il est compatible et supporte plusieurs classes et méthodes de JDBC version 2.0. Reportez-vous au chapitre 12, "Rubriques de référence", pour obtenir la liste des packages et des classes supportés.

Si votre système requiert un pilote JDBC sur le client, utilisez jConnect version 5.2 ou supérieure, qui supporte JDBC version 2.0.

Machine virtuelle Java

Sybase fournit une machine virtuelle Java pour assurer l'exécution rapide de chaque méthode appelée. La machine virtuelle Java est exécutée sur le serveur. Elle requiert peu ou aucune administration une fois son installation terminée.

Configuration de la mémoire pour Java dans la base de données

Utilisez la procédure système sp_configure pour modifier l'allocation de mémoire pour Java dans Adaptive Server. Vous pouvez modifier l'allocation de mémoire pour :

- size of global fixed heap – spécifie l'espace mémoire pour les structures de données internes.
- size of process object fixed heap – spécifie l'espace mémoire total pour toutes les connexions des utilisateurs via la machine virtuelle Java.
- size of shared class heap – spécifie l'espace mémoire partagé pour toutes les classes Java appelées dans la machine virtuelle Java.

Pour une description complète de ces paramètres de configuration, reportez-vous au *Guide d'administration système*.

Activation de la gestion des objets Java par le serveur

Pour que le serveur et ses bases de données gèrent les objets Java, tapez la commande suivante à partir de isql :

```
sp_configure "enable java", 1
```

Puis arrêtez et redémarrez le serveur.

Par défaut, Adaptive Server ne gère pas les objets Java. Vous ne pouvez pas installer de classe Java ni exécuter des opérations Java tant que la gestion des objets Java n'a pas été activée sur le serveur.

Vous pouvez augmenter ou diminuer la quantité de mémoire disponible pour Java dans Adaptive Server et optimiser les performances à l'aide de la procédure système sp_configure. Les paramètres de configuration Java sont décrits dans le document *Guide d'administration système*.

Désactivation de la gestion des objets Java par le serveur

Pour désactiver la gestion des objets Java dans la base de données, tapez la commande suivante à partir de isql :

```
sp_configure "enable java", 0
```

Création de classes Java et de fichiers JAR

Les classes du JDK supportées par Sybase sont installées sur le système en même temps qu'Adaptive Server version 12 ou supérieure. Cette section décrit les étapes à suivre pour créer et installer vos propres classes Java.

Pour rendre vos classes Java (ou des classes provenant d'autres sources) disponibles sur le serveur, suivez les étapes ci-dessous :

1 Écrivez et sauvegardez le code Java de définition des classes.

2 Compilez le code Java.

3 Créez des fichiers d'archivage Java (JAR) pour organiser et stocker vos classes.

4 Installez les fichiers JAR/les classes dans la base de données.

Ecriture du code Java

Pour écrire le code Java destiné aux déclarations de classes, utilisez Sun Java SDK ou un outil de développement tel que Sybase PowerJ. Sauvegardez le code Java dans un fichier avec l'extension *.java*. Le nom et la casse du fichier doivent être les mêmes que ceux de la classe.

Remarque Assurez-vous que toutes les classes d'API Java utilisées par vos classes sont répertoriées dans la liste des classes d'API supportées, qui figure au chapitre 12, "Rubriques de référence"

Compilation du code Java

Cette étape permet de convertir la déclaration de classe contenant le code Java en un nouveau fichier distinct contenant le code octet. Le nom du nouveau fichier est le même que celui du fichier de code Java, à ceci près qu'il possède l'extension *.class*. Vous pouvez exécuter une classe Java compilée dans un environnement d'exécution Java quels que soient la plate-forme ayant servi à la compilation et le système d'exploitation utilisé pour son exécution.

Sauvegarde des classes dans un fichier JAR

Vous pouvez organiser vos classes Java en regroupant les classes associées dans des packages et en les stockant dans des fichiers JAR. Les fichiers JAR permettent d'installer ou de supprimer des classes associées en tant que groupe.

Installation de fichiers JAR non compressés

Pour installer des classes Java dans une base de données, les classes ou les packages doivent être sauvegardés dans un fichier JAR non compressé. Pour créer un fichier JAR non compressé contenant des classes Java, utilisez la commande Java jar cf0 ("zéro").

Dans cet exemple UNIX, la commande jar crée un fichier JAR non compressé contenant tous les fichiers *.class* du répertoire jcsPackage :

```
jar cf0 jcsPackage.jar jcsPackage/*.class
```

Installation de fichiers JAR compressés

Vous pouvez également installer un fichier JAR compressé si vous le décompressez tout d'abord à l'aide de l'option x de la commande jar. Dans cet exemple UNIX, abcPackage est un fichier compressé.

1 Placez le fichier JAR compressé dans un répertoire vide et décompressez-le :

```
jar xf0 abcPackage.jar
```

2 Supprimez le fichier JAR compressé pour qu'il ne soit pas inclus dans le nouveau fichier JAR non compressé :

```
rm abcPackage.jar
```

3 Créez le fichier JAR non compressé :

```
jar cf0 abcPackage.jar*
```

Installation de classes Java dans la base de données

Pour installer des classes Java depuis un fichier de système d'exploitation client, utilisez l'utilitaire installjava (UNIX) ou instjava (Windows NT) à partir de la ligne de commande.

Reportez-vous au guide *Utilitaires d'Adaptive Server Enterprise* pour plus d'informations sur ces utilitaires. Les deux utilitaires exécutent les mêmes tâches ; pour plus de simplicité, ce document utilise des exemples UNIX.

Utilisation de *installjava*

L'utilitaire installjava copie un fichier JAR dans le système Adaptive Server et permet à la base de données courante d'utiliser les classes Java contenues dans le fichier JAR. Respectez la syntaxe suivante :

installjava
-f *nom_fichier*
[-new | -update]
[-j *nom_jar*]
[-S *nom_serveur*]
[-U *nom_utilisateur*]
[-P *mot_de_passe*]
[-D *nom_basededonnées*]
[-I *fichier_interface*]
[-a *jeu_car_affichage*]
[-J *jeu_car_client*]
[-z *langue*]
[-t *temporisation*]

Par exemple, pour installer des classes contenues dans le fichier *addr.jar*, tapez :

```
installjava -f "/home/usera/jars/addr.jar"
```

Le paramètre -f désigne un fichier de système d'exploitation contenant un fichier JAR. Spécifiez le chemin d'accès complet à ce fichier.

Cette section décrit l'enregistrement des fichiers JAR (à l'aide du -j) et la mise à jour des fichiers JAR et des classes installées (à l'aide de new et de update). Pour plus d'informations sur ces opérations et sur les autres options disponibles avec l'utilitaire installjava, reportez-vous au guide *Utilitaires*.

Remarque Lorsque vous installez un fichier JAR, Application Server copie le fichier dans une table temporaire et l'installe à partir de là. Si vous installez un fichier JAR volumineux, vous devrez augmenter la taille de tempdb à l'aide de la commande alter database.

Enregistrement du fichier JAR

Lorsqu'un fichier JAR est installé dans une base de données, le serveur le désassemble, extrait les classes et les stocke séparément. Le fichier JAR n'est pas stocké dans la base de données, à moins de spécifier installjava avec le paramètre -j.

Le paramètre -j permet de déterminer si le système Adaptive Server conserve le fichier JAR spécifié dans installjava ou s'il ne l'utilise que pour extraire les classes à installer.

- Si vous spécifiez le paramètre -j, Adaptive Server installe les classes contenues dans le fichier JAR normalement, puis conserve le fichier JAR et son association avec les classes installées.
- Si vous ne spécifiez pas le paramètre -j, Adaptive Server ne conserve aucune association entre le fichier JAR et ses classes. Il s'agit de l'option par défaut.

Sybase recommande de spécifier un nom pour le fichier JAR pour vous permettre de mieux gérer les classes installées. Si vous enregistrez le fichier JAR :

- Vous pouvez le supprimer avec toutes les classes qui lui sont associées en une seule opération, au moyen de l'instruction remove java. Sinon, vous devez supprimer individuellement chaque classe ou chaque groupe de classes.
- Vous pouvez utiliser extractjava pour télécharger le fichier JAR dans un fichier de système d'exploitation. Reportez-vous à la section "Téléchargement des informations sur les classes installées et les fichiers JAR", page 22.

Mise à jour des classes installées

Les clauses new et update de installjava permettent d'indiquer si les nouvelles classes doivent remplacer ou non celles déjà installées.

- Si vous spécifiez new, vous ne pouvez pas installer une classe portant le même nom qu'une classe existante.

- Si vous spécifiez update, vous pouvez installer une classe portant le même nom qu'une classe existante. La classe nouvellement installée remplace alors la classe existante.

  Avertissement ! Si vous modifiez une classe utilisée comme type de données colonne en réinstallant une version modifiée de la classe, assurez-vous que la classe modifiée peut lire et utiliser les objets existants (lignes) dans les tables où cette classe apparaît comme type de données. Sinon, il vous sera impossible d'accéder aux objets existants, à moins de réinstaller la classe.

Le remplacement de classes installées par de nouvelles classes s'effectue différemment selon que les classes en cours d'installation ou les classes déjà installées sont associées ou non à un JAR. Ainsi :

- Si vous mettez à jour un JAR, toutes les classes qui y figurent sont supprimées et remplacées par les classes du nouveau JAR.
- Une classe ne peut être associée qu'à un seul JAR. Vous ne pouvez pas installer une classe d'un JAR si une classe homonyme est déjà installée et associée à un autre JAR. De même, vous ne pouvez pas installer une classe en tant que classe non associée à un JAR, si elle est déjà installée et associée à un autre JAR.

  Cependant, vous pouvez installer une classe d'un JAR enregistré portant le même nom qu'une classe installée non associée à un JAR. Dans ce cas, la classe non associée est supprimée et la nouvelle classe homonyme est associée au nouveau JAR.

Si vous souhaitez réorganiser des classes installées dans de nouveaux JAR, il est plus simple de commencer par les dissocier des JAR où elles se trouvent. Pour plus d'informations, reportez-vous à la section "Enregistrement des classes", page 23.

Référencement d'autres classes Java-SQL

Les classes installées peuvent référencer d'autres classes dans le même fichier JAR ainsi que des classes précédemment installées dans la même base de données, mais elles ne peuvent pas référencer les classes d'autres bases de données.

Si les classes d'un fichier JAR référencent des classes non définies, une erreur risque de se produire :

- Si une classe non définie est référencée directement dans SQL, une erreur de syntaxe est générée pour la "classe non définie".
- Si une classe non définie est référencée à l'intérieur d'une méthode Java appelée auparavant, elle génère une exception Java qui peut être détectée dans la méthode Java appelée ou peut entraîner l'exception SQL générale décrite à la section "Exceptions dans les méthodes Java-SQL", page 35.

La définition d'une classe peut contenir des références à des classes et des méthodes non supportées, à condition qu'elles ne soient pas référencées, ni appelées de façon active. De même, une classe installée peut contenir une référence à une classe utilisateur non installée dans la même base de données à condition que la classe ne soit ni instancée ni référencée.

Visualisation des informations sur les classes installées et les fichiers JAR

Pour visualiser les informations sur les classes et les JAR installés dans la base de données, utilisez : Respectez la syntaxe suivante :

```
sp_helpjava ['class' [, nom [, 'detail' | , 'depends' ]] |
      'jar' [, nom [, 'depends' ] ]]
```

Par exemple, pour visualiser des informations détaillées sur la classe Address, connectez-vous à isql et tapez :

```
sp_helpjava "class", Address, detail
```

Pour plus d'informations, reportez-vous à la section "sp_helpjava" du document *Manuel de référence d'Adaptive Server Enterprise*.

Téléchargement des informations sur les classes installées et les fichiers JAR

Vous pouvez télécharger des copies de classes Java installées sur une base de données pour les utiliser dans d'autres bases de données ou dans des applications.

L'utilitaire système extractjava permet de télécharger un fichier JAR et ses classes vers un fichier de système d'exploitation client. Par exemple, pour télécharger *addr.jar* vers *~/home/usera/jars/addrcopy.jar*, tapez :

```
extractjava –j 'addr.jar' -f
    '~/home/usera/jars/addrcopy.jar'
```

Reportez-vous au guide *Utilitaires* pour plus d'informations.

Suppression de classes et de fichiers JAR

Utilisez l'instruction Transact-SQL remove java pour désinstaller une ou plusieurs classes Java-SQL de la base de données. remove java peut spécifier un ou plusieurs noms de classes Java, de packages Java ou de fichiers JAR enregistrés. Par exemple, pour désinstaller le package utilityClasses, à partir de isql, tapez :

```
remove java package "utilityClasses"
```

Remarque Adaptive Server ne permet pas la suppression de classes utilisées comme types de données de colonnes et de paramètres, ou référencés par des fonctions SQLJ ou des procédures stockées.

Veillez à ne pas supprimer de sous-classes ni de classes utilisées comme variables ou comme types de résultats de fonctions utilisateur.

Lorsque vous spécifiez remove java package, la commande supprime toutes les classes du package spécifié et tous les sous-packages.

Pour plus d'informations sur remove java, reportez-vous au document *Manuel de référence*.

Enregistrement des classes

Vous pouvez supprimer un fichier JAR de la base de données mais conserver ses classes en tant qu'objets qui ne sont plus associés à un fichier JAR. Utilisez remove java avec l'option retain classes si, par exemple, vous souhaitez réorganiser le contenu de plusieurs fichiers JAR enregistrés.

Par exemple, à partir de isql, tapez :

```
remove java jar 'utilityClasses' retain classes
```

Une fois les classes dissociées de leurs fichiers JAR, vous pouvez les associer à d'autres fichiers JAR via installjava avec le mot-clé new.

CHAPITRE 3

Utilisation des classes Java avec le langage SQL

Ce chapitre explique comment utiliser les classes Java dans un environnement Adaptive Server. Les premières sections vous donnent suffisamment d'informations pour démarrer ; les sections suivantes fournissent des informations plus avancées.

Rubriques	Page
Concepts généraux	26
Utilisation des classes Java comme types de données	27
Appel des méthodes Java en SQL	33
Représentation des instances Java	35
Propriétés de l'affectation des éléments de données Java-SQL	36
Mappage de type de données entre les champs Java et SQL	38
Jeux de caractères pour les données et les identificateurs	40
Sous-types Java-SQL	40
Traitement des valeurs NULL dans les données Java-SQL	43
Données de type String Java-SQL	47
Méthodes typées et void	48
Opérations d'égalité et d'ordre	51
Ordre d'évaluation et appels de méthode Java	52
Variables statiques dans les classes Java-SQL	53
Classes Java dans plusieurs bases de données	54
Classe Java	58

Dans ce document, les colonnes et les variables SQL dont les types de données sont des classes Java-SQL sont décrites comme des colonnes Java-SQL et des variables Java-SQL ou comme des éléments de données Java-SQL.

Vous trouverez les exemples de classe utilisés dans ce chapitre dans les répertoires suivants :

- *$SYBASE/$SYBASE_ASE/sample/JavaSql* (UNIX)
- *%SYBASE%\Ase-12_5\sample\JavaSql* (Windows NT)

Concepts généraux

Cette section fournit des informations d'ordre général sur les identificateurs Java et Java-SQL.

Considérations relatives à Java

Avant d'utiliser Java dans la base de données Adaptive Server, veuillez lire les considérations générales ci-après.

- Les classes Java contiennent :
  - des champs déclarés comme types de données Java ;
  - des méthodes dont les paramètres et les résultats ont été déclarés comme types de données Java ;
  - des types de données Java auxquels correspondent des types de données SQL, et qui sont définis à la section "Mappage de type de données entre Java et SQL", page 232.
- Les classes Java-SQL peuvent inclure des classes, des champs et des méthodes typées private, protected, friendly ou public.

  Les classes, les champs et les méthodes typées public peuvent être référencés en SQL. Les classes, les champs et les méthodes typées private, protected ou friendly ne peuvent pas être référencés en SQL, mais peuvent l'être dans Java et sont soumises aux règles Java d'usage.
- Les classes, les champs et les méthodes Java ont des propriétés syntaxiques :
  - Classes : le nombre de champs et leurs noms.
  - Champ : leurs types de données.
  - Méthodes : le nombre de paramètres et leurs types de données, et le type de données du résultat.

  Le système SQL détermine ces propriétés syntaxiques à partir des classes Java-SQL, à l'aide de l'API Java Reflection.

Noms Java-SQL

Les noms de classe Java-SQL (identificateurs) sont limités à 255 octets. La longueur des noms de champ et de méthode Java-SQL n'est pas limitée, mais si vous les utilisez dans Transact-SQL, elle ne doit pas dépasser 255 octets. Tous les noms Java-SQL doivent être conformes aux règles relatives aux identificateurs Transact-SQL si vous les utilisez dans des instructions Transact-SQL.

Les noms de classe, de champ et de méthode de 30 octets ou plus doivent être entre guillemets.

Le premier caractère du nom doit être un caractère alphabétique (en majuscule ou en minuscule) ou un tiret bas (_). Les caractères suivants peuvent être des caractères alphabétiques, des chiffres, des symboles dollars ($) ou des tirets bas (_).

Les noms Java-SQL distinguent toujours les majuscules des minuscules, que le système SQL ait été défini pour respecter les majuscules/minuscules ou non.

Pour plus d'informations sur les identificateurs, reportez-vous à la section Identificateurs Java-SQL, page 233.

Utilisation des classes Java comme types de données

Une fois que vous avez installé un ensemble de classes Java, vous pouvez les référencer comme des types de données dans SQL. Pour être utilisée comme un type de données pour une colonne, une classe Java-SQL doit être définie comme public et doit utiliser java.io.Serializable ou java.io.Externalizable.

Vous pouvez spécifier des classes Java-SQL en tant que :

- types de données des colonnes SQL ;
- types de données des variables Transact-SQL et paramètres des procédures stockées Transact-SQL ;
- valeurs par défaut pour les colonnes SQL.

Lorsque vous créez une table, vous pouvez spécifier des classes Java-SQL comme types de données de colonnes SQL :

```
create table emps (
       name varchar(30),
       home_addr Address,
       mailing_addr Address2Line null )
```

La colonne name est une chaîne de caractères SQL ordinaire, les colonnes home_addr et mailing_addr peuvent contenir des objets Java. Address et Address2Line sont des classes Java-SQL installées dans la base de données.

Vous pouvez spécifier des classes Java-SQL comme des types de données de variables Transact-SQL :

```
declare @A Address
declare @A2 Address2Line
```

Vous pouvez également spécifier des valeurs par défaut pour les colonnes Java-SQL, à condition que ces valeurs soient des expressions constantes. Cette expression est normalement un appel de constructeur qui utilise l'opérateur new avec des arguments constants, comme dans l'exemple suivant :

```
create table emps (
       name varchar(30),
       home_addr Address default new Address
           ('Not known', ''),
       mailing_addr Address2Line
)
```

Création et modification des tables contenant des colonnes Java-SQL

Lorsque vous créez ou que vous modifiez des tables contenant des colonnes Java-SQL, vous pouvez spécifier n'importe quelle classe Java installée comme type de données pour une colonne. Vous pouvez également spécifier le mode de stockage des informations dans la colonne. La vitesse à laquelle les champs de ces colonnes sont référencés et mis à jour par Adaptive Server dépend des options de stockage choisies.

Les valeurs des colonnes d'une ligne sont normalement stockées "dans la ligne", c'est-à-dire consécutivement sur les pages de données allouées à une table. Cependant, vous pouvez choisir de stocker des colonnes Java-SQL dans un emplacement "hors de la ligne" distinct, comme pour les éléments de données text et image. Par défaut, les valeurs des colonnes Java-SQL sont stockées hors de la ligne.

Si une colonne Java-SQL est stockée dans la ligne :

- Le traitement des objets stockés dans la ligne est plus rapide que celui des objets stockés hors de la ligne.

- Un objet stocké dans la ligne peuvent occuper jusqu'à 16 ko environ, selon la taille de page du serveur de base de données et autres variables. Ceci inclut sa sérialisation complète, et pas seulement les valeurs qui se trouvent dans ses champs. Un objet Java dont la représentation lors de l'exécution occupe plus de 16 K génère une exception et la commande échoue.

Si une colonne Java-SQL est stockée hors de la ligne, elle est soumise aux restrictions qui s'appliquent aux colonnes text et image :

- Le traitement des objets stockés hors de la ligne est plus lent que celui des objets stockés dans la ligne.
- La taille d'un objet stocké hors de la ligne est soumise aux restrictions habituelles applicables aux colonnes text et image.
- Une colonne hors de la ligne ne peut pas être référencée dans une contrainte de vérification.

  Il en est de même pour une table qui contient une colonne hors de la ligne. Adaptive Server permet d'inclure la contrainte de vérification lorsque vous créez ou modifiez la table, mais il émet un avertissement lors de la compilation, puis ignore la contrainte lors de l'exécution.
- Une colonne hors de la ligne ne peut pas être incluse dans la liste de sélection des colonnes d'une requête select avec select distinct.
- Une colonne hors de la ligne ne peut pas être utilisée avec un opérateur de comparaison, un prédicat ni une clause group by.

La syntaxe partielle de create table avec l'option in row/off row est la suivante :

```
create table...column_name datatype
      [default {expression_constante | user | null}]
 {[{identity | null | not null}]
      [off row | [ in row [ ( taille_en_octets ) ] ]...
```

La variable *taille_en_octets* indique la taille maximale de la colonne dans la ligne. Sa valeur peut être inférieure ou égale à 16 ko. La valeur par défaut est 255 octets.

La taille maximale de la colonne dans la ligne que vous spécifiez dans l'instruction create table doit inclure la sérialisation complète de la colonne, et pas seulement les valeurs qui se trouvent dans ses champs, ainsi que les valeur minimales de l'overhead.

Utilisez la fonction système datalength pour connaître la taille approximative d'une colonne qui comprend les valeurs d'overhead et de sérialisation. datalength permet de déterminer la taille réelle d'un objet représentatif que vous voulez stocker dans la colonne.

Par exemple :

```
select datalength (new nom_classe(...))
```

où *nom_classe* désigne une classe Java-SQL installée.

La syntaxe partielle de alter table se présente comme suit :

```
alter table...{add column_name datatype
 [default {expression_constante | user | null}]
 {identity | null} [off row | in row]...
```

Remarque Il est impossible de modifier la taille d'une colonne dans la ligne à l'aide de l'instruction alter column de cette version d'Adaptive Server.

Modification des tables partitionnées

Lorsqu'une table qui contient des colonnes Java est partitionnée, vous ne pouvez pas la modifier sans supprimer au préalable les partitions. Pour modifier le schéma de la table :

1 Supprimez la partition.

2 Exécutez la commande alter table.

3 Repartitionnez la table.

Sélection, insertion, mise à jour et suppression des objets Java

Une fois que vous avez spécifié des colonnes Java-SQL, les valeurs que vous affectez à ces éléments de données doivent être des instances Java. Ces instances sont initialement générées par des appels à des constructeurs Java à l'aide de l'opérateur new. Vous pouvez générer des instances Java pour des colonnes et des variables.

Les constructeurs sont des méthodes de pseudo-instance qui permettent de créer des instances. Ils portent le même nom que la classe et n'ont pas de type de données déclaré. Si vous n'incluez pas un constructeur dans la définition de classe, une méthode par défaut est fournie par la classe de base Java object. Vous pouvez fournir plusieurs constructeurs pour chaque classe, avec des nombres et des types d'arguments différents. Lorsqu'un constructeur est appelé, celui qui possède le nombre et le type d'arguments corrects est utilisé.

Dans l'exemple suivant, des instances Java sont générées pour des colonnes et des variables :

```
declare @A Address, @AA Address, @A2 Address2Line,
       @AA2 Address2Line

 select @A = new Address( )
 select @AA = new Address('123 Main Street', '99123')
 select @A2 = new Address2Line( )
 select @AA2 = new Address2Line('987 Front Street',
       'Unit 2', '99543')

 insert into emps values('John Doe', new Address( ),
       new Address2Line( ) )
 insert into emps values('Bob Smith',
       new Address('432 ElmStreet', '99654'),
       new Address2Line('PO Box 99', 'attn: Bob Smith', '99678') )
```

Les valeurs affectées aux colonnes et aux variables Java-SQL peuvent ensuite être réaffectées à d'autres colonnes et variables Java-SQL. Par exemple :

```
declare @A Address, @AA Address, @A2 Address2Line,
       @AA2 Address2Line

 select @A = home_addr, @A2 = mailing_addr from emps
       where name = 'John Doe'
 insert into emps values ('George Baker', @A, @A2)

 select @AA2 = @A2
 update emps
       set home_addr = new Address('456 Shoreline Drive', '99321'),
         mailing_addr = @AA2
         where name = 'Bob Smith'
```

Vous pouvez également copier les valeurs des colonnes Java-SQL d'une table à une autre. Par exemple :

```
create table trainees (
       name char(30),
       home_addr Address,
       mailing_addr Address2Line null
 )
insert into trainees
select * from emps
       where name in ('Don Green', 'Bob Smith',
       'George Baker')
```

Vous pouvez référencer et mettre à jour les champs des colonnes et des variables Java-SQL avec une qualification SQL normale. Pour éviter toute ambiguïté avec les points utilisés par SQL pour qualifier les noms, utilisez les chevrons (>>) pour qualifier les noms de champ et de méthode Java lorsque vous les référencez en SQL.

```
declare @name varchar(100), @street varchar(100),
       @streetLine2 varchar(100), @zip char(10), @A Address

 select @A = new Address()
 select @A>>street = '789 Oak Lane'
 select @street = @A>>street

 select @street = home_add>>street, @zip = home_add>>zip from emps
       where name = 'Bob Smith'
 select @name = name from emps
       where home_addr>>street = '456 Shoreline Drive'

 update emps
       set home_addr>>street = '457 Shoreline Drive',
          home_addr>>zip = '99323'
       where home_addr>>street = '456 Shoreline Drive'
```

Appel des méthodes Java en SQL

Vous pouvez appeler les méthodes Java en SQL en les référençant, par qualification du nom, sur des instances pour les méthodes d'instance, et sur des instances ou des classes pour les méthodes statiques.

Les méthodes d'instance sont généralement étroitement liées aux données encapsulées dans une instance précise de leur classe. Les méthodes (de classe) statiques conditionnent la classe entière, et pas seulement une instance donnée de celle-ci. Les méthodes statiques s'appliquent souvent à des objets et à des valeurs appartenant à une gamme étendue de classes.

Une fois une méthode statique installée, elle est prête à être utilisée. Une classe contenant une méthode statique utilisable comme fonction doit être de type public, mais elle n'a pas besoin d'être sérialisable.

L'un des principaux avantages de Java avec Adaptive Server est que vous pouvez utiliser des méthodes statiques qui renvoient une valeur au demandeur sous forme de fonctions utilisateur.

Il est possible d'utiliser une méthode statique Java comme fonction utilisateur dans une procédure stockée, un trigger, une clause where et partout où les fonctions SQL intégrées sont admises.

Les méthodes Java appelées directement en SQL comme fonctions utilisateur sont soumises aux restrictions suivantes :

- Si la méthode Java accède à la base de données via JDBC, les valeurs des jeux de résultats sont disponibles uniquement pour la méthode Java, et non pour l'application cliente.
- Les paramètres de sortie ne sont pas supportés. Une méthode peut manipuler les données qu'elle reçoit d'une connexion JDBC, mais la seule valeur qu'elle peut renvoyer à son demandeur est une valeur de retour unique déclarée dans le cadre de sa définition.
- Les appels, portant sur plusieurs bases de données, de méthodes statiques sont supportés uniquement si vous utilisez une instance de classe comme valeur d'une colonne.

L'autorisation d'exécuter une fonction utilisateur quelconque est octroyée implicitement au groupe public. Si la fonction utilisateur exécute des requêtes SQL via JDBC, l'autorisation d'accès aux données est vérifiée par rapport à l'appelant de la fonction utilisateur. Ainsi, lorsque l'utilisateur A appelle une fonction utilisateur qui accède à la table t1, il doit avoir l'autorisation select sur t1, à défaut de quoi la requête échoue. Pour plus d'informations sur les modèles de sécurité pour les appels de méthode Java, reportez-vous à la section "Sécurité et autorisations", page 85.

Si vous souhaitez utiliser les méthodes statiques Java pour renvoyer des jeux de résultats et des paramètres de sortie, encapsulez les méthodes dans SQL et appelez-les comme des fonctions ou des procédures stockées SQLJ. Pour obtenir un comparatif des différentes procédures d'appel des méthodes Java dans Adaptive Server, reportez-vous à la section "Appel des méthodes Java dans Adaptive Server", page 87.

Exemples de méthodes

Les classes exemples Address et Address2Line possèdent des méthodes d'instance nommées toString( ) et la classe exemple Misc possède des méthodes statiques nommées stripLeadingBlanks( ), getNumber( ) et getStreet( ). Vous pouvez appeler des méthodes de valeur en tant que fonctions dans une expression de valeur.

```
declare @name varchar(100)
declare @street varchar(100)
declare @streetnum int
declare @A2 Address2Line

select @name = Misc.stripLeadingBlanks(name),
          @street = Misc.stripLeadingBlanks(home_addr>>street),
          @streetnum = Misc.getNumber(home_addr>>street),
          @A2 = mailing_addr
       from emps
       where home_addr>>toString( ) like '%Shoreline%'
```

Pour plus d'informations sur les méthodes void (méthodes pour lesquelles aucune valeur n'est renvoyée), reportez-vous à la section "Méthodes typées et void", page 48.

Exceptions dans les méthodes Java-SQL

Lorsque l'appel d'une méthode Java-SQL donne lieu à des exceptions non gérées, une exception SQL est générée et le message d'erreur suivant s'affiche :

```
Unhandled Java method exception
```

Le texte du message relatif à l'exception comprend le nom de la classe Java qui a généré l'exception, suivi de la chaîne de caractères (si elle existe) fournie lorsque l'exception Java a été générée.

Représentation des instances Java

Les clients non-Java tels que isql ne peuvent pas recevoir des objets Java sérialisés du serveur. Afin de vous permettre d'afficher et d'utiliser l'objet, Adaptive Server le convertit sous une forme compatible.

Pour utiliser une valeur de chaîne réelle, Adaptive Server appelle une méthode qui convertit l'objet en une valeur char ou varchar. La méthode toString() de la classe Address constitue un exemple de ce type de méthode. Il convient de créer sa propre version de la méthode toString() afin de pouvoir manipuler la représentation compatible de l'objet.

Remarque La méthode toString() de l'API Java ne convertit pas l'objet en une représentation compatible. La méthode toString() que vous créez supplante la méthode toString() de l'API Java.

Lorsque vous utilisez la méthode toString( ), Adaptive Server impose un nombre maximal d'octets pouvant être renvoyés. Adaptive Server tronque la représentation imprimable de l'objet à la valeur de la variable globale @@stringsize. Vous pouvez modifier la valeur par défaut de @@stringsize (50) à l'aide de la commande set stringsize. Par exemple :

```
set stringsize 300
```

Le logiciel d'affichage de l'ordinateur peut tronquer l'élément de données à un nombre de caractères inférieur pour que la chaîne s'affiche à l'écran sans renvoi à la ligne.

Si vous incluez une méthode toString( ) ou une méthode similaire dans chaque classe, vous pouvez renvoyer la valeur de la méthode toString( ) de l'objet en procédant de l'une des manières suivantes :

- Soit en sélectionnant un champ particulier dans la colonne Java-SQL, ce qui appelle automatiquement la méthode toString( ) :

```
select home_addr>>street from emps
```

- Soit en sélectionnant la colonne et la méthode toString( ) qui concatène dans une chaîne toutes les valeurs des champs de la colonne :

```
select home_addr>>toString() from emps
```

Propriétés de l'affectation des éléments de données Java-SQL

En définitive, les valeurs affectées aux éléments de données Java-SQL sont dérivées des valeurs construites par les méthodes Java-SQL dans la machine virtuelle Java. Cependant, la représentation logique des variables, des paramètres et des résultats Java-SQL diffère de la représentation logique des colonnes Java-SQL.

- Les *colonnes* Java-SQL qui sont permanentes, sont des flux Java sérialisés stockés dans la ligne de regroupement de la table. Il s'agit de valeurs stockées contenant des représentations d'instances Java.
- Les *variables*, les *paramètres* et les *résultats de fonction* Java-SQL sont temporaires. Ils ne contiennent pas d'instances Java-SQL, mais plutôt des références aux instances Java contenues dans la machine virtuelle Java.

Ces différences de représentation sont à l'origine des différences de propriétés de l'affectation comme l'illustrent les exemples ci-après.

- Le constructeur Address avec l'opérateur new est évalué dans la machine virtuelle Java. Il construit une instance Address et lui renvoie une référence qui est assignée comme valeur de la variable Java-SQL *@A* :

```
declare @A Address, @AA Address, @A2 Address2Line,
       @AA2 Address2Line
select @A = new Address('432 Post Lane', '99444')
```

- La variable *@A* contient une référence à une instance Java dans la machine virtuelle Java. Cette référence est copiée dans la variable *@AA*. Les variables *@A* et *@AA* référencent maintenant la même instance.

```
select @AA=@A
```

- Cette affectation modifie le champ zip de l'instance Address référencée par la variable *@A*. L'instance Address est la même que celle référencée par la variable *@AA*. Aussi, les valeurs de *@A.zip* et *@AA.zip* sont maintenant toutes les deux '99222'.

```
select @A>>zip = '99222'
```

- Le constructeur Address avec l'opérateur new construit une instance Address et renvoie une référence à cette dernière. Cependant, étant donné que la cible est une colonne Java-SQL, le système SQL sérialise l'instance Address dénotée par cette référence et copie la valeur sérialisée dans la nouvelle ligne de la table emps.

```
insert into emps
       values ('Don Green', new Address('234 Stone
       Road', '99777'), new Address2Line( ) )
```

Le constructeur Address2Line fonctionne de la même manière que la méthode Address, à ceci près qu'elle renvoie une instance par défaut au lieu d'une instance avec des valeurs de paramètre spécifiées. Cependant, l'action effectuée est la même que pour l'instance Address. Le système SQL sérialise l'instance par défaut Address2Line et stocke la valeur sérialisée dans la nouvelle ligne de la table emps.

- L'instruction insert ne spécifie aucune valeur pour la colonne mailing_addr, de sorte que cette colonne sera définie à null, comme toutes les colonnes dont la valeur n'est pas spécifiée dans une instruction insert. Cette valeur NULL est générée en SQL, et l'initialisation de la colonne mailing_addr ne concerne pas la machine virtuelle Java.

```
insert into emps (name, home_addr) values ('Frank Lee', @A)
```

L'instruction insert spécifie que la valeur de la colonne home_addr doit être fournie par la variable Java-SQL *@A*. Cette variable contient une référence à une instance Address dans la machine virtuelle Java. Etant donné que la cible est une colonne Java-SQL, le système SQL sérialise l'instance Address dénotée par la variable *@A* et copie la valeur sérialisée dans la nouvelle ligne de la table emps.

- Cette instruction insère une nouvelle ligne emps pour 'Bob Brown'. La valeur de la colonne home_addr est fournie par la variable SQL *@A*. Il s'agit également d'une sérialisation de l'instance Java référencée par la variable *@A*.

```
insert into emps (name, home_addr) values ('Bob Brown', @A)
```

- Cette instruction update attribue au champ zip de la colonne home_addr de la ligne 'Frank Lee' la valeur '99777'. Ceci n'a aucun effet sur le champ zip de la ligne 'Bob Brown' qui conserve la valeur '99444'.

```
update emps
       set home_addr>>zip = '99777'
       where name = 'Frank Lee'
```

- La colonne Java-SQL home_addr contient une représentation sérialisée de la valeur d'une instance Address. Le système SQL appelle la machine virtuelle Java pour désérialiser cette représentation comme une instance Java de la machine virtuelle Java et renvoie une référence à la nouvelle copie désérialisée. Cette référence est affectée à la variable *@AA*. L'instance Address désérialisée référencée par la variable *@AA* est totalement indépendante de la valeur de la colonne et de l'instance référencée par la variable *@A*.

```
select @AA = home_addr from emps where name = 'Frank Lee'
```

- Cette affectation modifie le champ zip de l'instance Address référencée par la variable *@A*. Cette instance est une copie de la colonne home_addr de la ligne 'Frank Lee', mais elle est indépendante de la valeur de cette colonne. Aussi, l'affectation ne modifie pas le champ zip de la colonne home_addr de la ligne 'Frank Lee'.

```
select @A>>zip = '95678'
```

Mappage de type de données entre les champs Java et SQL

Lorsque vous transférez des données entre la machine virtuelle Java et Adaptive Server, quelle que soit la direction, vous devez tenir compte du fait que les types des éléments de données sont différents dans chaque système. Adaptive Server mappe automatiquement les éléments SQL sur les éléments Java et vice versa, conformément aux tables de correspondance décrites à la section "Mappage de type de données entre Java et SQL", page 232.

Ainsi, le type SQL char est converti en type Java String, le type SQL binary est converti en type Java byte[], etc.

- S'agissant des correspondances de type de données de SQL vers Java, les types char, varchar et varbinary, quelle que soit leur longueur, correspondent aux types de données Java String ou byte[ ].
- S'agissant des correspondances de type de données de Java vers SQL :
  - Les types de données Java String et byte[ ] correspondent aux types SQL varchar et varbinary, la longueur maximale de 16 ko étant définie par Adaptive Server.
  - Le type de données Java BigDecimal correspond au type SQL numeric (précision, échelle,), la précision et l'échelle étant définies par l'utilisateur.

Dans la table emps, la valeur maximale des classes Address et Address2Line et des champs street, zip et line2 est 255 octets (valeur par défaut). Le type de données Java de ces classes est java.String, mais elles sont traitées par le système SQL en tant que type de données varchar(255).

Une expression dont le type de données est un type d'objet Java n'est convertie dans le type de données SQL correspondant que lorsqu'elle est utilisée dans un contexte SQL. Par exemple, si le champ home_addr>>street pour l'employé 'Smith' compte 260 caractères et commence par '6789 Main Street ... :

```
select Misc.getStreet(home_addr>>street) from emps where name='Smith'
```

L'expression contenue dans la liste select transmet la valeur de 260 caractères de home_addr>>street à la méthode getStreet( ) (sans la tronquer à 255 caractères). La méthode getStreet( ) renvoie ensuite la chaîne de 255 caractères commençant par 'Main Street....'. Cette chaîne de 255 caractères est désormais un élément de la liste SQL select et est donc convertie dans le type de données SQL et (si nécessaire) tronquée à 255 caractères.

Jeux de caractères pour les données et les identificateurs

Le jeu de caractères Unicode est utilisé pour afficher le code source Java et les données Java String. Les champs des classes Java-SQL peuvent contenir des données Unicode.

Remarque Les identificateurs Java utilisés dans les noms qualifiés des classes visibles ou dans les noms de membre visibles ne peuvent utiliser que les caractères latins et les chiffres arabes.

Sous-types Java-SQL

Les sous-types de classe vous permettent d'utiliser les caractéristiques Java de substitution de sous-type et de remplacement de méthode. On parle de conversion élargissante lorsqu'une classe est convertie dans l'une de ses superclasses, et de conversion rétrécissante lorsqu'une classe est convertie dans l'une de ses sous-classes.

- Les conversions élargissantes sont exécutées implicitement avec des affectations et des comparaisons normales. Elles réussissent toujours, car chaque instance de sous-classe est également une instance de la superclasse.
- Les conversions rétrécissantes doivent être spécifiées avec des expressions convert explicites. Une conversion rétrécissante aboutit seulement si l'instance de superclasse est une instance de la sous-classe ou une sous-classe de la sous-classe. Sinon, une exception est générée.

Conversion élargissante

Vous n'avez pas besoin d'utiliser la fonction convert pour spécifier une conversion élargissante. Par exemple, étant donné que la classe Address2Line est une sous-classe de la classe Address, vous pouvez affecter des valeurs Address2Line aux éléments de données Address. Dans la table emps, la colonne home_addr est un type de données Address et la colonne mailing_addr est un type de données Address2Line :

```
update emps
    set home_addr = mailing_addr
    where home_addr is null
```

Dans toutes les lignes remplissant les conditions de la clause where, la colonne home_addr contient des données de type Address2Line, bien que le type déclaré de la colonne home_addr soit Address.

Ce type d'affectation traite implicitement une instance de classe comme une instance de superclasse de cette classe. Les instances d'exécution de la sous-classe conservent leurs types de données de sous-classe et leurs données associées.

Conversion rétrécissante

Vous devez utiliser la fonction convert pour convertir une instance de classe en une instance de sous-classe de cette classe. Par exemple :

```
update emps
       set mailing_addr = convert(Address2Line, home_addr)
       where mailing_addr is null
```

Les conversions rétrécissantes dans l'instruction update génèrent une exception lorsqu'elles sont appliquées à une colonne home_addr contenant une instance Address qui n'est pas de type Address2Line. Pour parer à de telles exceptions, incluez une condition dans la clause where :

```
update emps
       set mailing_addr = convert(Address2Line, home_addr)
       where mailing_addr is null
       and home_addr>>getClass( )>>toString( ) = 'Address2Line'
```

L'expression "home_addr>>getClass( )>>toString( )" appelle les méthodes getClass( ) et toString( ) de la classe Java Object. La classe Object est implicitement une superclasse de toutes les classes, de sorte que les méthodes définies pour elle sont disponibles pour toutes les classes.

Vous pouvez également utiliser une expression case :

```
update emps
       set mailing_addr =
          case
              when home_addr>>getClass( )>>toString( )
               ='Address2Line'
              then convert(Address2Line, home_addr)
              else null
          end
       where mailing_addr is null
```

Types de données d'exécution et de compilation

Ni les conversions élargissantes, ni les conversions rétrécissantes ne modifient la valeur d'instance réelle ou son type de données d'exécution ; elles spécifient simplement la classe à utiliser pour le type de compilation. Ainsi, lorsque vous stockez des valeurs Address2Line provenant de la colonne mailing_addr dans la colonne home_address, ces valeurs conservent le type d'exécution d'Address2Line.

Par exemple, la classe Address et la sous-classe Address2Line possèdent toutes deux la méthode toString( ) qui renvoie une forme String des données d'adresse complètes.

```
select name, home_addr>>toString( ) from emps
       where home_addr>>toString( ) not like '%Line2=[ ]'
```

Pour chaque ligne de emps, le type déclaré de la colonne home_addr est Address, mais le type d'exécution de la valeur home_addr est soit Address, soit Address2Line, selon l'effet de l'instruction update précédente. Pour les lignes dans lesquelles la valeur de la version d'exécution de la colonne home_addr est de type Address, la méthode toString( ) de la classe Address est appelée et, pour les lignes dans lesquelles la valeur de la version d'exécution de la colonne home_addr est Address2Line, la méthode toString( ) de la sous-classe Address2Line est appelée.

Pour obtenir une description des valeurs NULL dans les conversions élargissantes et rétrécissantes, reportez-vous à la section "Valeurs NULL lors de l'utilisation de la fonction SQL convert", page 46.

Traitement des valeurs NULL dans les données Java-SQL

Cette section traite de l'utilisation des valeurs NULL dans les éléments de données Java-SQL.

Références aux champs et aux méthodes des instances NULL

Si la valeur de l'instance spécifiée dans une référence de champ est NULL, la référence de champ est NULL. De même, si la valeur de l'instance spécifiée dans un appel de méthode d'instance vaut NULL, le résultat de l'appel est NULL.

L'effet d'une référence sur un champ ou une méthode d'une instance inexistante est régi par différentes règles Java. Dans le système Java, si vous tentez de référencer un champ d'une instance NULL, une exception est générée.

Par exemple, supposons que la table emps se compose des lignes suivantes :

```
insert into emps (name, home_addr)
    values ("Al Adams",
    Address home = new Address("123 Main", "98765");

insert into emps (name, home_addr)
    values ("Bob Baker",
    new Address("456 Side", "95123"))

insert into emps (name, home_addr)
    values ("Carl Carter", null)
```

Dans le cas de la commande select suivante :

```
select name, home_addr>>zip from emps
where home_addr>>zip in ('95123', '95125', '95128')
```

Si la règle Java est utilisée pour les références à "home_addr>>zip", celles-ci génèrent une exception pour la ligne "Carl Carter" dont la colonne "home_addr" prend la valeur NULL. Pour parer à une telle exception, écrivez l'instruction select suivante :

```
select name,
    case when home_addr is not null then home_addr>>zip
    else null end

from emps
    where case when home_addr is not null
    then home_addr>>zip
else
    null end
in ('95123', '95125', '95128')
```

Ainsi, la convention SQL est utilisée pour les références aux champs et aux méthodes des instances NULL : si l'instance est NULL, toutes les références de champ et de méthode le sont également. La règle SQL a pour effet de rendre implicite l'instruction case ci-dessus.

Cependant, cette règle SQL relative aux références de champ avec des instances NULL ne s'applique qu'aux références de champ dans des contextes source (à droite), et non aux références de champ qui sont des cibles (à gauche) d'affectation ou des clauses set. Par exemple :

```
update emps
    set home_addr>>zip = '99123'
    where name = 'Charles Green'
```

La clause where est vraie pour la ligne 'Charles Green', de sorte que l'instruction update tente d'exécuter la clause set. Ceci génère une exception, car vous ne pouvez pas affecter de valeur au champ d'une instance NULL, celle-ci n'ayant aucun champ auquel affecter une valeur. Ainsi, les références aux champs d'instances NULL sont corrects et renvoient la valeur NULL dans des contextes sources et génèrent des exceptions dans des contextes cibles.

Les mêmes considérations s'appliquent aux appels de méthodes d'instances NULL et la même règle est appliquée. Par exemple, en modifiant l'exemple précédent et en appelant la méthode toString( ) de la colonne home_addr :

```
select name, home_addr>>toString( ) from emps
    where home_addr>>toString( ) D
    'StreetD234 Stone Road ZIPD 99777'
```

Si la valeur de l'instance spécifiée dans un appel de méthode d'instance est NULL, le résultat de l'appel est NULL. Donc, l'instruction select est correcte ici, alors qu'elle génère une exception dans le système Java.

Valeurs NULL utilisées comme arguments des méthodes Java-SQL

Lorsque vous utilisez des valeurs NULL comme paramètres, le résultat obtenu est indépendant des actions de la méthode qui les prend comme arguments, mais dépendant de la capacité du type de données renvoyé à représenter la valeur NULL.

Vous ne pouvez pas passer de la valeur NULL comme paramètre à une méthode typée scalaire Java ; les types scalaires Java n'acceptent pas les valeurs NULL. Cependant, les types d'objet Java acceptent les valeurs NULL.

Pour la classe Java-SQL suivante :

```
public class General implements java.io.Serializable {
       public static int identity1(int I) {return I;}
       public static java.lang.Integer identity2
        (java.lang.Integer I) {return I;}
       public static Address identity3 (Address A) {return A;}
 }
```

Considérons ces appels :

```
declare @I int
declare @A Address;

select @I = General.identity1(@I)
select @I = General.identity2(new java.lang.Integer(@I))
select @A = General.identity3(@A)
```

Les valeurs des variables *@I* et *@A* sont NULL, car aucune valeur ne leur a été affectée.

- L'appel de la méthode identity1( ) a généré une exception. Le type de données du paramètre *@I* de identity1( ) est le type Java int qui est scalaire et n'a pas d'état NULL. Une tentative de transmission d'un argument NULL à identity1( ) a généré une exception.
- L'appel de la méthode identity2( ) réussit. Le type de données du paramètre de identity2( ) est la classe Java java.lang.Integer, et l'expression new crée une instance de java.lang.Integer définie à la valeur de la variable *@I*.
- L'appel de la méthode identity3( ) réussit.

Un appel réussi de identity1( ) ne renvoie jamais de résultat NULL, car le type de renvoi n'a pas d'état NULL. Une valeur NULL ne peut pas être transmise, car la résolution de méthode échoue sans les informations relatives au type de paramètre.

Les appels réussis de identity2( ) et de identity3( ) peuvent renvoyer des résultats NULL.

Valeurs NULL lors de l'utilisation de la fonction SQL *convert*

La fonction convert permet de convertir un objet Java d'une classe en un objet Java d'une superclasse ou d'une sous-classe de cette classe.

Comme indiqué à la section "Sous-types Java-SQL", page 40, la colonne home_addr de la table emps peut contenir des valeurs des classes Address et Address2Line. Dans cet exemple :

```
select name, home_addr>>street, convert(Address2Line, home_addr)>>line2,
       home_addr>>zip from emps
```

l'expression "convert(Address2Line, home_addr)" contient un type de données (Address2Line) et une expression (home_addr). Lors de la compilation, l'expression (home_addr) doit être un sous-type ou un supertype de la classe (Address2Line). Lors de l'exécution, l'action de cet appel convert varie selon que le type d'exécution de la valeur de l'expression est une classe, une sous-classe ou une superclasse :

- Si le type de la valeur de la version d'exécution de l'expression (home_addr) est la classe spécifiée (Address2Line) ou l'une de ses sous-classes, la valeur de l'expression est renvoyée, avec le type de données spécifié (Address2Line).
- Si le type de la valeur de la version d'exécution de l'expression (home_addr) est une superclasse de la classe spécifiée (Address), une valeur NULL est renvoyée.

Adaptive Server évalue l'instruction select pour chaque ligne du résultat. Pour chaque ligne :

- Si la valeur de la colonne home_addr est de type Address2Line, convert renvoie cette valeur, et la référence de champ extrait le champ line2. Si convert renvoie une valeur NULL, la référence de champ est NULL.
- Lorsque convert renvoie une valeur NULL, la référence de champ renvoie cette même valeur.

Aussi, le résultat de l'instruction select présente la valeur line2 pour les lignes dont la colonne home_addr est de type Address2Line et une valeur NULL pour les lignes dont la colonne home_addr est de type Address. Comme indiqué à la section "Traitement des valeurs NULL dans les données Java-SQL", page 43, l'instruction select présente également une valeur NULL line2 pour les lignes dont la colonne home_addr est NULL.

Données de type String Java-SQL

Dans les colonnes Java-SQL, les champs de type String sont stockés en Unicode.

Lorsqu'un champ Java-SQL String est affecté à un élément de données SQL dont le type est char, varchar, nchar, nvarchar ou text, les données Unicode sont converties dans le jeu de caractères du système SQL. Les erreurs de conversion sont spécifiées par les options set char_convert.

Lorsqu'un élément de données SQL dont le type est char, varchar, nchar ou text est affecté à un champ Java-SQL String stocké en Unicode, les données de type caractères sont converties en Unicode. Les codes caractère non définis des données de ce type génèrent des erreurs de conversion.

Chaîne longueur nulle

Dans Transact-SQL, une chaîne de caractères de longueur nulle est traitée comme une valeur NULL, et la chaîne vide ( ) est traitée comme un espace unique.

Par souci de cohérence avec Transact-SQL, lorsqu'une valeur Java-SQL String de longueur nulle est affectée à un élément de données SQL de type char, varchar, nchar, nvarchar ou text, la valeur Java-SQL String est remplacée par un espace unique.

Par exemple :

```
1> declare @s varchar(20)
2> select @s = new java.lang.String()
3> select @s, char_length(@s)
4> go
```

(1 row affected)

```
---------------- ----------------
                               1
```

Dans tous les autres cas, la valeur de longueur nulle est traitée par le système SQL comme une valeur SQL NULL, et lorsqu'elle est affectée à un élément de données Java-SQL de type String, le type de données String est NULL.

Méthodes typées et void

Les méthodes Java (d'instance et statiques) sont soit des méthodes typées, soit des méthodes void. En général, les méthodes typées renvoient une valeur ayant un résultat, alors que les méthodes void exécutent certaines actions et ne renvoient rien.

Par exemple, dans la classe Address :

- la méthode toString( ) est une *méthode typée* dont le type est String ;
- la méthode removeLeadingBlanks( ) est une *méthode void* ;
- le constructeur Address est une *méthode typée* dont le type est la classe Address.

Les méthodes typées sont appelées comme des fonctions, et le mot-clé new est utilisé en cas d'appel d'un constructeur :

```
insert into emps
       values ('Don Green', new Address('234 Stone Road', '99777'),
          new Address2Line( ) )

select name, home_addr>>toString( ) from emps
       where home_addr>>toString( ) like '%Baker%'
```

La méthode removeLeadingBlanks( ) de la classe Address est une méthode d'instance void qui modifie les champs street et zip d'une instance donnée. Vous pouvez appeler removeLeadingBlanks( ) pour la colonne home_addr de chaque ligne de la table emps. Par exemple :

```
update emps
    set home_addr =
       home_addr>>removeLeadingBlanks( )
```

removeLeadingBlanks( ) supprime les blancs de début de chaîne des champs street et zip de la colonne home_addr. L'instruction Transact-SQL update ne fournit ni modèle ni syntaxe pour une action de ce type. Elle remplace simplement les valeurs de colonne.

Méthodes d'instance Java void

Pour utiliser les actions "update-in-place" des méthodes d'instance Java void en SQL, Java dans Adaptive Server traite un appel d'une méthode d'instance Java void de la manière suivante :

Pour une méthode d'instance void M( ) d'une instance Cl d'une classe C, écrite "Cl.M(...)" :

- En SQL, l'appel est traité comme un appel de méthode typée. Le type de résultat est implicitement la classe C, et la valeur résultante est une référence à Cl. Cette référence identifie une copie de l'instance Cl après les actions de l'appel de la méthode d'instance void.
- Dans le système Java, cet appel est un appel de méthode void qui exécute ses actions et ne renvoie aucune valeur.

Par exemple, vous pouvez appeler la méthode the removeLeadingBlanks( ) pour la colonne home_addr d'un ensemble de lignes de la table emps en procédant de la manière suivante :

```
update emps
       set home_addr = home_addr>>removeLeadingBlanks( )
       where home_addr>>removeLeadingBlanks( )>>street like "123%"
```

1. Dans la clause where, "home_addr>>removeLeadingBlanks( )" appelle la méthode removeLeadingBlanks( ) pour la colonne home_addr d'une ligne de la table emps. removeLeadingBlanks( ) supprime les blancs de début de chaîne des champs street et zip d'une copie de la colonne. Le système SQL renvoie ensuite une référence à la copie modifiée de la colonne home_addr. La référence de champ suivante :

   ```
   home_addr>>removeLeadingBlanks( )>>street
   ```

   renvoie le champ street dont les blancs de début de chaîne ont été supprimés. Les références à home_addr dans la clause where s'appliquent à une copie de la colonne. Cette évaluation de la clause where ne modifie *pas* la colonne home_addr.

2. L'instruction update exécute la clause set pour chaque ligne de emps dans laquelle la clause where est vraie.

3. A droite de la clause set, l'appel de "home_addr>>removeLeadingBlanks( )" s'effectue de la même manière que pour la clause where : removeLeadingBlank( ) supprime les blancs de début de chaîne des champs street et zip de cette copie. Le système SQL renvoie ensuite une référence à la copie modifiée de la colonne home_addr.

4 L'instance Address identifiée par le résultat à droite de la clause set est sérialisée et copiée dans la colonne spécifiée à gauche de la clause set : le résultat de l'expression à droite de la clause set est une copie de la colonne home_addr dans laquelle les blancs de début de chaîne ont été supprimés des champs street et zip. La copie modifiée est ensuite réaffectée à la colonne home_addr comme la nouvelle valeur de cette colonne.

Les expressions à droite et à gauche de la clause set sont indépendantes, comme cela est normalement le cas avec l'instruction update.

L'instruction update suivante présente un appel de méthode d'instance void de la colonne mailing_addr, à droite de la clause set en cours d'affectation à la colonne home_address, à gauche de cette clause.

```
update emps
       set home_addr = mailing_addr>>removeLeadingBlanks( )
       where ...
```

Dans cette clause set, la méthode void removeLeadingBlanks( ) de la colonne mailing_addr renvoie une référence à une copie modifiée de l'instance Address2Line dans la colonne mailing_addr. L'instance identifiée par cette référence est ensuite sérialisée et affectée à la colonne home_addr. Cette action met à jour la colonne home_addr ; elle est sans effet sur la colonne mailing_addr.

Méthodes statiques Java Void

Vous ne pouvez pas appeler une méthode statique void à l'aide d'une simple commande SQL execute. Au lieu de cela, vous devez placer l'appel de la méthode statique void dans une instruction select.

Par exemple, supposons qu'une classe Java C ait une méthode statique void M(...) et que M( ) exécute une action que vous souhaitez appeler en SQL. Par exemple, M( ) peut utiliser des appels JDBC pour exécuter une série d'instructions SQL sans valeurs de renvoi, telles que create ou drop qui conviendraient pour une méthode void.

Vous devez appeler la méthode statique void dans une commande a select, telle que :

```
select C.M(...)
```

Pour que les méthodes statiques void puissent être appelées à l'aide d'une commande select, elles sont traitées en SQL comme si elles renvoyaient une valeur dont le type de données est int avec une valeur NULL.

Opérations d'égalité et d'ordre

Vous pouvez utiliser certains opérateurs d'égalité et d'ordre lorsque vous utilisez Java dans la base de données. Cependant, vous ne pouvez pas :

- référencer des éléments de données Java-SQL dans des opérations de tri ;
- référencer des éléments de données Java-SQL dans des opérations d'égalité s'ils sont stockés dans une colonne hors de la ligne ;
- utiliser la clause order by qui nécessite que vous déterminiez l'ordre de tri ;
- effectuer des comparaisons directes à l'aide des opérateurs ">", "<", "<=" ou ">=".

Ces opérations d'égalité sont autorisées dans les colonnes hors de la ligne :

- L'utilisation du mot-clé distinct, défini en termes d'égalité de lignes, notamment avec les colonnes Java-SQL.
- Les comparaisons directes à l'aide des opérateurs "=" et "!=".
- L'utilisation de l'opérateur union (et pas union all) qui élimine les doublons et requiert les mêmes types de comparaisons que la clause distinct.
- L'utilisation de la clause group by qui partitionne les lignes en groupes avec des valeurs égales de la colonne de regroupement.

Ordre d'évaluation et appels de méthode Java

Adaptive Server ne suit pas un ordre défini pour évaluer les opérandes des comparaisons et d'autres opérations. En fait, Adaptive Server évalue chaque requête et choisit l'ordre d'évaluation offrant la meilleure vitesse d'exécution.

Cette section montre comment les différents ordres d'évaluation influent sur le résultat lorsque vous transmettez des colonnes ou des variables et des paramètres comme arguments. Les exemples fournis dans cette section utilisent la classe Java-SQL suivante :

```
public class Utility implements java.io.Serializable {
          public static int F (Address A) {
              if (A.zip.length( ) > 5) return 0;
              else {A.zip = A.zip + "-1234"; return 1;}
          }
          public static int G (Address A) {
              if (A.zip.length( ) > 5) return 0;
              else {A.zip = A.zip + "-1234"; return 1;}
          }
}
```

Colonnes

En règle générale, évitez d'appeler plusieurs méthodes dans la même instruction SQL sur le même objet Java-SQL. Si au moins l'une d'entre elles modifie l'objet, l'ordre d'évaluation peut influer sur le résultat.

Dans l'exemple suivant :

```
select * from emp E
       where Utility.F(E.home_addr) > Utility.F(E.home_addr)
```

la clause where transmet la même colonne home_addr dans deux appels de méthode différents. Considérons l'évaluation de la clause where pour une ligne dont la colonne home_addr contient un zip de cinq caractères, tel que "95123".

Adaptive Server peut initialement évaluer le côté gauche ou droit de la comparaison. Au terme de la première évaluation, il effectue la deuxième. L'exécution étant plus rapide en procédant de cette manière, Adaptive Server montre au deuxième appel les modifications réalisées par le premier appel sur l'argument.

Dans cet exemple, le premier appel choisi par Adaptive Server renvoie 1 et le second renvoie 0. Si l'opérande de gauche est évalué en premier, la comparaison est 1>0 et la clause where est vraie ; si l'opérande de droite est évalué en premier, la comparaison est 0>1 et la clause where est fausse.

Variables et paramètres

De la même manière, l'ordre d'évaluation peut influer sur le résultat lorsque des variables et des paramètres sont transmis comme arguments.

Considérons les instructions suivantes :

```
declare @A Address
declare @Order varchar(20)

select @A = new Address('95444', '123 Port Avenue')
select case when Utility.F(@A)>Utility.G(@A)
          then 'Left'else 'Right'end
select @Order = case when utility.F(@A) > utility.G(@A)
          then 'Left'else 'Right'end
```

La nouvelle instance Address possède un champ zip (code postal) de cinq caractères. Lorsque l'expression case est évaluée, selon que l'opérande de gauche ou de droite de la comparaison est évalué en premier, la comparaison est soit 1>0, soit 0>1, et la variable *@Order* est définie à 'Left'ou 'Right', selon le cas.

S'agissant des arguments de colonne, la valeur de l'expression dépend de l'ordre d'évaluation. Selon que l'opérande de gauche ou de droite de la comparaison est évalué en premier, la valeur résultante du champ zip de l'instance Address référencée par *@A* est soit "95444-4321", soit "95444-1234".

Variables statiques dans les classes Java-SQL

Une variable Java déclarée statique est associée à la classe Java, plutôt qu'à chaque instance de la classe. La variable est allouée une fois pour la classe entière.

Par exemple, vous pouvez inclure une variable statique dans la classe Address qui spécifie la longueur limite recommandée du champ Street :

```
public class Address implements java.io.Serializable {

    public static int recommendedLimit;
    public String street;
    public String zip;
// ...
}
```

Vous pouvez indiquer qu'une variable statique est final, ce qui signifie qu'elle ne peut pas être mise à jour :

```
public static final int recommendedLimit;
```

Sinon, vous pouvez mettre à jour la variable.

Une variable statique d'une classe Java est référencée en SQL en qualifiant la variable avec une instance de la même classe. Par exemple :

```
declare @A Address
select @a>>recommendedLimit
```

En l'absence d'une instance de la classe, procédez de la manière suivante :

```
select (convert(null, Address))>>recommendedLimit
```

L'expression "(convert(null, Address))" convertit une valeur NULL dans un type Address ; en d'autres termes, elle génère une instance Address nulle, que vous pouvez alors qualifier en indiquant le nom de la variable statique. Vous ne pouvez pas référencer une variable statique d'une classe Java en SQL par qualification de cette variable en indiquant le nom de la classe. Ainsi, les deux exemples suivants sont incorrects :

```
select Address.recommendedLimit
```

```
select Address>>recommendedLimit
```

Les valeurs affectées aux variables statiques non finales ne sont accessibles que dans le cadre de la session courante.

Classes Java dans plusieurs bases de données

Vous pouvez stocker des classes Java homonymes dans différentes bases de données sur le même système Adaptive Server. Cette section explique comment utiliser ces classes.

Portée

Lorsque vous installez une classe Java ou un ensemble de classes, elles sont installées dans la base de données courante. Lorsque vous sauvegardez ou chargez une base de données, les classes Java-SQL installées dans cette base de données sont toujours incluses, même si des classes homonymes existent dans d'autres bases de données sur le système Adaptive Server.

Vous pouvez installer des classes Java homonymes dans différentes bases de données. Il peut s'agir :

- de classes identiques installées dans des bases de données différentes ;
- de classes différentes mutuellement compatibles. Ainsi, une valeur sérialisée générée par l'une des classes peut être acceptée par l'autre ;
- de classes différentes offrant une compatibilité "ascendante". En d'autres termes, une valeur sérialisée générée par l'une des classes peut être acceptée par l'autre, mais la réciproque n'est pas vraie ;
- de classes différentes mutuellement incompatibles ; par exemple, une classe nommée Sheet conçue pour un stock de papier et d'autres classes nommées Sheet conçues pour un stock de drap.

Références entre les bases de données

Vous pouvez référencer des objets stockés dans les colonnes d'une table d'une base de données à l'autre.

Par exemple, supposons la configuration suivante :

- La classe Address est installée dans db1 et db2.
- La table emps a été créée dans db1 avec comme propriétaire Smith et dans db2 avec comme propriétaire Jones.

Dans ces exemples, la base de données courante est db1. Vous pouvez appeler une jointure ou une méthode entre les bases de données. Par exemple :

- Une jointure entre les tables des deux bases de données ressemblera à ceci :

```
declare @count int
select @count(*)
   from db2.Jones.emps, db1.Smith.emps
   where db2.Jones.emps.home_addr>>zip =
      db1.Smith.emps.home_addr>>zip
```

- Un appel de méthode entre les bases de données ressemblera à ceci :

```
select db2.Jones.emps.home_addr>>toString( )
       from db2.Jones.emps
       where db2.Jones.emps.name = 'John Stone'
```

Dans ces exemples, les valeurs d'instance ne sont pas transférées. Les champs et les méthodes d'une instance contenue dans db2 sont simplement référencées par une routine dans db1. Ainsi, pour des jointures et des appels de méthodes entre les bases de données :

- db1 n'a pas besoin de contenir une classe Address.
- Si db1 contient une classe Address, ses propriétés peuvent différer de celles de la classe Address dans db2.

Transferts interclasse

Vous pouvez affecter une instance d'une classe d'une base de données à une instance d'une classe homonyme d'une autre base de données. Les instances créées par la classe de la base de données source sont transférées dans des colonnes ou des variables dont le type déclaré est la classe de la base courante (cible).

Vous pouvez effectuer une insertion ou une mise à jour à partir d'une table d'une base de données vers une table d'une autre base. Par exemple :

```
insert into db1.Smith.emps select * from
       db2.Jones.emps

update db1.Smith.emps
       set home_addr = (select db2.Jones.emps.home_addr
           from db2.Jones.emps
       where db2.Jones.emps.name =
           db1.Smith.emps.name)
```

Vous pouvez effectuer une insertion ou une mise à jour à partir d'une variable d'une base de données vers une autre base de données. (Le fragment suivant se trouve dans une procédure stockée sur db2.) Par exemple :

```
declare @home_addr Address
select @home_addr = new Address('94608', '222 Baker
       Street')
insert into db1.Janes.emps(name, home_addr)
       values ('Jone Stone', @home_addr)
```

Dans ces exemples, les valeurs d'instance sont transférées entre les bases de données. Vous pouvez :

- transférer des instances entre deux bases de données locales ;
- transférer des instances entre une base de données locale et une base de données distante ;

- transférer des instances entre un client SQL et un système Adaptive Server ;
- remplacer des classes à l'aide des instructions install et update ou des instructions remove et update.

Dans un transfert interclasse, la sérialisation Java est transférée de la source vers la cible.

Transmission d'arguments interclasse

Vous pouvez transmettre des arguments entre des classes homonymes dans différentes bases de données. Lorsque vous transmettez des arguments interclasse :

- une colonne Java-SQL est associée à la version de la classe Java spécifiée dans la base de données qui contient la colonne ;
- une variable Java-SQL (dans Transact-SQL) est associée à la version de la classe Java spécifiée dans la base de données courante ;
- un résultat intermédiaire Java-SQL de classe C est associé à la version de classe C dans la même base de données que la méthode Java qui a renvoyé le résultat ;
- lorsqu'une valeur d'instance Java *JI* est affectée à une variable ou une colonne cible ou transmise à une méthode Java, *JI* est convertie de sa classe associée dans la classe associée à la méthode ou à la cible réceptrice.

Bases de données temporaires et de travail

Toutes les règles valables pour les classes et les bases de données Java s'appliquent également aux bases de données temporaires et à la base de données modèle :

- Les colonnes Java-SQL des tables temporaires contiennent des sérialisations en chaîne d'octets des instances Java.
- Une colonne Java-SQL est associée à la version de la classe spécifiée dans la base de données temporaire.

Vous pouvez installer des classes Java dans une base de données temporaire, mais elles ne resteront disponibles que tant que la base de données temporaire sera disponible.

La manière la plus simple pour fournir des classes Java de référence dans des bases de données temporaires consiste à installer des classes Java dans la base de données modèle. Elles seront alors présentes dans n'importe quelle base de données temporaire dérivée du modèle.

Classe Java

Cette section présente les classes Java simples utilisées dans ce chapitre pour illustrer Java dans Adaptive Server. Vous trouverez également ces classes dans leur code source Java sous *$SYBASE/$SYBASE_ASE/sample/JavaSql* (UNIX) ou *%SYBASE%\Ase-12_5\sample\JavaSql* (Windows NT).

Voici la classe Address :

```
//
// Copyright (c) 1999
// Sybase, Inc
// Emeryville, CA 94608
// All Rights Reserved
//
/**
* A simple class for address data, to illustrate using a Java class
* as a SQL datatype.
*/
public class Address implements java.io.Serializable {

/**
* The street data for the address.
* @serial A simple String value.
*/
       public String street;

/**
* The zipcode data for the address.
* @serial A simple String value.
*/
       String zip;

/** A default constructor.
*/
       public Address ( ) {
          street = "Unknown";
```

```
            zip = "None";
        }
/**
* A constructor with parameters
* @param S       a string with the street information
* @param Z       a string with the zipcode information
*/
        public Address (String S, String Z) {
            street = S;
            zip = Z;
        }
/**
* A method to return a display of the address data.
* @returns a string with a display version of the address data.
*/
        public String toString( ) {
            return "Street= " + street +   "   ZIP= " + zip;
        }
/**
* A void method to remove leading blanks.
* This method uses the static method
* <code>Misc.stripLeadingBlanks</code>.
*/
        public void removeLeadingBlanks( ) {
        street = Misc.stripLeadingBlanks(street);
        zip = Misc.stripLeadingBlanks(street);
        }
}
```

Voici la classe Address2Line qui est une sous-classe de la classe Address :

```
//
// Copyright (c) 1999
// Sybase, Inc
// Emeryville, CA 94608
// All Rights Reserved
//
/**
* A subclass of the Address class that adds a seond line of address data,
* <p>This is a simple subclass to illustrate using a Java subclass
* as a SQL datatype.
*/
public class Address2Line extends Address implements java.io.Serializable {

/**
* The second line of street data for the address.
* @serial a simple String value
```

```
*/
       String line2;
/**
* A default constructor
*/
       public Address2Line ( ) {
          street = "Unknown";
          line2 = " ";
          zip = "None";
       }
/**
* A constructor with parameters.
* @param S a string with the street information
* @param L2 a string with the second line of address data
* @param Z a string with the zipcode information
*/
public Address2Line (String S, String L2, String Z) {
       street = S;
       line2 = L2;
       zip = Z;
}

/**
* A method to return a display of the address data
* @returns a string with a display version of the address data
*/

public String toString( ) {
       return "Street= " + street + " Line2= " + line2 +   " ZIP= " + zip;
}

/**
* A void method to remove leading blanks.
* This method uses the static method
* <code>Misc.stripLeadingBlanks</code>.
*/

       public void removeLeadingBlanks( ) {
          line2 = Misc.stripLeadingBlanks(line2);
          super.removeLeadingBlanks( );
       }
}
```

La classe Misc contient des ensembles de routines :

```
//
// Copyright (c) 1999
// Sybase, Inc
// Emeryville, CA 94608
```

```
// All Rights Reserved
//
/**
* A non-instantiable class with miscellaneous static methods
* that illustrate the use of Java methods in SQL.
*/

public class Misc{

/**
* The Misc class contains only static methods and cannot be instantiated.
*/

private Misc( ) { }

/**
* Removes leading blanks from a String
*/
       public static String stripLeadingBlanks(String s) {
          if (s == null) return null;
          for (int scan=0; scan<s.length( ); scan++)
             if (!java.lang.Character.isWhitespace(s.charAt(scan) ))
                break;
             } else if (scan == s.length( )){
                return "";
             } else return s.substring(scan);
             }
          }
       }
       return "";
}
/**
* Extracts the street number from an address line.
* e.g., Misc.getNumber("   123 Main Street") == 123
*       Misc.getNumber("   Main Street") == 0
*       Misc.getNumber("") == 0
*       Misc.getNumber("   123   ") == 123
*       Misc.getNumber("   Main 123 ") == 0
* @param s a string assumed to have address data
* @return a string with the extracted street number
*/

       public static int getNumber (String s) {
          String stripped = stripLeadingBlanks(s);
          if (s==null) return -1;
          for(int right=0; right < stripped.length( ); right++){
```

```
            if (!java.lang.Character.isDigit(stripped.charAt(right))) {
               break;
            } else if (right==0){
               return 0;
            } else {
               return java.lang.Integer.parseInt
                  (stripped.substring(0, right), 10);
            }
            }
            return -1;
         }

/**
* Extract the "street" from an address line.
* e.g., Misc.getStreet("   123 Main Street") == "Main Street"
*      Misc.getStreet("   Main Street") == "Main Street"
*      Misc.getStreet("") == ""
*      Misc.getStreet("   123   ") == ""
*      Misc.getStreet("   Main   123 ") == "Main 123"
* @param s a string assumed to have address data
* @return a string with the extracted street name
*/
       public static String getStreet(String s) {
          int left;
          if (s==null) return null;
          for (left=0; left<s.length( ); left++){
             if(java.lang.Character.isLetter(s.charAt(left))) {
                break;
             } else if (left == s.length( )) {
                return "";
             } else {
                return s.substring(left);
             }
          }
          return "";
       }
}
```

CHAPITRE 4

Accès aux données à l'aide de JDBC

Ce chapitre explique comment utiliser JDBC (Java Database Connectivity) pour accéder aux données.

Rubriques	Page
Présentation	63
Concepts JDBC et terminologie	64
Différences entre JDBC client et JDBC serveur	65
Autorisations	66
Utilisation de JDBC pour accéder aux données	66
Erreur de gestion du pilote JDBC natif	74
Classe JDBCExamples	76

Présentation

JDBC offre une interface SQL pour les applications Java. Pour accéder aux données relationnelles à partir de Java, utilisez des appels JDBC.

Vous pouvez utiliser JDBC avec l'interface SQL d'Adaptive Server en procédant de l'une des deux manières suivantes :

- *JDBC sur le client* : les applications clientes Java peuvent effectuer des appels JDBC vers Adaptive Server à l'aide du pilote JDBC de Sybase jConnect.
- *JDBC sur le serveur* : les classes JDBC Java installées dans la base de données peuvent effectuer des appels JDBC vers la base de données à l'aide du pilote JDBC natif d'Adaptive Server.

Les procédures d'appels JDBC pour exécuter des opérations SQL sont globalement similaires dans les deux contextes.

Ce chapitre fournit des exemples de classes et de méthodes JDBC qui expliquent comment exécuter des opérations SQL à l'aide de JDBC. Ces classes et ces méthodes JDBC ne sont pas conçues pour servir de modèles, mais plutôt de lignes directrices.

Concepts JDBC et terminologie

JDBC est une API Java et fait partie des bibliothèques de classes JDBC Java qui contrôlent des fonctions essentielles pour le développement d'applications Java. Les fonctionnalités SQL fournies par JDBC sont similaires à celles d'ODBC et du SQL dynamique.

La séquence d'événements suivante est caractéristique des applications JDBC :

1 Création d'un objet *Connection* : appelez la méthode statiquegetConnection( ) de la classe DriverManager pour créer un objet *Connection*. Ceci a pour effet d'établir une connexion avec une base de données.

2 Génération d'un objet *Statement* : utilisez l'objet *Connection* pour générer un objet *Statement*.

3 Transmission d'une instruction SQL à l'objet *Statement* : si l'instruction est une requête, cette action renvoie un objet *ResultSet*.

   L'objet *ResultSet* contient les données renvoyées par l'instruction SQL, mais les fournit ligne par ligne (de la même manière qu'un curseur).

4 Bouclage sur les lignes du jeu de résultats : appelez la méthode next( ) de l'objet *ResultSet* pour :

   - passer de la ligne courante (la ligne du jeu de résultats actuellement affichée via l'objet *ResultSet*) à la ligne suivante ;
   - renvoyer une valeur booléenne (vrai/faux) pour indiquer s'il existe une ligne à laquelle passer.

5 Pour chaque ligne, extraction des valeurs des colonnes de l'objet *ResultSet* : utilisez getInt( ), getString( ) ou une méthode JDBC similaire pour identifier le nom ou la position de la colonne.

Différences entre JDBC client et JDBC serveur

La différence entre JDBC sur le client et JDBC dans le serveur de base de données est le mode d'établissement d'une connexion avec l'environnement de la base de données.

Lorsque vous utilisez JDBC sur le client ou JDBC dans le serveur, vous faites appel à la méthode Drivermanager.getConnection() pour établir une connexion au serveur.

- Pour JDBC sur le client, vous utilisez le pilote JDBC de Sybase jConnect, et faites appel à la méthode Drivermanager.getConnection() en utilisant l'identification du serveur. Vous établissez ainsi une connexion au serveur désigné.
- Pour JDBC sur le client, vous utilisez le pilote JDBC natif d'Adaptive Server, et faites appel à la méthode Drivermanager.getConnection() en utilisant l'une des valeurs suivantes :
  - jdbc:default:connection
  - jdbc:sybase:ase
  - jdbc:default
  - chaîne vide

  Vous établissez ainsi une connexion au serveur courant. Seul le premier appel à la méthode getConnection() permet de créer la nouvelle connexion au serveur courant. Les appels suivants renvoient une encapsulation de cette connexion sans modifier les propriétés de connexion.

Vous pouvez écrire des classes JDBC afin qu'elles soient exécutées sur le client et sur le serveur en employant une instruction conditionnelle pour construire l'URL.

Autorisations

- *Autorisations d'exécution Java* : à l'instar de toutes les classes JDBC Java de la base de données, n'importe quel utilisateur peut accéder aux classes JDBC contenant des instructions JDBC. Aucun équivalent de l'instruction grant execute n'accorde d'autorisation d'exécution des procédures dans les méthodes Java et il n'est pas nécessaire de qualifier le nom d'une classe JDBC avec le nom de son propriétaire.
- *Autorisations d'exécution SQL* : les classes Java sont exécutées avec les autorisations de la connexion qui les exécute. Ce comportement diffère de celui des procédures stockées, qui sont exécutées avec les autorisations du propriétaire de la base de données.

Utilisation de JDBC pour accéder aux données

Cette section explique comment utiliser JDBC pour exécuter les opérations courantes d'une application SQL. Les exemples sont extraits de la classe JDBCExamples, décrite à la section "Classe JDBCExamples", page 76 et sous *$SYBASE/$SYBASE_ASE/sample/JavaSql* (UNIX) ou *%SYBASE%\Ase-12_5\sample\JavaSql* (Windows NT).

JDBCExamples illustre les principes de fonctionnement d'une interface utilisateur et présente les techniques de codage interne des opérations SQL.

Présentation de la classe JDBCExamples

La classe JDBCExamples utilise la classe Address décrite à la section "Exemples de classes Java", page 12. Pour exécuter ces exemples sur votre machine, installez la classe Address sur le serveur et incorporez-la dans la variable d'environnement CLASSPATH Java du client jConnect.

Vous pouvez appeler les méthodes JDBC de JDBCExamples à partir d'un client jConnect ou d'Adaptive Server.

Remarque Vous devez créer ou supprimer les procédures stockées à partir du client jConnect. Le pilote natif d'Adaptive Server ne supporte pas les instructions create procedure et drop procedure.

Les méthodes statiques JDBCExamples exécutent les opérations SQL suivantes :

- Création et suppression d'une table exemple, xmp :

```
create table xmp (id int, name varchar(50), home Address)
```

- Création et suppression d'une procédure stockée exemple, inoutproc :

```
create procedure inoutproc @id int, @newname varchar(50),
       @newhome Address, @oldname varchar(50) output, @oldhome
       Address output as

select @oldname = name, @oldhome = home from xmp
       where id=@id
update xmp set name=@newname, home = @newhome
       where id=@id
```

- Insertion d'une ligne dans la table xmp.
- Sélection d'une ligne de la table xmp.
- Mise à jour d'une ligne dans la table xmp.
- Appel de la procédure stockée inoutproc, qui possède des paramètres d'entrée et de sortie appartenant aux types de données java.lang.String et Address.

JDBCExamples n'agit que sur la table xmp et sur la procédure inoutproc.

Méthodes JDBC *main( )* et *serverMain( )*

JDBCExamples comprend deux méthodes JDBC principales :

- main( ) est appelée à partir de la ligne de commande du client jConnect.
- serverMain( ) exécute les mêmes actions que main( ), mais est appelée au sein d'Adaptive Server.

Toutes les actions de la classe JDBCExamples sont appelées par l'une de ces méthodes JDBC grâce à un paramètre qui indique l'action à exécuter.

Utilisation de *main( )*

Vous pouvez appeler la méthode JDBC main( ) à partir d'une ligne de commande jConnect en procédant comme suit :

```
java JDBCExamples
       "nom-serveur:numéro-port?user=nom-utilisateur&password=mot_de_passe"
       action
```

Vous pouvez déterminer *nom_serveur* et *numéro_port* à partir du fichier d'interfaces en utilisant l'outil dsedit. *nom_utilisateur* et *mot_de_passe* sont votre nom d'utilisateur et votre mot de passe. Si vous omettez &password=*mot_de_passe*, la valeur par défaut est le mot de passe vide. Par exemple :

```
"antibes:4000?user=smith&password=1x2x3"
"antibes:4000?user=sa"
```

Assurez-vous de mettre le paramètre entre guillemets.

Le paramètre *action* peut être create table, create procedure, insert, select, update ou call. La distinction majuscules/minuscules n'est pas applicable.

Vous pouvez appeler JDBCExamples à partir d'une ligne de commande jConnect pour créer la table xmp et la procédure stockée inoutproc en procédant comme suit :

```
java JDBCExamples "antibes:4000?user=sa" CreateTable
java JDBCExamples "antibes:4000?user=sa" CreateProc
```

Vous pouvez appeler JDBCExamples pour les actions insert, select, update et call en procédant comme suit :

```
java JDBCExamples "antibes:4000?user=sa" insert
java JDBCExamples "antibes:4000?user=sa" update
java JDBCExamples "antibes:4000?user=sa" call
java JDBCExamples "antibes:4000?user=sa" select
```

Ces appels affichent le message "Action performed".

Pour supprimer la table xmp et la procédure stockée inoutproc, entrez :

```
java JDBCExamples "antibes:4000?user=sa" droptable
java JDBCExamples "antibes:4000?user=sa" dropproc
```

Utilisation de *serverMain( )*

Remarque Le pilote JDBC serveur ne supportant pas create procedure ni drop procedure, créez la table xmp et la procédure stockée exemple inoutproc avec des appels clients de la méthode JDBC main( ) avant d'exécuter ces exemples. Pour plus d'informations, reportez-vous à la section "Présentation de la classe JDBCExamples", page 66.

Une fois que vous avez créé xmp et inoutproc, vous pouvez appeler la méthode JDBC serverMain( ) en procédant comme suit :

```
select JDBCExamples.serverMain('insert')
go
select JDBCExamples.serverMain('select')
go
select JDBCExamples.serverMain('update')
go
select JDBCExamples.serverMain('call')
go
```

Remarque Les appels serveur de serverMain( ) ne requièrent pas de paramètre *nom-serveur:numéro-port* ; Adaptive Server se connecte simplement à lui-même.

Etablissement d'une connexion JDBC : méthode JDBC *Connecter( )*

main( ) et serverMain( ) appellent la méthode JDBC connecter( ), qui renvoie un objet JDBC *Connection*. L'objet *Connection* est à la base de toutes les opérations SQL suivantes.

main( ) et serverMain( ) appellent connecter( ) avec un paramètre qui indique le pilote JDBC de l'environnement serveur ou client. L'objet *Connection* renvoyé est ensuite transmis comme argument aux autres méthodes JDBC de la classe JDBCExamples. Du fait de l'isolement des actions de connexion dans la méthode JDBCconnecter( ), les autres méthodes de JDBCExamples sont indépendantes de leur environnement serveur ou client.

Acheminement de l'action vers d'autres méthodes : méthode *doAction( )*

La méthode doAction( ) achemine l'appel vers l'une des autres méthodes JDBC, en fonction du paramètre *action*.

La méthode doAction( ), qui contient le paramètre *Connection*, le transmet simplement à la méthode cible. Elle possède également un paramètre *locale*, qui indique s'il s'agit d'un appel serveur ou client. *Connection* génère une exception si create procedure or drop procedure est appelé dans un environnement serveur.

Exécution des opérations SQL impératives : méthode *doSQL( )*

La méthode JDBC doSQL( ) exécute des actions SQL qui ne requièrent pas de paramètre d'entrée ou de sortie, telles que create table, create procedure, drop table et drop procedure.

doSQL( ) a deux paramètres : l'objet *Connection* est l'instruction SQL qu'il doit exécuter. doSQL( ) crée un objet JDBC *Statement* et l'utilise pour exécuter l'instruction SQL spécifiée.

Exécution d'une instruction *update* : méthode *UpdateAction( )*

La méthode updateAction( ) exécute une instruction Transact-SQL update. Cette action update est la suivante :

```
String sql = "update xmp set name = ?, home = ? where id = ?";
```

Elle met à jour les colonnes name et home de toutes les lignes avec une valeur *id* donnée.

Les valeurs update des colonnes name et home, ainsi que la valeur id, sont spécifiées par des marqueurs de paramètre (?). updateAction( ) fournit les valeurs de ces marqueurs après la phase de préparation de l'instruction mais avant son exécution. Les valeurs sont spécifiées par les méthodes setString( ), setObject( ) et setInt( ) avec les paramètres suivants :

- Le marqueur de paramètre ordinal à substituer.
- La valeur à substituer.

Par exemple :

```
pstmt.setString(1, name);
pstmt.setObject(2, home);
pstmt.setInt(3, id);
```

Après avoir effectué ces substitutions, updateAction( ) exécute l'instruction update.

Pour simplifier updateAction( ), les valeurs substituées dans l'exemple sont fixes. Normalement, les applications calculent les valeurs substituées ou les obtiennent sous forme de paramètres.

Exécution d'une instruction *select* : méthode *selectAction( )*

La méthode selectAction( ) exécute une instruction Transact-SQL select :

```
String sql = "select name, home from xmp where id=?";
```

La clause where a un marqueur de paramètre (?) pour la ligne à sélectionner. A l'aide de la méthode JDBC setInt( ), selectAction( ) fournit une valeur pour le marqueur de paramètre après la préparation de l'instruction SQL :

```
PreparedStatement pstmt =
    con.prepareStatement(sql);
pstmt.setInt(1, id);
```

Puis, selectAction( ) exécute l'instruction select :

```
ResultSet rs = pstmt.executeQuery();
```

Remarque Pour les instructions SQL qui ne renvoient pas de résultat, utilisez les méthodes doSQL( ) et updateAction( ). Elles exécutent des instructions SQL avec la méthode executeUpdate( ).

Pour les instructions SQL qui renvoient des résultats, utilisez la méthode executeQuery( ), qui renvoie un objet *ResultSet*.

L'objet *ResultSet* est similaire à un curseur SQL. Initialement, il est positionné avant la première ligne de résultats. Chaque appel de la méthode JDBC next( ) fait passer l'objet *ResultSet* à la ligne suivante, jusqu'à ce qu'il n'y ait plus de ligne.

selectAction( ) exige que l'objet *ResultSet* n'ait qu'une seule ligne. La méthode selecter( ) appelle la méthode suivante et vérifie si *ResultSet* ne contient pas de ligne ou s'il en contient plusieurs.

```
if (rs.next()) {
   name = rs.getString(1);
   home = (Address)rs.getObject(2);
   if (rs.next()) {
      throw new Exception("Error:  Select returned multiple rows");
   } else { // No action
   }
} else { throw new Exception("Error:  Select returned no rows");
}
```

Dans le code ci-dessus, l'appel des méthodes JDBC getString( ) et getObject( ) extrait les deux colonnes de la première ligne du jeu de résultats. L'expression "(Address)rs.getObject(2)" extrait la seconde colonne comme un objet Java, puis attache cet objet à la classe Address. Si l'objet renvoyé n'est pas une classe Address, une exception est générée.

selectAction( ) extrait une ligne et détermine dans quels cas il n'y a pas de ligne et dans quels cas il y en a plusieurs. Une application traitant plusieurs lignes *ResultSet* bouclerait simplement sur les appels de la méthode JDBC next( ) et traiterait chaque ligne comme dans le cas d'une seule ligne.

Exécution en mode batch

Si vous voulez exécuter un batch d'instructions SQL, assurez-vous d'utiliser la méthode execute( ). Si vous utilisez executeQuery( ) comme mode batch :

- Si l'opération batch ne renvoie pas un ensemble de résultats (ne contient pas d'instructions select), le batch s'exécute sans erreur.
- Si l'opération batch renvoie un ensemble de résultats, toutes les instructions suivant l'instruction qui renvoie le résultat sont ignorées. Si getXXX( ) est appelé pour obtenir un paramètre de sortie, les instructions restantes s'exécutent et l'ensemble de résultats courants est fermé.
- Si l'opération batch renvoie plus d'un ensemble de résultats, une exception est générée et l'opération est abandonnée.

L'utilisation de execute( ) assure l'exécution complète du batch pour tous les cas.

Appel d'une procédure stockée SQL : méthode *callAction( )*

La méthode JDBC callAction( ) appelle la procédure stockée inoutproc :

```
create proc inoutproc @id int, @newname varchar(50), @newhome Address,
          @oldname varchar(50) output,  @oldhome Address output as

       select @oldname = name, @oldhome = home from xmp where id=@id
       update xmp set name=@newname, home = @newhome where id=@id
```

Cette procédure possède trois paramètres d'entrée (@*id*, @*newname* et @*newhome*) et deux paramètres de sortie (@*oldname* et @*oldhome*). callAction( ) définit les colonnes name et home de la ligne de la table xmp en attribuant la valeur ID @*id* à @*newname* et @*newhome*, et renvoie les valeurs précédentes de ces colonnes aux paramètres de sortie @*oldname* et @*oldhome*.

La procédure inoutproc montre comment fournir les paramètres d'entrée et de sortie dans un appel JDBC.

callAction( ) exécute l'instruction qui prépare l'instruction d'appel :

```
CallableStatement cs = con.prepareCall("{call inoutproc (?, ?, ?, ?, ?)}");
```

Tous les paramètres de l'appel sont spécifiés sous forme de marqueurs de paramètre (?).

callAction( ) fournit les valeurs des paramètres d'entrée à l'aide des méthodes JDBC setInt( ), setString( ) et setObject( ) utilisées dans les méthodes JDBC doSQL( ), updatAction( ) et selectAction( ) :

```
cs.setInt(1, id);
cs.setString(2, newName);
cs.setObject(3, newHome);
```

Ces méthodes JDBC set ne sont pas applicables aux paramètres de sortie. Avant d'exécuter l'instruction d'appel, callAction( ) spécifie les types de données attendus des paramètres de sortie à l'aide de la méthode JDBC registerOutParameter( ) :

```
cs.registerOutParameter(4, java.sql.Types.VARCHAR);
cs.registerOutParameter(5, java.sql.Types.JAVA_OBJECT);
```

callAction( ) exécute ensuite l'instruction d'appel et obtient les valeurs de sortie en utilisant les mêmes méthodes JDBC getString( ) et getObject( ) que la méthode selectAction( ) utilisée :

```
int res = cs.executeUpdate();
String oldName = cs.getString(4);
Address oldHome = (Address)cs.getObject(5);
```

Erreur de gestion du pilote JDBC natif

Sybase supporte et applique toutes les méthodes des classes java.sql.SQLException et java.sql.SQLWarning. SQLException fournit des informations sur les erreurs d'accès à la base de données. SQLWarning étend la portée de SQLException et fournit des informations sur les avertissements d'accès à la base de données.

Les erreurs générées par Adaptive Server sont numérotées selon leur sévérité. Plus l'erreur est grave, plus son numéro est élevé. Les erreurs sont groupées selon leur sévérité :

- Les avertissements (EX_INFO : sévérité 10) – sont convertis en avertissements SQL (SQLWarnings).
- Les exceptions (sévérité 11 à 18) – sont converties en exceptions SQL (SQLExceptions).
- Les erreurs fatales (sévérité 19 à 24) – sont converties en exceptions SQL fatales (SQLExceptions).

Les exceptions SQLExceptions peuvent être générées par JDBC, Adaptive Server ou le pilote JDBC natif. La génération d'une exception SQLException annule la requête JDBC à la base de l'erreur. Le comportement ultérieur du système sera différent selon le niveau où l'erreur est détectée.

- *Si l'erreur est détectée dans Java* – un bloc d'essai (try), suivi d'un bloc de détection (catch) traitent l'erreur.

  Adaptive Server fournit plusieurs messages d'erreur SQLException propres aux pilotes JDBC étendus. Ces erreurs sont toutes EX_USER (sévérité 16) et sont toujours détectées. Aucun message SQLWarning propre à un pilote n'est disponible.

- *Si l'erreur n'est pas détectée dans Java* – la machine virtuelle Java transmet le contrôle à Adaptive Server, Adaptive Server détecte l'erreur et une erreur SQLException non gérée est générée.

  La commande raiserror s'utilise en général avec les procédures stockées pour générer une erreur et afficher un message destiné à l'utilisateur. Lorsqu'une procédure stockée appelant la commande raiserror est exécutée via JDBC, l'erreur est traitée comme une erreur interne de sévérité EX_USER, et une exception SQLException non fatale est générée.

Remarque Vous ne pouvez pas accéder aux données d'erreur détaillées en utilisant la commande raiserror ; la clause with errordata n'est pas active pour la commande SQLException.

Si une erreur provoque l'abandon d'une transaction, le résultat dépend du contexte de la transaction dans lequel la méthode Java est appelée :

- *Si la transaction contient plusieurs instructions* – la transaction est abandonnée et le contrôle est renvoyé au serveur qui annule la transaction entière. Le pilote JDBC arrête de traiter les requêtes jusqu'à ce que le contrôle soit revenu du serveur.
- *Si la transaction contient une seule requête* – la transaction est abandonnée, son instruction SQL est annulée, et le pilote JDBC continue le traitement des requêtes.

Les cas suivants illustrent les différents résultats : Méthode Java jdbcTests.Errorexample() contenant les instructions suivantes :

```
stmt.executeUpdate("delete from parts where partno = 0”);                Q2
stmt.executeQuery("select 1/0");                                         Q3
stmt.executeUpdate("delete from parts where partno = 10");               Q4
```

Une transaction contenant plusieurs instructions inclut les commandes SQL suivantes :

```
begin transaction
delete from parts where partno = 8                                       Q1
select JDBCTests.Errorexample()
```

Dans ce cas, ces actions sont le résultat d'une transaction abandonnée :

- Une erreur de division par zéro est générée dans Q3.
- Les modifications de Q1 et Q2 sont annulées.
- La transaction entière est abandonnée.

Une transaction contenant une seule instruction inclut les commandes SQL suivantes :

```
set chained off
delete from parts where partno = 8                                       Q1
select JDBCTests.Errorexample()
```

Dans ce cas :

- Une erreur de division par zéro est générée dans Q3.
- Les modifications de Q1 et Q2 ne sont pas annulées.
- L'exception est détectée dans les blocs de détection et d'essai "catch" et "try" dans JDBCTests.Errorexample.
- La suppression spécifiée dans Q4 n'est pas exécutée car elle est traitée dans les mêmes blocs d'essai et de détection "try" et "catch" que Q3.
- Les requêtes JDBC hors des blocs courants "try" et "catch" peuvent s'exécuter.

Classe JDBCExamples

```
// Classe exemple illustrant l'utilisation des fonctionnalités JDBC
// avec la fonction Java dans Adaptive Server.
//
// Les méthodes JDBC de cette classe exécutent une série d'opérations SQL.
// Ces méthodes JDBC peuvent être appelées soit depuis un client Java,
// en utilisant la méthode JDBC principale, soit depuis le serveur SQL,
// en utilisant la méthode JDBC internalMain.
//
import java.sql.*;              // JDBC
public class JDBCExamples {
{
```

Méthode main( )

```
// Méthode JDBC principale, qui doit être appelée depuis une ligne de commande
   du côté client
//
    public static void main(String args[]) {
        if (args.length!=2) {
            System.out.println("\n Usage:    "
                 + "java ExternalConnect server-name:port-number
                 action ");
            System.out.println("  The action is connect, createtable,
              " + "createproc, drop, "
              + "insert, select, update, or call \n" );
            return;
        }
        try{
           String server = args[0];
           String action = args[1].toLowerCase();
           Connection con = connecter(server);
           String workString = doAction( action, con, client);
           System.out.println("\n" + workString + "\n");
        } catch (Exception e) {
           System.out.println("\n Exception: ");
           e.printStackTrace();
        }
    }
```

Méthode internalMain( )

```
// Méthode JDBCExamples équivalente à 'main',
// à appeler depuis SQL ou Java dans le serveur.

    public static String internalMain(String action) {
        try {
           Connection con = connecter("default");
           String workString = doAction(action, con, server);
           return workString;
        }  catch ( Exception e ) {
             if (e.getMessage().equals(null)) {
               return "Exc: " + e.toString();
             } else {
               return "Exc - " + e.getMessage();
             }
        }
    }
```

Méthode connecter( )

```
// Méthode JDBCExamples utilisée pour établir une connexion.
// Elle peut être appelée depuis le serveur avec l'argument 'default',
// ou depuis un client, avec un argument correspondant au nom du serveur.
public static  Connection connecter(String server)
          throws Exception, SQLException, ClassNotFoundException  {

       String forName="";
       String url="";

       if (server=="default") {  // connexion serveur au serveur courant
          forName = "sybase.asejdbc.ASEDriver";
          url = "jdbc:default:connection";
       } else if (server!="default") {  //connexion cliente au serveur
          forName= "com.sybase.jdbc.SybDriver";
          url = "jdbc:sybase:Tds:"+ server;
       }

       String user = "sa";
       String password = "";

       // Chargement du pilote
       Class.forName(forName);
       // Etablissement d'une connexion
       Connection con = DriverManager.getConnection(url,
          user, password);
       return con;
    }
```

Méthode doAction( )

```
// Méthode JDBCExamples utilisée pour effectuer l'acheminement vers l'action
   à exécuter.

    public static String doAction(String action, Connection con,
             String locale)
           throws Exception {

        String createProcScript =
             " create proc inoutproc @id int, @newname varchar(50),
             @newhome Address, "
           + "   @oldname varchar(50) output,  @oldhome Address
             output as "
           + " select @oldname = name, @oldhome = home from xmp
             where id=@id
           + " update xmp set name=@newname, home = @newhome
             where id=@id "
        String createTableScript =
          " create  table xmp (id int, name varchar(50),
             home Address)" ;

        String dropTableScript = "drop   table xmp  ";
        String dropProcScript = "drop  proc inoutproc ";
          String insertScript = "insert  into xmp "
          + "values (1, 'Joe Smith', new Address('987 Shore',
          '12345'))";

        String workString = "Action (" + action + ) ;
        if (action.equals("connect")) {
             workString += "performed";
        } else if (action.equals("createtable")) {
             workString += doSQL(con, createTableScript );
        } else if (action.equals("createproc")) {
             if (locale.equals(server))  {
                 throw new exception (CreateProc cannot be performed
                 in the server);
             } else {
                 workString += doSQL(con, createProcScript );
             }
        } else if (action.equals("droptable")) {
             workString += doSQL(con, dropTableScript );
        } else if (action.equals("dropproc")) {
             if (locale.equals(server))  {
                 throw new exception (CreateProc cannot be performed
                 in the server);
             } else {
                 workString += doSQL(con, dropProcScript );
```

```
            }
        } else if (action.equals("insert")) {
            workString += doSQL(con, insertScript );
        } else if (action.equals("update")) {
            workString += updateAction(con);
        } else if (action.equals("select")) {
            workString += selectAction(con);
        } else if (action.equals("call")) {
            workString += callAction(con);
        } else { return "Invalid action:  " + action ;
        }
        return workString;
    }
```

Méthode doSQL( )

```
// Méthode JDBCExamples utilisée pour exécuter une instruction SQL.

    public static String doSQL (Connection con, String action)
           throws Exception {

        Statement stmt = con.createStatement();
        int res = stmt.executeUpdate(action);
        return "performed";
 }
```

Méthode updateAction( )

```
// Méthode JDBC utilisée pour mettre à jour une certaine ligne de la table 'xmp'.
// Cette méthode illustre l'utilisation des instructions préparées et des
   marqueurs de paramètre.

    public static String updateAction(Connection con)
           throws Exception {

        String sql = "update xmp set name = ?, home = ? where id = ?";
        int id=1;
        Address home = new Address("123 Main", "98765");
        String name = "Sam Brown";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setString(1, name);
        pstmt.setObject(2, home);
        pstmt.setInt(3, id);
        int res = pstmt.executeUpdate();
        return "performed";
    }
```

Méthode selectAction( )

```
// Méthode JDBCExamples utilisée pour extraire une certaine
// ligne de la table 'xmp'.
// Cette méthode illustre l'utilisation des instructions préparées, des
   marqueurs de paramètre
// et des jeux de résultats.

    public static String selectAction(Connection con)
           throws Exception {

        String sql = "select name, home from xmp where id=?";
        int id=1;
        Address home = null;
        String name = "";
        String street = "";
        String zip = "";
        PreparedStatement pstmt = con.prepareStatement(sql);
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
           name = rs.getString(1);
           home = (Address)rs.getObject(2);
           if (rs.next()) {
              throw new Exception("Error:  Select returned
                  multiple rows");
           } else { // No action
           }
        } else { throw new Exception("Error:  Select returned no rows");
        }
        return "- Row with id=1:  name("+ name + )
                + " street(" + home.street + ) zip("+ home.zip + );
```

Méthode callAction( )

```
// Méthode JDBCExamples utilisée pour appeler une procédure stockée,
// en transmettant les paramètres d'entrée et de sortie de type String
 // et Address.
 // Cette méthode illustre l'utilisation des instructions pouvant être appelées,
    des marqueurs de paramètre
 // et des jeux de résultats.

    public static String callAction(Connection con)
           throws Exception {
        CallableStatement cs  = con.prepareCall("{call inoutproc
          (?, ?, ?, ?, ?)}");
```

```
        int id = 1;
        String newName = "Frank Farr";
        Address newHome = new Address("123 Farr Lane", "87654");
        cs.setInt(1, id);
        cs.setString(2, newName);
        cs.setObject(3, newHome);
        cs.registerOutParameter(4, java.sql.Types.VARCHAR);
        cs.registerOutParameter(5, java.sql.Types.JAVA_OBJECT);
        int res = cs.executeUpdate();
        String oldName = cs.getString(4);
        Address oldHome = (Address)cs.getObject(5);
        return "- Old values of row with id=1: name("+oldName+ )
          street(" + oldHome.street + ")  zip("+ oldHome.zip + );
    }
}
```

CHAPITRE 5

Procédures stockées et fonctions SQLJ

Ce chapitre explique comment encapsuler les méthodes Java dans les noms SQL et les utiliser comme procédures et fonctions stockées d'Adaptive Server.

Nom	Page
Présentation	83
Appel des méthodes Java dans Adaptive Server	87
Aspects généraux	85
Utilisation de Sybase Central pour la gestion des fonctions et les procédures SQLJ	89
Fonctions SQLJ définies par l'utilisateur	90
Procédures stockées SQLJ	96
Visualisation des informations sur les fonctions et les procédures SQLJ	107
Fonctions avancées	108
Implémentation SQLJ et Sybase : comparaison	113
Classe SQLJExamples	116

Présentation

Vous pouvez inclure les méthodes statiques Java dans les encapsulations SQL et les utiliser comme des procédures stockées ou des fonctions Transact-SQL intégrées. Cette fonctionnalité :

- permet aux méthodes Java de renvoyer des paramètres de sortie et des jeux de résultats à l'environnement appelant ;
- est compatible avec la partie 1 de la spécification ANSI SQLJ standard ;
- permet de bénéficier de la syntaxe SQL traditionnelle, des métadonnées et des autorisations ;

- permet d'utiliser les méthodes Java existantes en tant que procédures ou fonctions SQLJ sur le serveur, le client et toute base de données externe compatible SQLJ.

❖ **Création d'une procédure stockée ou d'une fonction SQLJ**

Effectuez les opérations suivantes pour créer et exécuter une procédure stockée ou une fonction SQLJ.

1 Créez et compilez la méthode Java. Installez la classe de la méthode dans la base de données avec l'utilitaire installjava.

Reportez-vous au chapitre 2, "Préparation et maintenance de Java dans la base de données", pour plus d'informations sur la création, la compilation et l'installation des méthodes Java dans Adaptive Server.

2 Utilisez l'instruction SQLJ create procedure ou create function pour définir un nom SQL pour la méthode.

3 Exécutez la procédure ou la fonction. Les exemples dans ce chapitre utilisent des appels de méthode JDBC ou isql. Vous pouvez également exécuter la méthode en utilisant Embedded SQL ou ODBC.

Conformité avec les spécifications de SQLJ Partie 1

Les procédures stockées et fonctions SQLJ d'Adaptive Server sont conformes aux spécifications de la norme SQLJ Partie 1 pour l'utilisation de Java avec SQL. Reportez-vous à la section "Normes", page 5 pour obtenir une description des normes SQLJ.

Adaptive Server supporte la plupart des fonctions décrites dans la spécification SQLJ Partie 1, avec, toutefois, trois exceptions. Les fonctionnalités non supportées sont répertoriées dans le tableau 5-3, page 114 ; les fonctionnalités partiellement supportées sont répertoriées dans le tableau 5-4, page 114. Les fonctionnalités définies par Sybase – fonctionnalités non définies par le standard et confiées à l'implémentation – sont répertoriées dans le tableau 5-5, page 115.

Dans les cas où l'implémentation propriétaire de Sybase est différente des spécifications SQLJ, Sybase supporte la norme SQLJ. Par exemple, les procédures stockées Sybase SQL non Java supportent deux modes de paramètres : in et inout. La norme SQLJ supporte trois modes de paramètres : in, out et inout. La syntaxe Sybase pour créer des procédures stockées SQLJ supporte les trois modes de paramètres.

Aspects généraux

Cette section décrit les aspects généraux et les contraintes qui s'appliquent aux fonctions et procédures stockées SQLJ.

- Seules les méthodes (classes) statiques public peuvent être référencées dans une fonction SQLJ ou dans une procédure stockée.
- Le mappage de types de données est vérifié lorsque la routine SQLJ est créée. Pendant l'exécution d'une routine SQLJ, les données sont transmises de SQL à Java, puis de retour à SQL. Toute conversion de données requise pendant l'exécution doit suivre les règles de mappage de types de données définies dans la norme JDBC.

  Reportez-vous à la section "Mappage de type de données Java et SQL", page 108 pour plus d'informations sur le mappage des types de données et les conversions des routines SQLJ.
- Si une méthode référencée par une fonction ou une procédure stockée SQLJ appelle SQL, renvoie des paramètres du système SQL à l'environnement appelant, ou renvoie des jeux de résultats de SQL à l'environnement appelant, vous devez utiliser une interface JDBC d'Adaptive Server, telle que Sybase jConnect ou le driver interne JDBC d'Adaptive Server, permettant un accès orienté objet à la base de données relationnelle.

Sécurité et autorisations

Sybase fournit différents modèles de sécurité pour les procédures SQLJ stockées et les fonctions SQLJ.

Les fonctions SQLJ et les fonctions définies par l'utilisateur (UDF) (reportez-vous à la section "Appel des méthodes Java en SQL", page 33) utilisent un même modèle de sécurité. L'autorisation d'exécution d'une fonction définie par l'utilisateur ou d'une fonction SQLJ est octroyée de manière implicite à public (tous les utilisateurs). Si la fonction exécute les requêtes SQL en utilisant JDBC, l'autorisation d'accès aux données est comparée aux autorisations de l'appelant de la fonction. Ainsi, si l'utilisateur A appelle une fonction ayant accès à la table t1, l'utilisateur A doit posséder l'autorisation select pour t1 ; sinon, la requête échoue.

Les procédures stockées SQLJ utilisent le même modèle de sécurité que les procédures stockées Transact-SQL. L'utilisateur doit posséder une autorisation explicite pour exécuter une procédure stockée SQLJ ou Transact-SQL. Si une procédure SQLJ effectue des requêtes SQL en utilisant JDBC, le support d'autorisation implicite est octroyé. Ce modèle de sécurité permet au propriétaire d'une procédure stockée, s'il est le propriétaire de tous les objets référencés par la procédure, d'octroyer l'autorisation d'exécution sur la procédure à un autre utilisateur. L'utilisateur possédant l'autorisation d'exécution peut exécuter toutes les requêtes SQL dans la procédure stockée, même s'il ne possède pas l'autorisation d'accès à ces objets.

Pour plus d'informations sur la sécurité des procédures stockées, reportez-vous au *Guide d'administration système*.

SQLJExamples

Les exemples de ce chapitre illustrent une table SQL de ventes nommée sales_emps contenant les colonnes suivantes :

- name – le nom de l'employé
- id – le numéro d'identification de l'employé
- state – l'état de résidence de l'employé
- sales – le volume de ventes de l'employé
- jobcode – le code de la fonction de l'employé

La définition de la table est la suivante :

```
create table sales_emps
    (name varchar(50), id char(5),
    state char(20), sales decimal (6,2),
    jobcode integer null)
```

La classe exemple est SQLJExamples et les méthodes sont les suivantes :

- region() – associe un code d'état au numéro d'une région. La méthode n'utilise pas SQL.
- correctStates() – exécute une commande SQL update pour corriger l'orthographe des codes state. L'ancienne orthographe, ainsi que la nouvelle sont spécifiées par des paramètres d'entrée.

- bestTwoEmps() – définit les deux employés avec les meilleures ventes sales et renvoie ces valeurs comme paramètres de sortie.
- SQLJExamplesorderedEmps() – crée un jeu de résultats SQL comprenant des lignes d'employés sélectionnés, triées par la valeur des ventes dans la colonne sales et renvoie le jeu de résultats au client.
- job() – renvoie une valeur de chaîne correspondant à une valeur entière de code de fonction.

Reportez-vous à la section "Classe SQLJExamples", page 116 pour obtenir le texte de chaque méthode.

Appel des méthodes Java dans Adaptive Server

Vous pouvez faire appel aux méthodes Java dans Adaptive Server de deux façons :

- Appeller les méthodes Java directement dans SQL. Les instructions d'appel des méthodes selon ce mode sont présentées au chapitre 3, "Utilisation des classes Java avec le langage SQL".
- Appeller les méthodes Java indirectement en utilisant des procédures stockées et des fonctions SQLJ qui fournissent des alias Transact-SQL pour le nom de la méthode. Ce chapitre décrit l'appel des méthodes Java selon ce mode.

Quel que soit votre choix, vous devez tout d'abord créer vos méthodes Java et les installer dans la base de données Adaptive Server à l'aide de l'utilitaire installjava. Pour plus d'informations, reportez-vous au chapitre 2, "Préparation et maintenance de Java dans la base de données",

Appel des méthodes Java directement par leurs noms Java

Vous pouvez appeler des méthodes Java dans SQL en utilisant leurs noms Java qualifiés. Référencez les instances pour les méthodes d'instance et les instances ou les classes pour les méthodes statiques.

Vous pouvez utiliser les méthodes statiques comme fonctions définies par l'utilisateur (UDF) qui renvoient une valeur à l'environnement appelant. Il est possible d'utiliser une méthode statique Java comme fonction utilisateur dans des procédures stockées, des triggers, des clauses where, des instructions select et partout où les fonctions SQL intégrées sont admises.

Lorsque vous appelez une méthode Java en utilisant son nom, vous ne pouvez pas utiliser des méthodes qui renvoient des paramètres de sortie ou des jeux de résultats à l'environnement appelant. Une méthode peut manipuler les données qu'elle reçoit d'une connexion JDBC, mais la seule valeur qu'elle peut renvoyer à l'environnement appelant est la valeur de retour unique déclarée dans le cadre de sa définition.

Vous ne pouvez pas utiliser des appels de fonctions utilisateur à partir d'une autre base de données que celle dans laquelle est définie la fonction.

Reportez-vous au chapitre 3, "Utilisation des classes Java avec le langage SQL", pour des informations sur l'utilisation des méthodes Java de cette manière.

Appel des méthodes Java indirectement en utilisant SQLJ

Vous pouvez appeler des méthodes Java comme fonctions ou procédures stockées SQLJ. En encapsulant la méthode Java dans un conteneur SQL, vous pouvez exploiter les fonctions suivantes :

- Vous pouvez utiliser des procédures stockées SQLJ pour renvoyer des jeux de résultats et des paramètres de sortie à l'environnement appelant.
- Vous pouvez exploiter les fonctions de métadonnées SQL. Par exemple, vous pouvez visualiser la liste de toutes les procédures stockées ou des fonctions dans la base de données.
- SQLJ fournit un nom SQL pour une méthode, ce qui vous permet de protéger l'appel de la méthode avec les autorisations SQL standard.
- Sybase SQLJ est conforme à la norme reconnue SQLJ Partie 1, ce qui vous permet d'utiliser les procédures et fonctions Sybase SQLJ dans des environnements conformes non-Sybase.
- Vous pouvez appeler des fonctions SQLJ et des procédures stockées SQLJ entre les bases de données.
- Adaptive Server vérifie le mappage des types de données lorsque la routine SQLJ est créée ; ainsi, vous n'avez pas à vous inquiéter de ce mappage lorsque vous exécutez les routines.

Vous devez référencer les méthodes statiques dans une routine SQLJ ; vous ne pouvez pas référencer les méthodes d'instance.

Ce chapitre explique comment utiliser les méthodes Java comme procédures stockées et fonctions SQLJ.

Utilisation de Sybase Central pour la gestion des fonctions et les procédures SQLJ

Vous pouvez gérer les fonctions et les procédures SQLJ depuis la ligne de commande en utilisant isql et à partir du module d'extension Adaptive Server pour Sybase Central. A partir du module d'extension Adaptive Server, vous pouvez :

- créer une procédure ou fonction SQLJ ;
- exécuter une procédure ou fonction SQLJ ;
- visualiser et modifier les propriétés d'une procédure ou fonction SQLJ ;
- supprimer une procédure ou fonction SQLJ ;
- visualiser les dépendances d'une procédure ou fonction SQLJ ;
- créer des autorisations pour une procédure SQLJ.

Les procédures suivantes expliquent comment créer et visualiser les propriétés d'une routine SQLJ. Vous pouvez visualiser les dépendances et créer et visualiser les autorisations à partir de la feuille de propriétés de la routine.

❖ Création d'une fonction/procédure SQLJ

Tout d'abord, créez et compilez la méthode Java. Installez la classe de méthode dans la base de données avec l'utilitaire installjava. Puis, procédez comme suit :

1. Lancez le module d'extension Adaptive Server et connectez-vous à Adaptive Server.
2. Double-cliquez sur la base de données dans laquelle vous voulez créer la routine.
3. Ouvrez le dossier Procédures SQLJ /Fonctions SQLJ.
4. Double-cliquez sur l'icône Ajouter une nouvelle procédure stockée Java.
5. Utilisez l'assistant Ajouter une nouvelle procédure stockée Java pour créer la procédure ou fonction SQLJ.

   Lorsque vous avez terminé avec l'assistant, le module externe Adaptive Server affiche la routine SQLJ que vous venez de créer dans une fenêtre d'édition. Dans cette fenêtre, vous pouvez modifier et exécuter la routine.

❖ **Pour visualiser les propriétés d'une procédure ou fonction SQLJ**

1 Lancez le module d'extension Adaptive Server et connectez-vous à Adaptive Server.

2 Double-cliquez sur la base de données où la routine est stockée.

3 Ouvrez le dossier Procédures SQLJ/Fonctions SQLJ.

4 Sélectionnez une icône de procédure ou de fonction.

5 Sélectionnez Fichier | Propriétés.

Fonctions SQLJ définies par l'utilisateur

La commande create function spécifie un nom de fonction SQLJ et une signature pour une méthode Java. Vous pouvez utiliser des fonctions SQLJ pour lire et modifier SQL et pour renvoyer une valeur décrite par la méthode référencée.

La syntaxe SQLJ de la commande create function est la suivante :

```
create function [propriétaire].nom_fonction_sql
       ([nom_paramètre_sql type_de_données_sql
          [( longueur)| (précision[, échelle])]
       ([, nom_paramètre_sql type_de_données_sql
          [( longueur ) | ( précision[, échelle]) ]]
       ...])
   returns type_de_données_sql
       [( longueur)| (précision[, échelle])]
   [modifies sql data]
   [returns null on null input |
      called on null input]
   [deterministic | not deterministic]
   [exportable]
   language java
   parameter style java
   external name 'nom_méthode_java
       [([type_de_données_java[ {,
type_de_données_java }
       ...])]'
```

Lorsque vous créez une fonction SQLJ :

- La **signature de fonction SQL** est le type de données SQL *type_de_données_sql* de chaque paramètre de fonction.

- Pour respecter la norme ANSI, n'incluez pas le signe @ devant les noms de paramètres.

  Sybase ajoute un signe @ en interne pour supporter la liaison de nom de paramètre. Le signe @ est visible lorsque vous utilisez la commande sp_help pour imprimer les informations sur la procédure stockée SQLJ.

- Lorsque vous créez une fonction SQLJ, vous devez inclure les parenthèses autour des informations *nom_paramètre_sql* et *type_de_données_sql* – même si vous n'incluez pas ces informations.

  Par exemple :

  ```
  create function sqlj_fc()
     language java
     parameter style java
  external name 'SQLJExamples.method'
  ```

- La clause modifies sql data spécifie que la méthode appelle des opérations SQL et lit et modifie les données SQL. Il s'agit de la valeur par défaut. Vous n'êtes pas tenu de l'inclure, sauf pour compatibilité syntaxique avec la norme SQLJ Partie 1.

- returns null on null input et called on null input indiquent comment Adaptive Server traite les arguments NULL d'un appel de fonction. returns null on null input spécifie que, si la valeur d'un argument est NULL lors de l'exécution, la valeur de la fonction renvoyée est définie sur NULL et le corps de la fonction n'est pas appelé. called on null input est la valeur par défaut. Elle spécifie que la fonction est appelée sans tenir compte des valeurs NULL des arguments.

  Les appels de fonction et les valeurs NULL des arguments sont décrits en détail dans la section "Gestion des valeurs NULL dans l'appel de fonction", page 95.

- Vous pouvez inclure les mots-clés deterministic ou not deterministic, mais Adaptive Server ne les utilise pas. Ils sont inclus pour compatibilité syntaxique avec la norme SQLJ Partie 1.

- Le mot-clé des clauses exportable spécifie que la fonction doit s'exécuter sur un serveur distant utilisant les fonctionnalités de Sybase OmniConnect™. La fonction et la méthode correspondantes doivent être installées sur le serveur distant.

- Les clauses language java et parameter style java spécifient que la méthode référencée est écrite en Java et que les paramètres sont des paramètres Java. Vous *devez* inclure ces phrases lorsque vous créez une fonction SQLJ.
- La clause external name spécifie que la routine n'est pas écrite en SQL et identifie la méthode Java, la classe et le nom de package (le cas échéant).
- La signature de méthode Java spécifie le type de données Java *type_de_données_java* de chaque paramètre de méthode. La signature de méthode Java est facultative. Si ce n'est pas spécifié, Adaptive Server déduit la signature de méthode Java de la signature de fonction SQL.

  Sybase recommande d'inclure la signature de méthode car cette opération traite toutes les conversions de types de données. Reportez-vous à la section "Mappage de type de données Java et SQL", page 108.
- Vous pouvez définir différents noms SQL pour une même méthode Java avec la commande create function et les utiliser de la même manière.

Ecriture de la méthode Java

Avant de créer une fonction SQLJ, vous devez écrire la méthode Java référencée, compiler la classe de la méthode et l'installer dans la base de données.

Dans cet exemple, SQLJExamples.region() mappe le code d'un état à un numéro de région et renvoie ce numéro à l'utilisateur.

```
public static int region(String s)
       throws SQLException {
   s = s.trim();
   if (s.equals("MN") || s.equals("VT") ||
      s.equals("NH") ) return 1;
   if (s.equals("FL") || s.equals("GA") ||
      s.equals("AL") ) return 2;
   if (s.equals("CA") || s.equals("AZ") ||
      s.equals("NV") ) return 3;
   else throw new SQLException
      ("Invalid state code", "X2001");
}
```

Création de la fonction SQLJ

Une fois que vous avez écrit la méthode et que vous l'avez installée, vous pouvez créer la fonction SQLJ. Par exemple :

```
create function region_of(state char(20))
       returns integer
language java parameter style java
external name
       'SQLJExamples.region(java.lang.String)'
```

L'instruction SQLJ create function spécifie un paramètre d'entrée `(state char(20))` et une valeur de retour integer (entière). La signature de fonction SQL est char(20). La signature de méthode Java est java.lang.String.

Appel de fonction

Vous pouvez appeler une fonction SQLJ directement, tout comme s'il s'agissait d'une fonction intégrée. Par exemple :

```
select name, region_of(state) as region
    from sales_emps
where region_of(state)=3
```

Remarque L'ordre de recherche des fonctions dans Adaptive Server est le suivant :

1 Fonctions intégrées

2 Fonctions SQLJ

3 Fonctions Java-SQL appelées directement

Gestion des valeurs d'argument NULL

Les types de données de classe Java et les types de données Java primitifs gèrent différemment les valeurs NULL des arguments.

- **Les types de données objet Java** qui sont des classes, tels que java.lang.Integer, java.lang.String, java.lang.byte[] et java.sql.Timestamp, peuvent contenir les valeurs réelles, ainsi que les valeurs de référence NULL.
- **Les types de données Java primitifs**, tels que boolean, byte, short et int, n'ont pas de représentation pour la valeur NULL. Ces types ne peuvent contenir que des valeurs n'acceptant pas NULL.

Lorsqu'une méthode Java est appelée et que celle-ci transmet une valeur SQL NULL comme argument à un paramètre Java (dont le type de données est une classe Java), cette valeur est transmise comme valeur de référence Java NULL. Toutefois, lorsqu'une valeur SQL NULL est transmise comme argument à un paramètre Java d'un type de données Java primitif, une exception est générée car ce type de données ne possède pas de représentation pour une valeur NULL.

En général, vous écrivez des méthodes Java qui spécifient des classes comme types de données de paramètres Java. Dans ce cas, les valeurs NULL sont gérées sans provoquer d'erreur. Si vous écrivez des fonctions Java utilisant des paramètres Java qui ne peuvent pas gérer les valeurs NULL, vous avez deux possibilités :

- vous pouvez inclure la clause returns null on null input lorsque vous créez la fonction SQLJ, ou
- appeler la fonction SQLJ en utilisant la commande case ou une autre expression conditionnelle pour vérifier la présence de valeurs NULL et appeler la fonction SQLJ uniquement pour les valeurs non NULL.

Vous pouvez gérer les valeurs NULL que vous attendez lorsque vous créez la fonction SQLJ ou lorsque vous l'appelez. Les sections suivantes décrivent les deux cas et font référence à cette méthode :

```
public static String job(int jc)
      throws SQLException {
if (jc==1) return "Admin";
else if (jc==2) return "Sales";
else if (jc==3) return "Clerk";
else return "unknown jobcode";
}
```

Gestion des valeurs NULL lors de la création de fonction

Si vous attendez des valeurs NULL, vous pouvez inclure la clause returns null on null input lorsque vous créez la fonction. Par exemple :

```
create function job_of(jc integer)
   returns varchar(20)
returns null on null input
language java parameter style java
external name 'SQLJExamples.job(int)'
```

Vous pouvez alors appeler job_of comme suit :

```
select name, job_of(jobcode)
    from sales_emps
where job_of(jobcode) <> "Admin"
```

Lorsque le système SQL évalue l'appel job_of(jobcode) pour une ligne de sales_emps dans laquelle la valeur de la colonne jobcode est NULL, l'appel est défini sur la valeur NULL sans réellement appeler la méthode Java SQLJExamples.job. Pour les lignes contenant des valeurs non NULL dans la colonne jobcode, l'appel s'effectue normalement.

Ainsi, lorsqu'une fonction SQLJ créée en utilisant la clause returns null on null input rencontre un argument NULL, le résultat de l'appel de fonction est défini sur NULL et la fonction n'est pas appelée.

Remarque Si vous incluez la clause returns null on null input lorsque vous créez la fonction SQLJ, la clause returns null on null input s'applique à *tous* les paramètres de fonction, y compris les paramètres NULL.

Si vous incluez la clause called on null input (la valeur par défaut), les arguments NULL des paramètres n'acceptant pas NULL génèrent une erreur.

Gestion des valeurs NULL dans l'appel de fonction

Vous pouvez utiliser un appel de fonction conditionnelle pour gérer les valeurs NULL des paramètres n'acceptant pas NULL. L'exemple suivant utilise une expression case :

```
select name,
    case when jobcode is not null
       then job_of(jobcode)
       else null end
from sales_emps where
    case when jobcode is not null
       then job_of(jobcode)
       else null end <> "Admin"
```

Cet exemple suppose que la fonction job_of a été créée en utilisant la clause par défaut called on null input.

Suppression d'un nom de fonction SQLJ

Vous pouvez supprimer le nom de fonction SQLJ pour une méthode Java avec la commande drop function. Par exemple, tapez :

```
drop function region_of
```

Ceci supprime le nom de fonction region_of et sa référence à la méthode SQLJExamples.region. drop function n'a aucun impact sur la méthode ou la classe Java référencée.

Pour plus d'informations sur la syntaxe complète et son utilisation, reportez-vous au document *Manuel de référence*.

Procédures stockées SQLJ

Vous pouvez installer des classes Java dans la base de données à l'aide des fonctionnalités Java-SQL, puis appeler ces méthodes à partir du système SQL ou d'un système client. Vous pouvez également appeler des méthodes (de classe) statiques Java comme procédures stockées SQLJ.

Procédures stockées SQLJ :

- Renvoient des jeux de résultats et/ou des paramètres de sortie au client.
- Se comportent exactement comme les procédures stockées Transact-SQL lorsqu'elles sont exécutées.
- Peuvent être appelées à partir du client en utilisant ODBC, isql ou JDBC.
- Peuvent être appelées dans le serveur à partir d'autres procédures stockées ou du driver JDBC natif d'Adaptive Server.

L'utilisateur final n'a pas besoin de savoir si la procédure appelée est une procédure stockée SQLJ ou Transact-SQL. Toutes les deux sont appelées de la même manière.

La syntaxe SQLJ pour create procedure est la suivante :

```
create procedure [propriétaire.]nom_procédure_sql
    ([[ in | out | inout ] nom_paramètre_sql
       type_de_données_sql [( longueur) |
       (précision[, échelle])]
    [, [ in | out | inout ] nom_paramètre_sql
```

```
    type_de_données_sql [( longueur) |
   (précision[, échelle]) ]]
...])
[modifies sql data]
[dynamic result sets entier]
[deterministic | not deterministic]
language java
parameter style java
external name 'nom_méthode_java
   [([type_de_données_java[,
type_de_données_java
...]])]'
```

Remarque Pour respecter la norme ANSI, la syntaxe de la commande SQLJ create procedure est différente de la syntaxe utilisée pour créer les procédures stockées Sybase Transact-SQL.

Reportez-vous au document *Manuel de référence* pour une description des mots-clés et options de cette commande.

Lors de la création de procédures stockées SQLJ :

- La **signature de procédure SQL** est le type de données SQL *type_de_données_sql* de chaque paramètre de procédure.
- Lors de la création d'une procédure stockée SQLJ, n'incluez pas le signe @ devant les noms de paramètres. Cette méthode est conforme à la norme ANSI.

  Sybase ajoute un signe @ en interne pour supporter la liaison de nom de paramètre. Le signe @ est visible lorsque vous utilisez la commande sp_help pour afficher les informations sur la procédure stockée SQLJ.
- Lorsque vous créez une procédure stockée SQLJ, vous devez inclure les parenthèses autour des informations *nom_paramètre_sql* et *type_de_données_sql*, même si vous n'incluez pas ces informations.

  Par exemple :

  ```
  create procedure sqlj_sproc ()
     language java
     parameter style java
  external name 'SQLJExamples.method1'
  ```

- Vous pouvez inclure les mots-clés modifies sql data pour indiquer que la méthode appelle des opérations SQL et lit et modifie les données SQL. Il s'agit de la valeur par défaut.
- Vous devez inclure l'option *entier* de la commande dynamic result sets lorsque les jeux de résultats doivent être renvoyés à l'environnement appelant. Utilisez la variable *entier* pour spécifier le nombre maximum de jeux de résultats attendus.
- Vous pouvez inclure les mots-clés deterministic ou not deterministic pour compatibilité avec la norme SQLJ. Cependant, Adaptive Server n'utilise pas cette option.
- Vous devez inclure le paramètre language java et les mots-clés style java, pour indiquer à Adaptive Server que la routine externe est écrite en Java et que les conventions d'exécution des arguments transmises à la routine externe sont des conventions Java.
- La clause external name indique que la routine externe est écrite en Java et identifie la méthode Java, la classe et le nom de package (le cas échéant).
- La signature de méthode Java spécifie le type de données Java *type_de_données_java* de chaque paramètre de méthode. La signature de méthode Java est facultative. Si une méthode n'est pas spécifiée, Adaptive Server en déduit une de la procédure SQL.

  Sybase recommande d'inclure la signature de méthode car cette opération traite toutes les conversions de types de données. Pour plus d'informations, reportez-vous à la section "Mappage de type de données Java et SQL", page 108.
- Vous pouvez définir différents noms SQL pour une même méthode Java avec la commande create procedure et les utiliser de la même manière.

Modification de données SQL

Vous pouvez utiliser une procédure stockée SQLJ pour modifier les informations dans la base de données. La méthode référencée par la procédure SQLJ doit être :

- une méthode de type void, ou
- une méthode avec un type de renvoi int (l'incorporation du type de renvoi int est une extension Sybase de la norme SQLJ).

Écriture de la méthode Java

La méthode SQLJExamples.correctStates() exécute une instruction SQL update pour corriger l'orthographe des codes des états. Les paramètres d'entrée spécifient l'ancienne orthographe, ainsi que la nouvelle. correctStates() est une méthode void ; aucune valeur n'est renvoyée à l'appelant.

```
public static void correctStates(String oldSpelling,
          String newSpelling) throws SQLException {

    Connection conn = null;
    PreparedStatement pstmt = null;
    try {
       Class.forName("sybase.asejdbc.ASEDriver");
       conn = DriverManager.getConnection
          ("jdbc:default:connection");
    }
    catch (Exception e) {
       System.err.println(e.getMessage() +
          ":error in connection");
    }
    try {
       pstmt = conn.prepareStatement
          ("UPDATE sales_emps SET state = ?
          WHERE state = ?");
       pstmt.set.String(1, newSpelling);
       pstmt.set.String(2, oldSpelling);
       pstmt.executeUpdate();
    }
    catch (SQLException e) {
       System.err.println("SQLException: " +
       e.getErrorCode() + e.getMessage());
    }
    return;
}
```

Création de la procédure stockée

Pour pouvoir appeler une méthode Java avec un nom SQL, vous devez tout d'abord créer son nom SQL avec la commande SQLJ create procedure. La clause modifies sql data est facultative.

```
create procedure correct_states(old char(20),
       not_old char(20))
    modifies sql data
    language java parameter style java
    external name
       'SQLJExamples.correctStates
       (java.lang.String, java.lang.String)'
```

La procédure correct_states possède une signature de procédure SQL de char(20), char(20). La signature de méthode Java est java.lang.String, java.lang.String.

Appel de la procédure stockée

Vous pouvez exécuter la procédure SQLJ exactement comme une procédure Transact-SQL. Dans cet exemple, la procédure s'exécute à partir de isql :

```
execute correct_states 'GEO', 'GA'
```

Utilisation de paramètres d'entrée et de sortie

Les méthodes Java ne supportent pas les paramètres de sortie. Toutefois, lorsque vous encapsulez une méthode Java dans SQL, vous pouvez exploiter les fonctionnalités de Sybase SQLJ qui autorisent les paramètres d'entrée, de sortie et d'entrée/sortie pour les procédures stockées SQLJ.

Lorsque vous créez une procédure SQLJ, vous identifiez le mode pour chaque paramètre comme in, out ou inout.

- Pour les paramètres d'entrée, utilisez le mot-clé in pour qualifier le paramètre. in est le paramètre par défaut ; si vous ne spécifiez pas un mode de paramètre, Adaptive Server suppose un paramètre d'entrée.
- Pour les paramètres de sortie, spécifiez le mot-clé out.
- Pour des paramètres pouvant transmettre des valeurs en entrée et en sortie à partir de la méthode Java référencée, utilisez le mot-clé inout.

Remarque Vous créez les procédures stockées Transact-SQL en utilisant uniquement les mots-clés in et out. Le mot-clé out correspond au mot-clé SQLJ inout. Reportez-vous aux pages de référence sur create procedure dans le document *Manuel de référence d'Adaptive Server* pour plus d'informations.

Pour créer une procédure stockée SQLJ définissant les paramètres de sortie, vous devez :

- Définir le ou les paramètres de sortie avec l'option out ou inout lorsque vous créez la procédure stockée SQLJ.
- Déclarer ces paramètres comme des tableaux Java dans la méthode Java. SQLJ utilise des tableaux comme conteneurs pour les valeurs des paramètres de sortie de la méthode.

Par exemple, si vous voulez qu'un paramètre entier renvoie une valeur à l'appelant, vous devez spécifier le type du paramètre comme Entier[] (un tableau de Entier) dans la méthode.

L'objet tableau pour un paramètre out ou inout est créé implicitement par le système. Il contient un seul élément. La valeur d'entrée (le cas échéant) est placée dans le premier (et unique) élément du tableau avant que la méthode Java ne soit appelée. Lors du retour d'exécution de la méthode Java, le premier élément est supprimé et affecté à la variable de sortie. En général, la méthode appelée affectera une nouvelle valeur à cet élément.

Les exemples suivants illustrent l'utilisation de paramètres de sortie avec une méthode Java bestTwoEmps() et une procédure stockée best2 faisant référence à cette méthode.

Écriture de la méthode Java

La méthode SQLJExamples.bestTwoEmps() renvoie le nom, l'ID, la région et les ventes des deux employés avec les meilleures performances. Les huit premiers paramètres représentent des paramètres de sortie requérant un tableau conteneur. Le neuvième est un paramètre d'entrée et ne requiert pas de tableau.

```
public static void bestTwoEmps(String[] n1,
   String[] id1, int[] r1,
   BigDecimal[] s1, String[] n2,
   String[] id2, int[] r2, BigDecimal[] s2,
   int regionParm) throws SQLException {

n1[0] = "****";
id1[0] = "";
r1[0] = 0;
s1[0] = new BigDecimal(0):
n2[0] = "****";
id2[0] = "";
r2[0] = 0;
s2[0] = new BigDecimal(0);

try {
   Connection conn = DriverManager.getConnection
      ("jdbc:default:connection");
   java.sql.PreparedStatement stmt =
      conn.prepareStatement("SELECT name, id,"
      + "region_of(state) as region, sales FROM"
      + "sales_emps WHERE"
      + "region_of(state)>? AND"
      + "sales IS NOT NULL ORDER BY sales DESC");
   stmt.setInteger(1, regionParm);
```

```
            ResultSet r = stmt.executeQuery();

            if(r.next()) {
                n1[0] = r.getString("name");
                id1[0] = r.getString("id");
                r1[0] = r.getInt("region");
                s1[0] = r.getBigDecimal("sales");
            }
            else return;

            if(r.next()) {
                n2[0] = r.getString("name");
                id2[0] = r.getString("id");
                r2[0] = r.getInt("region");
                s2[0] = r.getBigDecimal("sales");
            }
            else return;
        }
        catch (SQLException e) {
            System.err.println("SQLException: " +
                e.getErrorCode() + e.getMessage());
            }
        }
```

Création de la procédure SQLJ

Créez un nom SQL pour la méthode bestTwoEmps. Les huit premiers paramètres représentent des paramètres de sortie ; le neuvième est un paramètre d'entrée.

```
create procedure best2
    (out n1 varchar(50), out id1 varchar(5),
    out s1 decimal(6,2), out r1 integer,
    out n2 varchar(50), out id2 varchar(50),
    out r2 integer, out s2 decimal(6,2),
    in region integer)
    language java
    parameter style java
    external name
        'SQLJExamples.bestTwoEmps (java.lang.String,
        java.lang.String, int, java.math.BigDecimal,
        java.lang.String, java.lang.String, int,
        java.math.BigDecimal, int)'
```

La signature de procédure SQL pour best2 est la suivante : varchar(20), varchar(5), decimal (6,2) et ainsi de suite. La signature de méthode Java est String, String, int, BigDecimal et ainsi de suite.

Appel de la procédure

Une fois que la méthode est installée dans la base de données et que la procédure SQLJ faisant référence à la méthode a été créée, vous pouvez appeler la procédure SQLJ.

Lors de l'exécution, le système SQL effectue les opérations suivantes :

1 création des tableaux nécessaires pour les paramètres out et inout lorsque la procédure SQLJ est appelée ;

2 copie du contenu des tableaux de paramètres dans les variables cibles out et inout au retour de la procédure SQLJ.

L'exemple suivant appelle la procédure best2 à partir de isql. La valeur du paramètre d'entrée region indique le numéro de la région.

```
declare @n1 varchar(50), @id1 varchar(5),
    @s1 decimal (6,2), @r1 integer, @n2 varchar(50),
    @id2 varchar(50), @r2 integer, @s2 decimal(6,2),
    @region integer
select @region = 3
execute best2 @n1 out, @id1 out, @s1 out, @r1 out,
    @n2 out, @id2 out, @r2 out, @s2 out, @region
```

Remarque Adaptive Server appelle les procédures stockées SQLJ exactement comme les procédures Transact-SQL. Ainsi, lorsque vous utilisez isql ou tout autre client non Java, vous devez ajouter le signe @ devant les noms de paramètres.

Retour de jeux de résultats

Un jeu de résultats SQL est une séquence de lignes SQL qui est envoyée à l'environnement appelant.

Lorsqu'une procédure stockée Transact-SQL renvoie un ou plusieurs jeux de résultats, ces jeux de résultats sont des sorties implicites de l'appel de procédure. C'est-à-dire, qu'ils ne sont pas déclarés comme paramètres explicites ou valeurs de retour.

Les méthodes Java peuvent renvoyer des objets de jeux de résultats, uniquement comme des valeurs de méthodes explicitement déclarées.

Pour renvoyer un jeu de résultats de style SQL à partir d'une méthode Java, vous devez tout d'abord encapsuler la méthode Java dans une procédure stockée SQLJ. Lorsque vous appelez la méthode comme une procédure stockée SQLJ, les jeux de résultats, qui sont renvoyés par la méthode Java comme des objets de jeu de résultats Java, sont transformés par le serveur en jeux de résultats SQL.

Lorsque vous écrivez la méthode Java à appeler comme une procédure SQLJ renvoyant un jeu de résultats de style SQL, vous devez spécifier un paramètre supplémentaire dans la méthode pour chaque jeu de résultats que la méthode peut renvoyer. Chacun de ces paramètres est un tableau d'un seul élément de la classe Java ResultSet.

Cette section décrit la procédure de base pour écrire une méthode, créer une procédure stockée SQLJ et appeler la méthode. Pour plus d'informations sur le retour de jeux de résultats, reportez-vous à la section "Spécification de signatures de méthode Java explicitement ou implicitement", page 110.

Écriture de la méthode Java

La méthode suivante, SQLJExamples.orderedEmps, appelle SQL, inclut un paramètre ResultSet et utilise des appels JDBC pour sécuriser une connexion et ouvrir une instruction.

```
public static void orderedEmps
       (int regionParm, ResultSet[] rs) throws
       SQLException {

   Connection conn = null;
   PreparedStatement pstmt = null;

   try {
       Class.forName
          ("sybase.asejdbc.ASEDriver");
       Connection conn =
          DriverManager.getConnection
          ("jdbc:default:connection");
   }
   catch (Exception e) {
       System.err.println(e.getMessage()
          + ":error in connection");
   }

   try {
       java.sql.PreparedStatement
          stmt = conn.prepareStatement
```

```
            ("SELECT name, region_of(state)"
            "as region, sales FROM sales_emps"
            "WHERE region_of(state) > ? AND"
            "sales IS NOT NULL"
            "ORDER BY sales DESC");
        stmt.setInt(1, regionParm);
        rs[0] = stmt.executeQuery();
        return;
    }
    catch (SQLException e) {
        System.err.println("SQLException:"
        + e.getErrorCode() + e.getMessage());
    }
    return;
}
```

orderedEmps renvoie un seul jeu de résultats. Vous pouvez également écrire des méthodes qui renvoient plusieurs jeux de résultats. Pour chaque jeu de résultats renvoyé, vous devez procéder comme suit :

- incluez un paramètre séparé de tableau ResultSet dans la signature de méthode ;
- créez un objet Statement pour chaque jeu de résultats ;
- affectez chaque jeu de résultats au premier élément de son tableau ResultSet.

Adaptive Server renvoie toujours l'objet ResultSet ouvert courant pour chaque objetStatement. Lorsque vous créez des méthodes Java qui renvoient des jeux de résultats, vous devez procéder comme suit :

- créez un objet Statement pour chaque jeu de résultats qui doit être renvoyé au client.
- ne fermez pas explicitement les objets ResultSet et Statement. Adaptive Server les ferme automatiquement.

Remarque Adaptive Server s'assure que les objets ResultSet et Statement ne sont pas fermés par le ramasse-miettes, sauf si et jusqu'à ce que les jeux de résultats concernés soient traités et renvoyés au client.

- Si des lignes du jeu de résultats sont extraites par des appels de la méthode Java next(), seules les lignes restantes sont renvoyées au client.

Création de la procédure stockée SQLJ

Lorsque vous créez une procédure stockée SQLJ qui renvoie des jeux de résultats, vous devez spécifier le nombre maximum de jeux de résultats qui peuvent être renvoyés. Dans cet exemple, la procédure ranked_emps renvoie un seul jeu de résultats.

```
create procedure ranked_emps(region integer)
    dynamic result sets 1
    language java parameter style java
    external name 'SQLJExamples.orderedEmps(int,
       ResultSet[]'
```

Si ranked_emps génère plus de jeux de résultats que le nombre spécifié par create procedure, un avertissement s'affiche et la procédure ne renvoie que le nombre de jeux de résultats spécifié. Comme indiqué lors de l'écriture, les procédures stockées SQLJ ranked_emps correspondent à une seule méthode Java.

Remarque Certaines restrictions s'appliquent à la surcharge de méthode lorsque vous déduisez une signature de méthode impliquant des jeux de résultats. Pour plus d'informations, reportez-vous à la section "Mappage de type de données Java et SQL", page 108.

Appel de la procédure

Une fois que la classe de méthode est installée dans la base de données et que la procédure stockée SQLJ faisant référence à la méthode a été créée, vous pouvez appeler la procédure. Vous pouvez écrire l'appel en utilisant tout mécanisme capable de traiter les jeux de résultats SQL.

Par exemple, pour appeler la procédure ranked_emps en utilisant JDBC, vous devez entrer ce qui suit :

```
java.sql.CallableStatement stmt =
    conn.prepareCall("{call ranked_emps(?)}");
stmt.setInt(1,3);
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
    String name = rs.getString(1);
    int.region = rs.getInt(2);
    BigDecimal sales = rs.get.BigDecimal(3);
    System.out.print("Name = " + name);
    System.out.print("Region = " + region);
    System.out.print("Sales = " + sales);
    System.out.printIn():
}
```

La procédure ranked_emps fournit uniquement le paramètre déclaré dans l'instruction create procedure. Le système SQL fournit un tableau vide de paramètres ResultSet et appelle la méthode Java, qui affecte le jeu de résultats de sortie au paramètre du tableau. Une fois la méthode Java terminée, le système SQL renvoie le jeu de résultats dans l'élément de sortie comme un jeu de résultats SQL.

Remarque Vous pouvez renvoyer des jeux de résultats à partir d'un tableau temporaire uniquement lorsque vous utilisez un pilote JDBC externe, tel que jConnect. Vous ne pouvez pas utiliser le pilote JDBC natif d'Adaptive Server pour effectuer cette tâche.

Suppression d'un nom de procédure stockée SQLJ

Vous pouvez supprimer le nom de procédure stockée SQLJ pour une méthode Java avec la commande drop procedure. Par exemple, tapez :

```
drop procedure correct_states
```

Ceci supprime le nom de procédure correct_states et sa référence à la méthode SQLJExamples.correctStates. drop procedure n'a aucun impact sur la méthode ou la classe Java référencée.

Visualisation des informations sur les fonctions et les procédures SQLJ

Plusieurs procédures système stockées fournissent des informations sur les routines SQLJ.

- sp_depends répertorie des objets de base de données référencés par la routine SQLJ et des objets de base de données qui font référence à la routine SQLJ.
- sp_help répertorie chaque nom de paramètre, type, longueur, précision, échelle, ordre de paramètre, mode de paramètre et type de retour de la routine SQLJ.

- sp_helpjava répertorie les informations sur les classes Java et les JAR installés dans la base de données. Le paramètre depends répertorie des références de classes spécifiées, nommées dans la clause external name de l'instruction SQLJ create function ou SQLJ create procedure.
- sp_helprotect signale les autorisations des procédures stockées SQLJ et des fonctions SQLJ.

Pour plus d'informations sur la syntaxe complète et son utilisation, reportez-vous au document *Manuel de référence d'Adaptive Server*.

Fonctions avancées

Les rubriques suivantes présentent une description détaillée des fonctions SQLJ destinées aux utilisateurs avancés.

Mappage de type de données Java et SQL

Lorsque vous créez une procédure stockée ou une fonction qui référence une méthode Java, les types de données des paramètres d'entrée et de sortie ou les jeux de résultats ne doivent pas entrer en conflit lorsque les valeurs sont converties de l'environnement SQL vers l'environnement Java et vice versa. Les règles pour effectuer ce mappage respectent l'implémentation de la norme JDBC. Elles sont décrites ci-dessous et dans le tableau 5-1, page 109.

Chaque paramètre SQL, ainsi que son paramètre Java correspondant doit être mappable. Les types de données SQL et Java sont mappables comme suit :

- Un type de données SQL et un type de données Java primitif sont dits "*simplement mappables*" si cela est spécifié dans le tableau 5-1.
- Un type de données SQL et un type de données Java non primitif sont dits "*mappables Objet*" si cela est spécifié dans le tableau 5-1.
- Un type de données abstrait (ADT) SQL et un type de données Java non primitif sont dits "*mappables ADT*" si tous deux partagent la même classe ou interface.

- Un type de données SQL et un type de données Java sont dits "*mappables Sortie*" si le type de données Java est un tableau et le type SQL est simplement mappable, mappable Objet ou mappable ADT vers le type de données Java. Par exemple, character et String[] sont "mappables Sortie".
- Un type de données Java est dit "*mappable Jeu de résultats*" s'il s'agit d'un tableau de la classe orientée jeu de résultats : java.sql.ResultSet.

En général, une méthode Java est mappable vers SQL si chacun de ses paramètres est mappable vers SQL, si les paramètres de ses jeux de résultats sont dits "mappables Jeu de résultats" et si le type de retour est mappable (fonctions) ou void, ou int (procédures).

Le support pour les types de retour int pour les procédures stockées SQLJ est une extension Sybase de la norme SQLJ Partie 1.

Tableau 5-1 : Types de données SQL et Java simplement mappables et mappables Objet.

Type de données SQL	Types de données Java correspondants	
	Simplement mappables	Mappables Objet
char/unichar		java.lang.String
nchar		java.lang.String
varchar/univarchar		java.lang.String
nvarchar		java.lang.String
text		java.lang.String
numeric		java.math.BigDecimal
decimal		java.math.BigDecimal
money		java.math.BigDecimal
smallmoney		java.math.BigDecimal
bit	boolean	Boolean
tinyint	byte	Integer
smallint	short	Integer
integer	int	Integer
real	float	Float
float	double	Double
double precision	double	Double
binary		byte[]
varbinary		byte[]
datetime		java.sql.Timestamp
smalldatetime		java.sql.Timestamp

Spécification de signatures de méthode Java explicitement ou implicitement

Lorsque vous créez une fonction SQLJ ou une procédure stockée, vous spécifiez en général une signature de méthode Java. Vous pouvez également autoriser Adaptive Server à déduire la signature de méthode Java de la signature SQL de la routine selon les règles de correspondance du type de données de la norme JDBC décrites précédemment dans cette section et dans le tableau 5-1.

Sybase recommande d'inclure la signature de méthode Java car cette opération assure le traitement des conversions de types de données comme spécifié.

Vous pouvez autoriser Adaptive Server à déduire la signature de méthode pour les types de données :

- Simplement mappables
- Mappables ADT
- Mappables Sortie
- Mappables Jeu de résultats

Par exemple, si vous voulez qu'Adaptive Server déduise la signature de méthode pour correct_states, l'instruction create procedure est la suivante :

```
create procedure correct_states(old char(20),
       not_old char(20))
   modifies sql data
   language java parameter style java
   external name 'SQLJExamples.correctStates'
```

Adaptive Server déduit une signature de méthode Java de java.lang.String et java.lang.String. Si vous ajoutez explicitement la signature de méthode Java, l'instruction create procedure est la suivante :

```
create procedure correct_states(old char(20),
       not_old char(20))
   modifies sql data
   language java parameter style java
   external name 'SQLJExamples.correctStates'
      (java.lang.String, java.lang.String)'
```

Vous *devez* explicitement spécifier la signature de méthode Java pour les types de données qui sont "mappables Objet". Sinon, Adaptive Server déduit le type de données primitif, simplement mappable.

Par exemple, la méthode SQLJExamples.job contient un paramètre de type int. (Reportez-vous à la section "Gestion des valeurs d'argument NULL", page 93.) Lorsque vous créez une fonction référençant cette méthode, Adaptive Server déduit une signature Java int et vous n'avez pas besoin de la spécifier.

Toutefois, supposons que le paramètre de SQLJExamples.job était Java Integer, qui est de type "mappable Objet". Par exemple :

```
public class SQLJExamples {
    public static String job(Integer jc)
           throws SQLException...
```

Vous devez alors spécifier la signature de méthode Java lorsque vous créez une fonction qui y fait référence :

```
create function job_of(jc integer)
...
external name
    'SQLJExamples.job(java.lang.Integer)'
```

Retour de jeux de résultats et surcharge de méthode

Lorsque vous créez une procédure stockée SQLJ qui renvoie des jeux de résultats, vous spécifiez le nombre maximum de jeux de résultats qui peuvent être renvoyés.

Si vous spécifiez une signature de méthode Java, Adaptive Server recherche la méthode unique qui correspond au nom de la méthode et à la signature. Par exemple :

```
create procedure ranked_emps(region integer)
    dynamic result sets 1
    language java parameter style java
    external name 'SQLJExamples.orderedEmps
       (int, java.sql.ResultSet[])'
```

Dans ce cas, Adaptive Server résout les types de paramètres en utilisant les conventions de surcharge Java normales.

Supposons, toutefois, que vous ne spécifiez pas la signature de méthode Java :

```
create procedure ranked_emps(region integer)
    dynamic result sets 1
    language java parameter style java
    external name 'SQLJExamples.orderedEmps
```

S'il existe deux méthodes, une avec une signature int, RS[ ], l'autre avec une signature int, RS[ ], RS[ ], Application Server est incapable de faire la distinction entre les deux méthodes et la procédure échoue. Si vous autorisez Adaptive Server à déduire la signature de méthode Java lors du renvoi des jeux de résultats, assurez-vous que *seule une méthode* satisfait les conditions déduites.

Remarque Le nombre de jeux de résultats dynamiques spécifié influence uniquement le nombre maximum de résultats pouvant être renvoyés. Il n'a pas d'incidence sur la surcharge de méthode.

Garantie de validité de la signature

Si une classe installée a été modifiée, Adaptive Server vérifie que la signature de méthode est valide lorsque vous appelez la procédure SQLJ ou la fonction qui référence cette classe. Si la signature d'une méthode modifiée est toujours valide, la routine SQLJ s'exécute avec succès.

Utilisation de la méthode de commande main

En général, dans un client Java, vous commencez les applications Java en exécutant la machine virtuelle Java (VM) à partir de la méthode de commande main d'une classe. La classe JDBCExamples, par exemple, contient une méthode main. Lorsque vous exécutez la classe depuis la ligne de commande, comme suit :

```
java JDBCExamples
```

la méthode de commande main s'exécute.

Remarque Vous ne pouvez pas référencer une méthode Java main dans une instruction SQLJ create function.

Si vous référencez une méthode Java main dans une instruction SQLJ create procedure, la méthode de commande main doit posséder la signature de méthode Java `String[]` comme indiqué ici :

```
public static void main(java.lang.String[]) {
...
}
```

Si la signature de méthode Java est spécifiée dans l'instruction create procedure, elle doit être spécifiée comme `(java.lang.String[])`. Si la signature de méthode Java n'est pas spécifiée, elle est supposée être `(java.lang.String[])`.

Si la signature de procédure SQL contient des paramètres, ces derniers doivent être char, unichar, varchar ou univarchar. Lors de l'exécution, ils sont transmis comme tableau Java de java.lang.String.

Chaque argument que vous ajoutez à la procédure SQLJ doit être char, unichar, varchar, univarchar ou une chaîne littérale car elle est transmise à la méthode main comme un élément du tableau java.lang.String. Vous ne pouvez pas utiliser la clause dynamic result sets lorsque vous créez une procédure main.

Implémentation SQLJ et Sybase : comparaison

Cette section présente les différences entre les spécifications standard de SQLJ Partie 1 et l'implémentation propriétaire de Sybase pour les procédures stockées SQLJ et les fonctions.

Le tableau 5-2 décrit les améliorations Adaptive Server de l'implémentation SQLJ.

Tableau 5-2 : Améliorations Sybase

Catégorie	Norme SQLJ	Implémentation Sybase
Commande create procedure	Supporte uniquement les méthodes Java qui ne renvoient pas de valeur. Le type de retour des méthodes doit être void.	Supporte les méthodes Java qui autorisent une valeur de retour integer. Les méthodes référencées dans create procedure peuvent avoir des types de retour void ou integer.
Commandes create procedure et create function	Supporte uniquement les types de données SQL dans la procédure create ou dans la liste de paramètres create function.	Supporte les types de données SQL et les types de données Java non primitifs comme des types de données abstraits (ADT).
Appel de fonction SQLJ et de procédure SQLJ	Ne supporte pas la conversion SQL implicite vers les types de données SQLJ.	Supporte la conversion SQL implicite vers les types de données SQLJ.
Fonctions SQLJ	N'autorise pas l'exécution de fonctions SQLJ sur les serveurs distants.	Autorise l'exécution de fonctions SQLJ sur les serveurs distants avec les fonctionnalités Sybase OmniConnect.

Catégorie	Norme SQLJ	Implémentation Sybase
Commandes drop procedure et drop function	Requiert le nom complet de la commande : drop procedure ou drop function.	Supporte le nom complet de la fonction et les noms abrégés : drop proc et drop func.

Le tableau 5-3 décrit les fonctions standard SQLJ qui ne sont pas implémentées dans Sybase.

Tableau 5-3 : Fonctionnalités SQLJ non supportées

Catégorie SQLJ	Norme SQLJ	Implémentation Sybase
Commande create function	Permet aux utilisateurs de spécifier le même nom SQL pour plusieurs fonctions SQLJ.	Requiert des noms uniques pour les procédures stockées et les fonctions.
Utilitaires	Supporte sqlj.install_jar, sqlj.replace_jar, sqlj.remove_jar et autres utilitaires similaires pour installer, remplacer et supprimer des fichiers JAR.	Supporte l'utilitaire installjava et la commande remove java Transact-SQL pour l'exécution de fonctions similaires.

Le tableau 5-4 décrit les fonctions standard SQLJ qui sont supportées partiellement dans l'implémentation Sybase.

Tableau 5-4 : Fonctionnalités SQLJ partiellement supportées

Catégorie SQLJ	Norme SQLJ	Implémentation Sybase
Commandes create procedure et create function	Permet aux utilisateurs d'installer différentes classes avec le même nom, dans une même base de données, à condition qu'elles se trouvent dans des fichiers JAR différents.	Requiert des noms de classe uniques dans une même base de données.
Commandes create procedure et create function	Supporte les mots-clés no sql, contains sql, reads sql data et modifies sql data pour spécifier les opérations SQL que la méthode Java peut effectuer.	Supporte modifies sql data uniquement.
Commande create procedure	Supporte java.sql.ResultSet et la déclaration itérative SQL/OLB.	Supporte uniquement java.sql.ResultSet.
Commandes drop procedure et drop function	Supporte le mot-clé restrict, qui requiert que l'utilisateur supprime tous les objets SQL (tableaux, vues et routines) appelant la procédure ou la fonction avant de supprimer la procédure ou la fonction.	Ne supporte pas le mot-clé ni la fonctionnalité restrict.

Le tableau 5-5 décrit les fonctionnalités définies par l'implémentation SQLJ dans l'implémentation Sybase.

Tableau 5-5 : Fonctionnalités SQLJ définies par l'implémentation

Catégorie SQLJ	Norme SQLJ	Implémentation Sybase
Commandes create procedure et create function	Supporte les mots-clésdeterministic | not deterministic, qui spécifient si la procédure ou fonction renvoient toujours les mêmes valeurs pour les paramètres out et inout, ainsi que le résultat de la fonction.	Supporte uniquement la syntaxe pour deterministic | not deterministic, mais ne supporte pas la fonctionnalité.
Commandes create procedure et create function	Vous pouvez valider le mappage entre la signature SQL et la signature de méthode Java lors de l'exécution de la commande create ou lors de l'appel de la procédure ou de la fonction. L'implémentation définit le moment d'exécution de la validation.	Si la classe référencée a été modifiée, toutes les validations sont effectuées lors de l'exécution de la commande create ; l'exécution est ainsi plus rapide.
Commandes create procedure et create function	Peut spécifier les commandes create procedure ou create function dans les fichiers de descripteur de déploiement ou comme instructions SQL DDL. L'implémentation définit le type (ou les types) de support des commandes.	Supporte create procedure et create function comme instructions SQL DDL hors des descripteurs de déploiement.
Appel de routines SQLJ	Lorsqu'une méthode Java exécute une instruction SQL, les conditions d'exception sont générées dans la méthode Java de la sous-classe Exception.SQLException. L'effet de la condition d'exception est défini par l'implémentation.	Suit les règles d'Adaptive Server JDBC.
Appel de routines SQLJ	L'implémentation définit si une méthode Java appelée en utilisant un nom SQL doit s'exécuter avec les privilèges de l'utilisateur ayant créé la procédure ou la fonction, ou avec ceux de l'appelant.	Les procédures et fonctions SQLJ héritent les options de sécurité des procédures stockées SQL et des fonctions Java-SQL, respectivement.
Commandes drop procedure et drop function	Pour spécifier les commandes drop procedure ou drop function dans les fichiers de descripteur de déploiement ou comme instructions SQL DDL. L'implémentation définit le type (ou les types) de support des commandes.	Supporte create procedure et create function comme instructions SQL DDL hors des descripteurs de déploiement.

Classe SQLJExamples

Cette section affiche la classe SQLJExamples utilisée pour illustrer des procédures stockées SQLJ et des fonctions. Vous les trouverez également dans *$SYBASE/$SYBASE_ASE/sample/JavaSql* (UNIX) ou *%SYBASE%\Ase-12_5\sample\JavaSql* (Windows NT).

```
import java.lang.*;
import java.sql.*;
import java.math.*;

static String _url = "jdbc:default:connection";

public class SQLExamples {

    public static int region(String s)
           throws SQLException {
       s = s.trim();
       if (s.equals("MN") || s.equals("VT") ||
          s.equals("NH") ) return 1;
       if (s.equals("FL") || s.equals("GA") ||
          s.equals("AL") ) return 2;
       if (s.equals("CA") || s.equals("AZ") ||
          s.equals("NV") ) return 3;
       else throw new SQLException
          ("Invalid state code", "X2001");

    }
    public static void correctStates
       (String oldSpelling, String newSpelling)
       throws SQLException {

       Connection conn = null;
       PreparedStatement pstmt = null;
       try {
          Class.forName
             ("sybase.asejdbc.ASEDriver");
          conn = DriverManager.getConnection(_url);
       }
       catch (Exception e) {
          System.err.println(e.getMessage() +
             ":error in connection");
       }
       try {
          pstmt = conn.prepareStatement
             ("UPDATE sales_emps SET state = ?
             WHERE state = ?");
```

```
            pstmt.setString(1, newSpelling);
            pstmt.setString(2, oldSpelling);
            pstmt.executeUpdate();
        }
        catch (SQLException e) {
            System.err.println("SQLException: " +
            e.getErrorCode() + e.getMessage());
        }

}
public static String job(int jc)
            throws SQLException {
    if (jc==1) return "Admin";
    else if (jc==2) return "Sales";
    else if (jc==3) return "Clerk";
    else return "unknown jobcode";
}
public static String job(int jc)
            throws SQLException {
if (jc==1) return "Admin";
else if (jc==2) return "Sales";
else if (jc==3) return "Clerk";
else return "unknown jobcode";
}

public static void bestTwoEmps(String[] n1,
    String[] id1, int[] r1,
    BigDecimal[] s1, String[] n2,
    String[] id2, int[] r2, BigDecimal[] s2,
    int regionParm) throws SQLException {

n1[0] = "****";
id1[0] = "";
r1[0] = 0;
s1[0] = new BigDecimal(0):
n2[0] = "****";
id2[0] = "";
r2[0] = 0;
s2[0] = new BigDecimal(0);

try {
    Connection conn = DriverManager.getConnection
        ("jdbc:default:connection");
    java.sql.PreparedStatement stmt =
        conn.prepareStatement("SELECT name, id,"
        + "region_of(state) as region, sales FROM"
        + "sales_emps WHERE"
```

```
            + "region_of(state)>? AND"
            + "sales IS NOT NULL ORDER BY sales DESC");
        stmt.setInteger(1, regionParm);
        ResultSet r = stmt.executeQuery();

        if(r.next()) {
            n1[0] = r.getString("name");
            id1[0] = r.getString("id");
            r1[0] = r.getInt("region");
            s1[0] = r.getBigDecimal("sales");
        }
        else return;

        if(r.next()) {
            n2[0] = r.getString("name");
            id2[0] = r.getString("id");
            r2[0] = r.getInt("region");
            s2[0] = r.getBigDecimal("sales");
        }
        else return;
    }
    catch (SQLException e) {
        System.err.println("SQLException: " +
            e.getErrorCode() + e.getMessage());
        }
    }

    public static void orderedEmps
           (int regionParm, ResultSet[] rs) throws
           SQLException {

        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            Class.forName
                ("sybase.asejdbc.ASEDriver");
            Connection conn =
                DriverManager.getConnection
                ("jdbc:default:connection");
        }
        catch (Exception e) {
            System.err.println(e.getMessage()
                + ":error in connection");
        }
```

```
        try {
            java.sql.PreparedStatement
                stmt = conn.prepareStatement
                ("SELECT name, region_of(state)"
                "as region, sales FROM sales_emps"
                "WHERE region_of(state) > ? AND"
                "sales IS NOT NULL"
                "ORDER BY sales DESC");
            stmt.setInt(1, regionParm);
            rs[0] = stmt.executeQuery();
            return;
        }
        catch (SQLException e) {
            System.err.println("SQLException:"
            + e.getErrorCode() + e.getMessage());
        }
        return;
    }  return;
    }
}
```

CHAPITRE 6

Introduction à XML dans la base de données

Ce chapitre présente le langage XML (eXtensible Markup Language, ainsi que les méthodes de stockage des documents XML dans Adaptive Server et leur création à partir des données SQL.

Rubrique	Page
Introduction	122
Présentation du langage XML	123

D'autres aspects de XML sont décrits dans les chapitres suivants :

- Le chapitre 7, "Sélection de données avec XQL", explique comment sélectionner des données brutes depuis Adaptive Server en utilisant le langage XQL et en les présentant sous la forme d'un document XML.
- Le chapitre 8, "Traitement XML spécialisé", décrit la classe OrderXML, destinée à une application exemple qui exploite les données des commandes client des documents XML et qui a été spécifiquement conçue pour le traitement des documents XML pour les données de bons de commande.
- Le chapitre 9, "XML pour les jeux de résultats SQL", décrit la classe ResultSetXML, qui permet de créer un document XML représentant un jeu de résultats SQL, d'accéder et de mettre à jour ce type de document XML.

Introduction

Comme le langage HTML (Hypertext Markup Language), XML est un langage de marquage et une ramification du langage SGML (Standardized General Markup Language). Cependant, le langage XML est plus complet et obéit à des règles plus strictes ; il vous permet de définir vos propres balises de marquage orientées application. Ces propriétés font du XML un langage particulièrement adapté pour l'échange de données.

Vous pouvez générer des documents formatés en XML à partir des données stockées dans Adaptive Server et, inversement, stocker des documents XML ainsi que les données extraites de ces documents dans Adaptive Server. Adaptive Server vous permet également d'effectuer des recherches dans les documents XML stockés sur le Web.

Adaptive Server utilise le langage XQL (XML Query Language) pour effectuer des recherches dans les documents XML. XQL est un langage de requête reposant sur le chemin, qui effectue des recherches dans les documents XML en suivant la structure XML.

De nombreux outils XML nécessaires à la génération et au traitement des documents XML sont écrits en Java. Java dans Adaptive Server fournit une base stable pour les applications XML-SQL qui utilisent à la fois des outils généraux et des outils spécifiques aux applications.

Le processeur XQL est une fonctionnalité Java livrée avec Adaptive Server. Il vous permet d'interroger et d'accéder aux données XML stockées dans Adaptive Server, puis d'afficher le jeu de résultats ainsi obtenu dans des documents XML. Reportez-vous au chapitre 7, "Sélection de données avec XQL".

Code source et Javadoc

Adaptive Server comprend le code source Java pour les classes XMLResultSet et OrderXML, qui introduisent le codage des classes Java pour le traitement XML. Le code source Java se trouve dans les emplacements suivants :

- *$SYBASE/ASE-12_5/sample/JavaSql* (UNIX)
- *%SYBASE%\Ase-12_5\sample\JavaSql* (Windows NT)

Ces répertoires contiennent également des pages HTML générées par javadoc avec les spécifications des classes, des méthodes et des packages référencés.

Références

Ce chapitre présente un aperçu du langage XML. Pour obtenir des informations plus détaillées, reportez-vous aux documents suivants sur le Web.

- World Wide Web Consortium (W3C), (http://www.w3.org)
- W3C, Document Object Model (DOM), (http://www.w3.org/DOM/)
- W3C, Extensible Markup Language (XML), (http://www.w3.org/XML/)
- W3C, Extensible Stylesheet Language (XSL), (http://www.w3.org/TR/WD-xsl/)

Présentation du langage XML

XML est un langage de marquage et une ramification du langage SGML. Ce langage a été créé en vue d'apporter des fonctionnalités plus évoluées que celles du langage HTML pour la publication sur le Web et le traitement de documents distribués.

Le langage XML est plus simple que le langage SGML, mais plus complexe et plus souple que le langage HTML. Bien que les langages XML et HTML soient généralement interprétables par les mêmes navigateurs et les mêmes processeurs, les caractéristiques du XML rendent ce langage particulièrement adapté pour le partage de documents :

- La structure syntaxique stricte des documents XML facilite la recherche et l'accès aux données. Par exemple, à chacune des balises d'ouverture de tous les éléments doit correspondre une balise de fermeture, par exemple, <*p*> A paragraphe.</*p*>.
- Le langage XML permet de développer et d'utiliser des balises capables de distinguer différents types de données, par exemple, le nombre de clients ou le nombre d'articles.
- Le langage XML autorise la création d'un type de document spécifique à une application, ce qui permet de distinguer un type de document d'un autre.

- Les documents XML permettent d'afficher différentes vues des données XML. Les documents XML, à l'instar de leurs équivalents HTML, ne comprennent que du marquage et un contenu ; ils ne contiennent pas d'instructions de présentation. Ces dernières sont normalement fournies sur le client par des spécifications XSL (Extensible Style Language).

Vous pouvez stocker les documents XML dans Adaptive Server sous les formes suivantes :

- Données XML dans un champ d'un objet Java
- Données XML dans une colonne text ou image
- Données XML dans une colonne char ou varchar
- Données XML dans une colonne image

Exemple de document XML

L'exemple de document Commande est conçu pour une application de traitement des bons de commande. Les clients passent des commandes qui sont identifiées par une date et un ID client. Chaque article de la commande comprend un ID d'article, un nom d'article, une quantité et une unité.

Ce document peut s'afficher sous la forme suivante :

ORDER

Date : 4 juillet 2000

Customer ID : 123

Customer Name : Acme Alpha

Articles :

Item ID	Item Name	Quantity
987	Coupleur	5
654	Connecteur	3 douzaines
579	Fermoir	1

Voici une représentation de ces données en XML :

```
<?xml version="1.0"?>
    <Order>
 <Date>2000/07/04</Date>
 <CustomerId>123</CustomerId>
 <CustomerName>Acme Alpha</CustomerName>
```

```
<Item>
 <ItemId>987</ItemId>
 <ItemName>Coupleur</ItemName>
 <Quantity>5</Quantity>
 </Item>
<Item>
 <ItemId>654</ItemId>
 <ItemName>Connecteur</ItemName>
 <Quantity unit="12">3</Quantity>
 </Item>
<Item>
 <ItemId>579</ItemId>
 <ItemName>Fermoir</ItemName>
 <Quantity>1</Quantity>
 </Item>
</Order>
```

Le document XML se distingue par deux caractéristiques uniques :

- Le document XML n'indique pas le type, le style ou la couleur de l'affichage des éléments.
- Les balises de marquage sont strictement imbriquées. A chaque balise d'ouverture (<*balise*>) correspond une balise de fermeture (</*balise*>).

Le document XML correspondant aux données de la commande intègre les composants suivants :

- La déclaration XML, <?xml version="1.0"?>, qui identifie "Commande" comme un document XML.

  XML représente les documents comme des données caractère. Dans chaque document, le codage caractère (jeu de caractères) est spécifié explicitement ou implicitement. Pour spécifier explicitement le jeu de caractères, incluez-le dans la déclaration XML. Par exemple :

  ```
  <?xml version="1.0" encoding="ISO-8859-1">
  ```

  Sinon, le jeu de caractères par défaut, UTF8, est utilisé.

  Remarque Lorsque les jeux de caractères par défaut du client et du serveur diffèrent, Adaptive Server ignore les conversions de jeux de caractères normales, de sorte que le jeu de caractères déclaré continue à correspondre au jeu de caractères réel. Reportez-vous à la section "Jeux de caractères et données XML", page 131.

- Les balises d'élément créées par l'utilisateur, telles que <Order>...</Order>, <CustomerId>...</CustomerId>, <Item>...</Item>.

- Les données texte, telles que "Acme Alpha", "Coupleur" et "579".
- Les attributs intégrés aux balises d'élément, tels que <Quantity unit = "12">. Cette imbrication permet de personnaliser les éléments.

Un document comprenant ces composants ainsi que des balises d'élément strictement imbriquées est appelé un **document respectant les règles XML**. Dans l'exemple ci-dessus, les balises d'élément décrivent les données qu'elles contiennent, et le document ne contient pas d'instruction de présentation.

Voici un autre exemple de document XML :

```
<?xml version="1.0"?>
 <Info>
    <OneTag>1999/07/04</OneTag>
    <AnotherTag>123</AnotherTag>
    <LastTag>Acme Alpha</LastTag>
<Thing>
        <ThingId> 987</ThingId>
        <ThingName>Coupler</ThingName>
        <Amount>5</Amount>
        <Thing/>
<Thing>
    <ThingId>654</ThingId>
    <ThingName>Connecter</ThingName>
 </Thing>
    <Thing>
        <ThingId>579</ThingId>
        <ThingName>Clasp</ThingName>
        <Amount>1</Amount>
    </Thing>
 </Info>
```

Cet exemple, nommé Info, est un document qui respecte les règles XML, dont la structure et les données sont identiques à celles du document XML Commande. Néanmoins, il n'est pas interprétable par un processeur conçu pour les documents Commande car Info utilise une DTD (déclaration de type de données) différente. Pour plus d'informations sur les DTD, reportez-vous à la section "Types de document XML", page 128.

Affichage HTML des données Commande

Prenons l'exemple d'une application de traitement des bons de commande. Les clients passent des commandes identifiées par une date et un ID client et composées d'un ou plusieurs articles, chacun d'eux comportant un ID d'article, un nom d'article, une quantité et des unités.

Les données de cette commande peuvent s'afficher sous la forme suivante :

COMMANDE

Date : 4 juillet 2000

Customer ID : 123

Customer Name : Acme Alpha

Articles :

Item ID	Item Name	Quantity
987	Coupleur	5
654	Connecteur	3 douzaines
579	Fermoir	1

Ces données indiquent que le client "Acme Alpha" dont le Customer ID est "123" a passé le 04/07/2000 une commande de coupleurs, de connecteurs et de fermoirs.

Le texte HTML de cette commande se présente sous la forme suivante :

```
<html>
<body>
<p>ORDER
<p>Date:  July 4, 2000
<p>Customer ID:  123
<p>Customer Name:  Acme Alpha
<p>Items:</p>
<table bgcolor=white align=left border="3"
    cellpadding=3>
<tr><td><B>Item ID   </B></tr>
    <td><B>Item Name   </B></tr>
    <td><B>Quantity   </B>
    </td></td></tr>
<tr><td>987</td>
    <td>Coupler</td>
    <td>5</td></tr>
<tr><td>654</td>
    <td>Connector</td>
    <td>3 dozen</td></tr>
<tr><td>579</td>
    <td>Clasp</td>
    <td>1</td></tr>
</table>
</body>
</html>
```

Ce texte HTML présente certaines restrictions :

- Il contient à la fois des données et des spécifications de présentation.
  - Les éléments Customer Id, Customer Name, Item Names et Quantities sont des données.
  - Les spécifications de présentation regroupent les indications de style de texte (*....*), de couleur (*bcolor=white*) et de structure (*<table>....</table>*, ainsi que les noms de champs supplémentaires tels que "*Customer Name*", etc.
- La structure des documents HTML ne convient pas particulièrement à l'extraction des données.Certains éléments comme les tableaux doivent impérativement comprendre des balises d'ouverture et de fermeture entre crochets, tandis que d'autres, les balises de paragraphe ("*<p>*") par exemple, acceptent des balises de fermeture facultatives.

  Certains éléments comme les balises de paragraphe ("*<p>*") s'appliquent à plusieurs types de données. Il est donc difficile de faire la différence entre le "123" désignant un Customer ID et le "123" représentant un Item Name, sans les informations données par les noms de champs avoisinants.

Ce mélange de données et de présentation ainsi que l'absence d'une structure syntaxique stricte rendent difficile l'adaptation des documents HTML à des styles de présentation différents et l'utilisation de ces documents pour l'échange et le stockage de données. XML est similaire à HTML, à ceci près qu'il présente des restrictions et des extensions qui remédient à ces inconvénients.

Types de document XML

Une **déclaration de type de données** (DTD) définit la structure d'une classe de documents XML et permet de distinguer les classes. Une DTD est une liste de définitions d'éléments et d'attributs spécifiques à une classe. Une fois la DTD établie, vous pouvez la référencer dans un autre document ou l'intégrer au document XML.

La DTD pour les documents Commande XML, abordée à la section "Exemple de document XML" se présente sous la forme suivante :

```
<!ELEMENT Order (Date, CustomerId, CustomerName,Item+)>
 <!ELEMENT Date (#PCDATA)>
 <!ELEMENT CustomerId (#PCDATA)>
 <!ELEMENT CustomerName (#PCDATA)>
 <!ELEMENT Item (ItemId, ItemName, Quantity)>
 <!ELEMENT ItemId (#PCDATA)>
 <!ELEMENT ItemName (#PCDATA)>
 <!ELEMENT Quantity (#PCDATA)>
 <!ATTLIST Quantity units CDATA #IMPLIED>
```

Lue ligne par ligne, cette DTD spécifie que :

- Une commande doit comporter une date (Date), un ID client (Customer ID), un nom de client (Customer Name) et un ou plusieurs articles (Item). Le signe plus (+) indique un ou plusieurs éléments. Les éléments désignés par un signe plus sont obligatoires. Un point d'interrogation au même endroit indique un élément facultatif. Un astérisque accolé à un élément indique que celui-ci peut se présenter zéro ou plusieurs fois. (Par exemple, si le mot "Item*" de la première ligne ci-dessus était doté d'un astérisque, la commande pourrait comporter aucun élément ou un nombre quelconque d'éléments.)
- Les éléments définis par "(#PCDATA)" sont des textes.
- La définition "<ATTLIST…>" de la dernière ligne spécifie que les éléments "quantity" comportent un attribut "units" ; "#IMPLIED", à la fin de la dernière ligne, indique que l'attribut "units" est facultatif.

Les textes des documents XML ne sont soumis à aucune contrainte. Par exemple, il n'existe aucun moyen de spécifier que le texte d'un élément quantitatif doit être numérique, de sorte qu'il est possible de spécifier :

```
<Quantity unit="Baker's dozen">three</Quantity>
<Quantity unit="six packs">plenty</Quantity>
```

Les restrictions relatives au texte des éléments doivent être gérées par les applications de traitement des données XML.

La DTD d'un document XML doit suivre l'instruction <?xml version="1.0"?>. Vous pouvez inclure la DTD à l'intérieur de votre document XML ou référencer une DTD externe.

- Pour référencer une DTD externe, spécifiez ce qui suit :

```
<?xml version="1.0"?>
 <!DOCTYPE Order SYSTEM "Order.dtd">
 <Order>
…
 </Order>
```

- Une DTD encapsulée peut se présenter sous la forme suivante :

```
<?xml version="1.0"?>
 <!DOCTYPE Order [
 <!ELEMENT Order (Date, CustomerId, CustomerName,
Item+)>
 <!ELEMENT Date (#PCDATA)
 <!ELEMENT CustomerId (#PCDATA)>
 <!ELEMENT CustomerName (#PCDATA)>
 <!ELEMENT Item (ItemId, ItemName, Quantity)>
 <!ELEMENT ItemId (#PCDATA)>
 <!ELEMENT ItemName (#PCDATA)>
 <!ELEMENT Quantity (#PCDATA)>
 <!ATTLIST Quantity units CDATA #IMPLIED>

]>
 <Order>
      <Date>2000/07/04</Date>
      <CustomerId>123</CustomerId>
      <CustomerName>Acme Alpha</CustomerName>

 <Item>
      …
 </Item>
 </Order>
```

Les DTD ne sont pas obligatoires avec les documents XML. Cependant, un **document XML conforme** possède une DTD et s'y conforme.

XSL : Présentation des informations XML

Les spécifications XSL permettent de formater des documents XML. Les spécifications XSL (feuilles de style) sont constituées d'un ensemble de règles qui définissent la conversion d'un document XML en document HTML ou en un autre document XML :

- Les spécifications XSL de conversion d'un document XML en un document HTML peuvent spécifier des détails de présentation HTML standard dans le document HTML cible.

- Les spécifications XSL de conversion d'un document XML en un autre document XML peuvent faire correspondre le document XML source à un document XML cible mais avec des noms d'élément et une structure syntaxique différents.

Vous pouvez créer vos propres feuilles de style pour afficher des classes particulières dans certaines applications. Le langage XSL est généralement utilisé avec les applications de présentation plutôt qu'avec des applications d'échange ou de stockage de données.

Jeux de caractères et données XML

Si les jeux de caractères déclarés de votre client et de votre serveur diffèrent, vous devez redoubler d'attention lors de la déclaration du jeu de caractères de vos documents XML.

Chaque document XML possède une valeur de jeu de caractères. Si ce codage n'est pas déclaré lors de la déclaration XML, le jeu par défaut (UTF8) est appliqué. Lors de l'analyse syntaxique des données XML, le processeur XML lit cette valeur et traite les données en conséquence. Lorsque les jeux de caractères par défaut du client et du serveur diffèrent, Adaptive Server ignore les conversions de jeux de caractères normales afin de s'assurer que le jeu de caractères déclaré et le jeu de caractères réel sont identiques.

- Si vous introduisez un document XML dans la base de données en fournissant le texte complet dans la clause "values" d'une instruction insert, Adaptive Server convertit l'intégralité de l'instruction SQL dans le jeu de caractères du serveur avant de procéder à l'insertion. Adaptive Server convertit généralement les textes de cette manière. Vous devez donc vous assurer que le jeu de caractères déclaré du document XML correspond à celui du serveur.
- Si vous introduisez un document XML dans la base de données à l'aide de writetext ou des programmes Open Client CT-Library ou Open Client DB-Library, Adaptive Server reconnaît le document XML de la déclaration XML et *ne convertit pas* le jeu de caractères dans celui du serveur.
- Si vous lisez un document XML à partir de la base de données, Adaptive Server *ne convertit pas* le jeu de caractères des données dans celui du client, afin de préserver l'intégrité du document XML.

Analyseur XML

Vous pouvez analyser des documents XML et extraire leurs données à l'aide d'opérations de chaîne de caractères SQL telles que substring, charindex et patindex. Cependant, il est plus efficace d'utiliser Java dans le système SQL et des outils écrits en Java, comme les analyseurs XML.

Les analyseurs XML permettent de :

- vérifier qu'un document respecte les règles XML ;
- gérer les problèmes de jeu de caractères ;
- générer une représentation Java de l'arbre d'analyse syntaxique d'un document ;
- construire ou modifier l'arbre d'analyse syntaxique d'un document ;
- générer le texte d'un document à partir de son arbre d'analyse syntaxique.

De nombreux analyseurs XML sont fournis avec une licence gratuite ou appartiennent au domaine public. Ils mettent généralement en œuvre deux interfaces standard : Simple API for XML (SAX) et Document Object Model (DOM).

- *SAX* est une interface d'analyse. Elle spécifie les sources d'entrée, les jeux de caractères et les routines de gestion des références externes. Lors de l'analyse, elle génère des événements permettant aux routines utilisateur de traiter le document de manière incrémentielle et renvoie un objet DOM qui constitue l'arbre d'analyse syntaxique du document.
- *DOM* est l'interface pour manipuler l'arbre d'analyse syntaxique d'un document XML. Elle offre des fonctionnalités pour progresser dans l'arbre d'analyse syntaxique et procéder à l'assemblage.

Les applications qui utilisent les interfaces SAX et DOM sont portables sur tous les analyseurs XML.

CHAPITRE 7

Sélection de données avec XQL

Ce chapitre explique comment sélectionner les données brutes d'Adaptive Server en utilisant le langage XQL et les afficher sous forme de document XML.

Rubrique	Page
Accès à l'analyseur XML	134
Définition de la variable d'environnement CLASSPATH	134
Installation de XQL dans Adaptive Server	134
Autres applications du package XQL	142
Méthodes XQL	148

Remarque isql affiche uniquement les 50 premiers caractères d'un jeu de résultats dérivé de données XML. Toutefois, pour une meilleure illustration, les exemples dans ce chapitre affichent tous les caractères du jeu de résultats. Pour visualiser la totalité du jeu de résultats d'un exemple, utilisez la commande com.sybase.xml.xql.XqlDriver pour exécuter la requête. Reportez-vous à la section "Autres applications du package XQL", page 142. Vous pouvez également utiliser le client JDBC qui facilite le stockage du résultat comme une chaîne java.lang.String.

Adaptive Server inclut un moteur de requêtes écrit en Java, que vous pouvez installer dans le serveur ou exécuter hors du serveur. Vous exécutez le moteur hors du serveur comme tout programme Java en la ligne de commande.

Ce chapitre explique tout d'abord l'exécution du moteur de requêtes comme un programme autonome, hors d'Adaptive Server. Pour des instructions sur l'exécution du moteur de requêtes dans Adaptive Server, reportez-vous à la section "Installation de XQL dans Adaptive Server", page 134.

Accès à l'analyseur XML

Indépendamment de la méthode d'installation de votre moteur de requêtes, comme programme autonome ou dans Adaptive Server, vous devez tout d'abord accéder à l'analyseur XML. Sybase recommande l'analyseur *xerces.jar (vs.1.3.1)* disponible à partir de

- *$SYBASE/ASE-12_5/lib/xerces.jar* (UNIX)
- *%SYBASE%\5\ASE-12_5\lib/xerces.jar* (Windows NT)

Vous pouvez télécharger l'analyseur depuis le site:

Xerces Java Parser (http://xml.apache.org/xerces-j).

Vous pouvez également utiliser tout analyseur conforme à SAX 2.0.

Définition de la variable d'environnement CLASSPATH

Pour créer un programme autonome hors d'Adaptive Server, vous devez inclure les répertoires contenant *xerces.jar* et *xml.zip* dans la définition de votre variable d'environnement CLASSPATH. Pour UNIX, tapez :

```
setenv CLASSPATH $SYBASE/ASE-12_5/lib/xerces.jar
$SYBASE/ASE-12_5/lib/xml.zip
```

Pour Windows NT, tapez :

```
set CLASSPATH = D:\%SYBASE%\ASE-12_5\lib\xerces.jar
D:\%SYBASE%\ASE-12_5\lib\xml.zip
```

Installation de XQL dans Adaptive Server

Cette section suppose que vous avez déjà activé Java dans Adaptive Server. Pour plus d'informations, reportez-vous au chapitre 2, "Préparation et maintenance de Java dans la base de données".

L'utilitaire installjava copie un fichier JAR dans le système Adaptive Server et permet à la base de données courante d'utiliser les classes Java contenues dans le fichier JAR. Respectez la syntaxe suivante :

```
installjava
-f nom_fichier
 [-new | -update ]
...
```

où :

- *nom_fichier* est le nom du fichier JAR que vous installez dans le serveur.
- new indique au serveur qu'il s'agit d'un nouveau fichier.
- update indique au serveur que vous mettez à jour un fichier JAR existant.

Pour plus d'informations sur installjava, reportez-vous au manuel Utilitaires.

Pour ajouter le support XML dans Adaptive Server, vous devez installer les fichiers *xml.zip* et *xerces.jar*. Ces fichiers sont présents dans les répertoires *$SYBASE/ASE-12_5/lib/xml.zip* et *$SYBASE/ASE-12_5/lib/xerces.jar.*

Par exemple, pour installer *xml.zip*, vous devez taper :

```
installjava -Usa -P -Snom_serveur -f $SYBASE/ASE-12_5/lib/xml.zip
```

Pour installer *xerces.jar*, vous devez taper :

```
installjava -Usa -P -Snom_serveur -f $SYBASE/ASE-12_5/lib/xerces.jar.
```

Remarque Pour installer *xerces.jar* dans une base de données, vous devez augmenter la taille de tempdb de 10 Mo.

Conversion d'un document XML brut en version analysée

Utilisez la méthode parse() pour convertir et analyser un document XMLtext ou image et stocker le résultat. Utilisez la commande alter table pour convertir le document XML brut. Par exemple :

```
alter table XMLTEXT add xmldoc IMAGE null
update XMLTEXT
set xmldoc = com.sybase.xml.xql.Xql.query.parse(xmlcol)
```

Cet exemple convertit la colonne xmlcol de la table XMLTEXT en données analysées et les stocke dans la colonne xmldoc.

Insertion de documents XML

Utilisez la méthode parse() pour insérer un document XML qui prend le document XML comme argument et renvoie sybase.aseutils.SybXmlStream.

Adaptive Server comporte un mappage implicite entre les données image ou texte et InputStream. Vous pouvez transférer des colonnes de type image ou texte vers parse() sans effectuer de conversion de type de données. La fonction parse() définie par l'utilisateur analyse le document et renvoie la commande sybase.ase.SybXmlStream utilisée par Adaptive Server pour l'écriture des données dans la colonne image. Adaptive Server écrit ces données dans les colonnes image uniquement et pas dans les colonnes text . L'instruction ci-dessous est de type insert , où XMLDAT est une colonne image :

```
insert XMLDAT
values (..,
com.sybase.xml.xql.Xql.parse("<xmldoc></xmldoc>",..))
```

Mise à jour de documents XML

Pour mettre à jour un document, vous devez supprimer les données originales et insérer les nouvelles données. Les mises à jour d'un document ou d'une partie d'un document sont peu fréquentes par rapport au nombre de lectures de ces documents. Une mise à jour est similaire à :

```
update XMLDAT
set xmldoc =
com.sybase.xml.xql.Xql.parse("<xmldoc></xmldoc>")
```

Suppression de documents XML

La suppression d'un document XML est similaire à la suppression d'une colonne de texte. Par exemple, pour supprimer les données dans la table XMLDAT, vous devez taper :

```
delete XMLDAT
```

Besoins en mémoire pour l'exécution du moteur de requêtes dans Adaptive Server

Vous devrez augmenter la mémoire disponible par rapport à la taille des données XML que vous voulez sélectionner et présenter sous forme de document XQL. Pour un document XML typique de 2 Ko, Sybase recommande de définir les paramètres de configuration dans Java Services sur les valeurs indiquées Tableau 7-1. Pour plus d'informations sur les paramètres de configuration, reportez vous au document *Guide d'administration système de Sybase Adaptive*.

Tableau 7-1 : Paramètres de mémoire Java Services

Section	Valeur réinitialisée
enable java	1
size of process object heap	5000
size of shared class heap	5000
size of global fixed heap	5000

Utilisation de XQL

XQL (XML Query Language) a été conçu comme un langage générique de requêtes pour XML. XQL est un langage de requête reposant sur un chemin pour l'adressage et le filtrage des éléments et du texte des documents XML ; XQL est une extension naturelle de la syntaxe XSL. XQL fournit une notation concise et compréhensible pour l'identification d'éléments spécifiques et pour la recherche de nœuds présentant des caractéristiques particulières. La navigation XQL s'effectue à travers des éléments dans l'arbre XML.

Remarque SQL et XQL sont des langages indépendants. Les exemples présentés ici s'appliquent uniquement à XQL.

Les opérateurs XQL les plus communs incluent :

- Opérateur fils, / – indiquent la hiérarchie. L'exemple suivant renvoie des éléments <*livre*> qui sont les fils des éléments <*librairie*> de la colonne xmlcol de la table xmlimage :

```
select
com.sybase.xml.xql.Xql.query("/bookstore/book",
xmlcol)
from xmlimage

<xql_result>
       <book style=autobiography>
<title>S
```

- Opérateur descendant, // – indique que la requête recherche dans tous les niveaux intervenants. Cela signifie qu'une recherche utilisant l'opérateur descendant peut trouver une occurrence d'un élément dans n'importe quel niveau de la structure XML. La requête suivante trouve toutes les instances d'éléments <*emph*> dans un élément <*excerpt*> :

```
select com.sybase.xml.xql.Xql.query
    ("/bookstore/book/excerpt//emph",xmlcol)
from xmlimage

<xql_result>
       <emph>I</emph>
</xql_result>
```

- Opérateur égal, = – spécifie le contenu d'un élément ou la valeur d'un attribut. La requête suivante trouve tous les exemples où "last-name = Bob" :

```
select com.sybase.xml.xql.Xql.query
    ("/bookstore/book/author[last-name='Bob']", xmlcol)
from xmlimage

<xql_result>
       <author>
       <first-name>Joe</first-name>
       <last-name>Bob</last-name>
       <award>Trenton Literary Review Honorable Mention</award></author>
       <author>
       <first-name>Mary</first-name>
       <last-name>Bob</last-name>
       <publication>Selected Short Stories of
       <first-name>Mary</first-name>
       <last-name>Bob</last-name></publication></author>
       <author>
```

```
       <first-name>Toni</first-name>
       <last-name>Bob</last-name>
       <degree from=Trenton U>B.A.</degree>
       <degree from=Harvard>Ph.D.</degree>
       <award>Pulizer</award>
       <publication>Still in Trenton</publication>
       <publication>Trenton Forever</publication></author>
"</xql_result>
```

- Opérateur filtre, [ ] – filtre l'ensemble des nœuds sur sa gauche d'après les conditions entre crochets. Cet exemple trouve toutes les occurrences des auteurs ayant pour prénom Mary et qui sont répertoriés dans un élément de livre :

```
select com.sybase.xml.xql.Xql.query
   ("/bookstore/book[author/first-name = 'Mary']", xmlcol)
from xmlimage
<xql_result>
       <book style=textbook>
       <title>History of Trenton</title>
       <author>
       <first-name>Mary</first-name>
       <last-name>Bob</last-name>
       <publication>Selected Short Stories of
       <first-name>Mary</first-name>
       <last-name>Bob</last-name></publication></author>
<price>55</price></book>
```

- Opérateur sous-script, [*valeur_numérique*] – trouve une instance spécifique d'un élément. Cet exemple trouve le deuxième livre répertorié dans le document XML. Rappelez-vous que XQL est fondé sur zéro et commence la numérotation à partir de 0 :

```
select com.sybase.xml.xql.Xql.query("/bookstore/book[1]", xmlcol)
from xmlimage
Query returned true and the result is
<xql_result>
              <book style=textbook>
              <title>History of Trenton</title>
              <author>
              <first-name>Mary</first-name>
              <last-name>Bob</last-name>
              <publication>Selected Short Stories of
              <first-name>Mary</first-name>
              <last-name>Bob</last-name></publication></author>
              <price>55</price></book>
</xql_result>
```

- Expressions booléennes – vous pouvez utiliser des expressions booléennes à l'intérieur des opérateurs filtres. Par exemple, cette requête renvoie tous les éléments <*author (auteur)*> contenant au moins un <*degree (licence)*> et un <*award (certificat)*> :

```
select com.sybase.xml.xql.Xql.query
    ("/bookstore/book/author[degree and award]", xmlcol)
from xmlimage

<xql_result>
       <author>
       <first-name>Toni</first-name>
       <last-name>Bob</last-name>
       <degree from=Trenton U>B.A.</degree>
       <degree from=Harvard>Ph.D.</degree>
       <award>Pulizer</award>
       <publication>Still in Trenton</publication>
       <publication>Trenton Forever</publication></author>
</xql_result>
```

Structures de requête ayant une influence sur les performances

Cette section décrit différentes utilisations du moteur de requêtes XML.

Exemples

Le placement de la clause where dans une requête a une influence sur le traitement. Par exemple, cette requête sélectionne tous les livres dont l'auteur se prénomme Mary :

```
select com.sybase.xml.xql.Xql.query
       ("/bookstore/book[author/first-name ='Mary']”, xmlcol)
from XMLDAT
where
          com.sybase.xml.xql.Xql.query("/bookstore/book
              [author/first-name= 'Mary']", xmlcol)!=
          convert(com.sybase.xml.xql.Xql, null)>>EmptyResult
<xql_result ><book style="textbook">
       <title>History of Trenton</title>
       <author>
       <first-name>Mary</first-name>
       <last-name>Bob</last-name>
       <publication>
```

```
       Selected Short Stories of
       <first-name>Mary</first-name>
       <last-name>Bob</last-name>
       </publication>
       </author>
       <price>55</price>
</book></xql_result>
```

query() est appelé deux fois, une fois dans la clause where et une autre fois dans la clause select ; ceci signifie que la requête s'exécute deux fois et risque d'être lente pour les documents volumineux.

Ainsi, vous pouvez enregistrer le jeu de résultats dans un objet pendant l'exécution de la requête dans la clause where, puis rétablir le résultat dans la clause select.

Ou bien, vous pouvez écrire une classe, telle que HoldString, produisant la concaténation des résultats obtenus par chaque invocation de com.sybase.xml.xql.Xql.query() pour les documents XML dans chaque ligne :

```
declare @result HoldString
select @result = new HoldString()
select @result>>get()
from XMLDAT
where
    @result>>put(com.sybase.xml.xql.Xql.query
    ("/bookstore/book[author/first-name= 'Mary']",
    xmlcol))!=
convert(com.sybase.xml.xql.Xql,null)>>EmptyResult
```

Sybase conseille de ne pas stocker le jeu de résultats dans la clause where. La requête n'exécute pas toujours la clause where ; ainsi, lorsque vous essayez de récupérer le résultat dans la clause select, vous risquez d'obtenir un jeu de résultats incorrect. HoldString est une classe exemple.

Etant donné qu'Adaptive Server stocke chaque document dans une colonne d'une ligne spécifiée, lorsque la requête parcourt un ensemble de lignes dans la clause where, il est possible que plusieurs lignes correspondent au critère de recherche. Si c'est le cas, la requête renvoie un document de résultats XML séparé pour chaque ligne en question. Par exemple, si vous créez la table suivante :

```
create table XMLTAB ( xmlcol image)
insert XMLTAB values
       ( com.sybase.xml.xql.Xql.parse(<xml><A><B><C>c</C></B></A></xml>));
insert XMLTAB values
       ( com.sybase.xml.xql.Xql.parse(<xml><D><E><C>c</C></E></D></xml>));
```

Exécutez la requête suivante :

```
select com.sybase.xml.xql.Xql.query("//C", xmlcol)
from XMLTAB
```

Vous attendez le jeu de résultats suivant :

```
<xql_result>
<C>c</C>
<C>c</C>
</xql_result>
```

Au lieu de cela, le jeu de résultats renvoie la même ligne deux fois ; une fois provenant de la clause select et une autre fois provenant de la clause where :

```
<xql_result>
<C>c</C>
</xql_result>

<xql_result>
<C>c</C>
</xql_result>
```

Autres applications du package XQL

Remarque Sybase ne supporte pas ces applications du package XQL. Elles requièrent JDK 1.2 ou version supérieure.

Vous pouvez interroger des documents XML à partir de la ligne de commande de l'application autonome com.sybase.xml.xql.XqlDriver.

Vous pouvez utiliser des méthodes du package Java à partir de com.sybase.xml.xql.Xql pour interroger des documents XML dans les applications Java. Vous pouvez également utiliser ces méthodes de package Java pour interroger les documents XML dans Adaptive Server 12.5, utilisant la fonction Java VM (machine virtuelle Java).

com.sybase.xml.xql.XqlDriver peuter et interroger uniquement des documents XML stockés comme fichiers sur votre système local. Vous ne pouvez pas utiliser com.sybase.xml.xql.XqlDriver pour analyser ou interroger des documents XML stockés dans une base de données ou sur le serveur.

Vous pouvez utiliser com.sybase.xml.xql.XqlDriver pour le développement de scripts XQL, ainsi que pour l'apprentissage de XQL. Toutefois, Sybase recommande d'utiliser com.sybase.xml.xql.XqlDriver uniquement comme programme autonome et non pas comme partie d'une autre application Java, car com.sybase.xml.xql.XqlDriver inclut une méthode main(). Un programme Java ne doit inclure qu'une seule méthode main() ; ainsi, si vous incluez com.sybase.xml.xql.XqlDriver dans un autre programme Java comprenant une méthode main(), l'application essaie d'appliquer les deux méthodes main() et une erreur se produit dans Java.

Sybase recommande l'utilisation de la classe com.sybase.xml.xql.Xql dans les applications pour communiquer avec le moteur de requêtes XML. Les méthodes de cette classe sont répertoriées dans la section "Méthodes dans com.sybase.xml.xql.Xql", page 149.

Syntaxe com.sybase.xml.xql.XqlDriver

La syntaxe pour com.sybase.xml.xql.XqlDriver est la suivante :

```
java com.sybase.xml.xql.XqlDriver
-qstring requête_XQL
-validate true | false
-infile string
-outfile string
-help
-saxparser string
```

où :

- qstring indique la requête XQL en cours d'exécution.
- validate vérifie la validité des documents XML.
- infile est le document XML que vous interrogez.
- outfile est le fichier du système d'exploitation où vous stockez le document XML analysé.
- help affiche la syntaxe com.sybase.xml.xql.XqlDriver.
- saxparser spécifie le nom d'un analyseur défini dans CLASSPATH conforme à SAX 2.0.

Pour plus d'informations sur XQL, reportez-vous à la section "Utilisation de XQL", page 137.

Exemples de requêtes

Cette requête sélectionne tous les titres des livres dans *bookstore.xml* :

```
java com.sybase.xml.xql.XqlDriver -qstring "/bookstore/book/title"
       -infile bookstore.xml

Query returned true and the result is

<xql_result>
<title>Seven Years in Trenton</title>
<title>History of Trenton</title>
<title>Trenton Today, Trenton Tomorrow</title>
</xql_result>
```

Cet exemple énumère tous les prénoms des auteurs dans *bookstore.xml*. XQL utilise un système de numérotation à partir de zéro : "0" est le premier élément trouvé dans un fichier.

```
java com.sybase.xml.xql.XqlDriver
       -qstring "/bookstore/book/author/first-name[0]"
       -infile bookstore.xml
Query returned true and the result is

<xql_result>
          <first-name>Joe</first-name>
          <first-name>Mary</first-name>
          <first-name>Toni</first-name>
</xql_result>
```

L'exemple suivant reprend tous les auteurs ayant pour nom "Bob" et qui sont répertoriés dans *bookstore.xml* :

```
java com.sybase.xml.xql.XqlDriver
       -qstring "/bookstore/book/author[last-name='Bob']"
       -infile bookstore.xml

Query returned true and the result is

<xql_result>
       <author>
       <first-name>Joe</first-name>
       <last-name>Bob</last-name>
       <award>Trenton Literary Review Honorable Mention</award></author>
       <author>
       <first-name>Mary</first-name>
       <last-name>Bob</last-name>
       <publication>Selected Short Stories of
       <first-name>Mary</first-name>
```

```
        <last-name>Bob</last-name></publication></author>
        <author>
        <first-name>Toni</first-name>
        <last-name>Bob</last-name>
        <degree from=Trenton U>B.A.</degree>
        <degree from=Harvard>Ph.D.</degree>
        <award>Pulizer</award>
        <publication>Still in Trenton</publication>
        <publication>Trenton Forever</publication></author>
</xql_result>
```

Validation de votre document

L'option validate appelle un analyseur pour vérifier que le document XML que vous interrogez est conforme à sa DTD. Assurez-vous que votre document XML autonome contient une DTD correcte avant de lancer l'option de validation.

Par exemple, cette commande assure que le document *bookstore.xml* respecte sa DTD :

```
java com.sybase.xml.xql.XqlDriver -qstring "/bookstore" -validate
       -infile bookstore.xml
```

Utilisation de XQL dans le développement d'applications autonomes

Vous pouvez utiliser XQL et développer des applications autonomes, des clients JDBC, des JavaBeans et des EJB pour traiter les données XML. Les méthodes query() et parse() dans com.sybase.xml.xql.Xql vous permettent d'interroger et d'analyser des documents XML. Etant donné que vous pouvez écrire des applications autonomes, vous ne dépendez pas d'Adaptive Server pour obtenir le jeu de résultat. Ainsi, vous pouvez interroger les documents XML stockés comme fichiers système ou stockés sur le Web.

Exemple d'application autonome

L'exemple suivant utilise la requête FileInputStream() pour lire le fichier *bookstore.xml*, ainsi que la méthode URL() pour lire une page Web appelée *bookstore.xml* contenant des informations sur tous les livres dans la librairie :

```
String result;
FileInputStream XmlFile = new FileInputStream("bookstore.xml");
if ((result =
             Xql.query("/bookstore/book/author/first-name", XmlFile))
             != Xql.EmptyResult )
{
          System.out.println(result);
}else{
          System.out.println("Query returned false\n");
}

URL _url = new URL("http://mybookstore/bookstore.xml");
if ((result =
             Xql.query("/bookstore/book/author/first-name",url.openStream()))
                 != Xql.EmptyResult )
{
             System.out.println(result);
}else{
             System.out.println("Query returned false\n");
}
```

Exemple de client JDBC

Le fragment de code suivant utilise la méthode Xql.query pour lire la colonne xmlcol dans le fichier XMLTEXT :

```
String selectQuery = "select xmlcol from XMLTEXT";
Statement stmt = _con.createStatement();
ResultSet rs = (SybResultSet)stmt.executeQuery(selectQuery);
String result;

InputStream is = null;
while ((rs != null) && (rs.next()))
{
          is = rs.getAsciiStream(1);
          result = Xql.query("/bookstore/book/author", is);
}
```

L'exemple suivant suppose que les données XML analysées sont stockées dans une colonne image de la table XMLDOC. Cette application extrait une colonne image comme flux binaire ; toutefois, elle ne l'analyse pas pendant la requête car elle identifie le contenu du flux binaire comme un document XML analysé. Comme alternative, l'application crée une instance SybXmlStream, puis exécute la requête. Ces opérations sont effectuées en utilisant la méthode Xql.query(), sans aucune entrée de la part de l'utilisateur.

```
String selectQuery = "select xmlcol from XMLDOC";
Statement stmt = _con.createStatement();
ResultSet rs = (SybResultSet)stmt.executeQuery(selectQuery);
InputStream is = null;
String result
while ((rs != null) && (rs.next()))
{
          is = rs.getBinaryStream(1);
          result = Xql.query("/bookstore/book/author/first-name", is));
}
```

Exemple d'un exemple EJB

Vous pouvez écrire des fragments de code EJB servant de moteurs de requêtes sur un serveur EJB.

Le fragment de code ci-dessous inclut un EJB appelé *XmlBean*. *XmlBean* inclut la méthode query() qui permet d'interroger tout document XML sur Web. Dans ce composant, query() crée tout d'abord un objet XmlDoc, puis interroge le document.

L'interface distante se présente de la façon suivante :

```
public interface XmlBean extends javax.ejb.EJBObject
{
             /**
          * XQL Method
       */
       public String XQL(String query, URL location) throws
java.rmi.RemoteException
;
}
```

L'implémentation de Bean se présente de la façon suivante :

```
public class XmlBean extends java.lang.Object implements
javax.ejb.SessionBean
{
             ....
             /***
             * XQL Method
```

```
            */
          public String XQL(String query, java.net.URL location) throws
              java.rmi.RemoteException
{
              try {
                     String result;

                     if((result =
                        Xql.query(query, location.openStream())) !=
                        Xql.EmptyResult)
                     {
                           return (result);
                     }else{
                           return (null);
                              }
                     }catch(Exception e){
                        throw new java.rmi.RemoteException(e.getMessage()));
                     }
                  }
....
}
```

Le code client se présente de la façon suivante :

```
....
Context ctx = getInitialContext();
// make the instance of the class in Jaguar
XmlBeanHome -beanHome =
(XmlBeanHome)ctx.lookup("XmlBean");
_xmlBean = (XmlBean)_beanHome.create();
URL u = new URL("http://mywebsite/bookstore.xml");
String res= xmlBean.XQL("/bookstore/book/author/first-name",u);
```

Méthodes XQL

Les méthodes XQL supportées par Sybase et qui sont fournies avec Adaptive Server sont répertoriées ci-dessous. Pour plus d'informations sur ces méthodes, reportez-vous aux sites Web mentionnés à la section Références au chapitre 6, "Introduction à XML dans la base de données".

- attribute
- comment
- element

- id
- node
- pi
- textNode
- textName
- text
- value

Méthodes dans com.sybase.xml.xql.Xql

Les méthodes suivantes sont spécifiques à com.sybase.xml.xql.Xql.

parse(String xmlDoc)

Description

Prend une chaîne Java comme argument et renvoie *SybXmlStream*. Vous pouvez vous servir de cette méthode pour interroger un document en utilisant XQL.

Syntaxe

parse(**String** *xml_document*)

où :

- String est une chaîne de type Java.
- *xml_document* est le document XML contenant la chaîne.

Exemples

L'exemple suivant :

```
SybXmlStream xmlStream = Xql.parse("<xml>..</xml>");)
```

renvoie *SybXmlStream*.

Utilisation

L'analyseur n'effectue pas les opérations suivantes :

- validation du document si une DTD est fournie ;
- analyse de DTD externes ;
- création de liens externes (par exemple, Xlinks) ;
- navigation à travers des IDREF.

parse(InputStream xml_document, boolean validate)

Description

Prend une donnée InputStream et un indicateur booléen comme arguments. L'indicateur signale que l'analyseur doit valider le document selon une DTD spécifiée. Renvoie *SybXmlStream*. Vous pouvez vous servir de cette méthode pour interroger un document en utilisant XQL.

Syntaxe

parse(**InputStream** *xml_document, boolean validate)*

où :

- *InputStream* est un flux d'entrée.
- *xml_document* est le document XML d'origine du flux d'entrée.

Exemples

L'exemple suivant :

```
SybXmlStream is = Xql.parse(new
FileInputStream("file.xml"), true);
```

Renvoie *SybXmlStream*.

Utilisation

- Une valeur vraie (true) dans l'indicateur signale que l'analyseur doit valider le document selon la DTD spécifiée.
- Une valeur fausse (false) dans l'indicateur signale que l'analyseur ne valide pas le document selon la DTD spécifiée.
- L'analyseur n'effectue pas les opérations suivantes :
  - analyse de DTD externes ;
  - création de liens externes (par exemple, Xlinks) ;
  - navigation à travers des IDREF.

query(String query, String xmlDoc)

Description

Interroge un document XML. Utilise le document XML comme l'argument d'entrée.

Syntaxe

query(**String query,***String xmlDoc*)

où :

- *String query* est la chaîne que vous recherchez.
- *String xmldoc* est le document XML que vous interrogez.

Exemples Les commandes suivantes renvoient le résultat sous forme de chaîne Java :

```
String result= Xql.query("/bookstore/book/author",
"<xml>...</xml>");
```

Utilisation Renvoie une chaîne Java.

query(String query, InputStream xmlDoc)

Description Interroge un document XML en utilisant un flux d'entrée comme deuxième argument.

Syntaxe query(**String query,*InputStream xmlDoc***)

où :

- *String query* est la chaîne que vous recherchez.
- *Input Stream xmlDoc* est le document XML que vous interrogez.

Exemples Cet exemple interroge la librairie sur tous les auteurs répertoriés dans *bookstore.Xql*.

```
FileInputStream xmlStream = new FileInputStream("doc.xml");
String result = Xql.query("/bookstore/book/author", xmlStream);
```

L'exemple suivant interroge un document XML sur le Web en utilisant une URL comme argument de recherche :

```
URL xmlURL = new URL("http://mywebsite/doc.xml");
String result = Xql.query("/bookstore/book/author", xmlURL.openStream());
```

Utilisation Renvoie une chaîne Java.

query(String query, SybXmlStream xmlDoc)

Description Interroge un document XML en utilisant un document XML analysé comme deuxième argument.

Syntaxe query(String query, *SybXmlStream xmldoc*)

où :

- *String query* est la chaîne que vous recherchez.

- *xmldoc* est le document XML analysé que vous interrogez.

Exemples — Cet exemple interroge la librairie sur tous les auteurs répertoriés dans *bookstore.Xml*.

```
SybXmlStream xmlStream = Xql.parse("<xml>..</xml>);
String result = Xql.query("/bookstore/book/author",xmlStream);
```

query(String query, JXml jxml)

Description — Interroge un document XML stocké dans un format JXML.

Syntaxe — query(String query, *JXml jxml*)

où :

- *String query* est la chaîne que vous recherchez.
- *JXml jxml* est un objet créé à partir des classes dans *$SYBASE/ASE-12_5/samples/*

Exemples — Cet exemple interroge la librairie sur les auteurs répertoriés dans *bookstore.Xql*.

```
JXml xDoc = new JXml("<xml>...</xml>");;
String result = Xql.query("/bookstore/book/author", xDoc);
```

Utilisation — Vous permet d'exécuter une requête dans un document JXML en utilisant XQL.

sybase.aseutils.SybXmlStream

Description — Définit une interface nécessaire à un flux InputStream pour accéder aux données XML analysées pendant l'interrogation.

Syntaxe — sybase.aseutils.SybXmlStream interface

com.sybase.xml.xql.store.SybMemXmlStream

Description — Conserve le document XML analysé dans la mémoire principale, une fonction de SybXMLStream, fournie par Sybase.

Syntaxe — com.sybase.xml.xql.store.SybMemXmlStream

Utilisation — La méthode parse() renvoie une instance de SybMemXmlStream après l'analyse d'un document XML.

com.sybase.xml.xql.store.SybFileXmlStream

Description — Vous permet d'interroger un fichier où vous avez stocké un document XML analysé.

Syntaxe — com.sybase.xml.xql.store.SybFileXmlStream {*nom_fichier*}

Où *nom_fichier* est le nom du fichier où vous stockez le document XML analysé.

Exemples — Dans les exemples suivants, un membre de RandomAccessFile lit un fichier et positionne le flux de données :

```
SybXmlStream xis = Xql.parse("<xml>..</xml>");
FileOutputStream ofs = new FileOutputStream("xml.data");
((SybMemXmlStream)xis).writeToFile(ofs);

SybXmlStream is = new SybFileXmlStream("xml.data");
String result = Xql.query("/bookstore/book/author", is);
```

setParser(String parserName)

Description — Cette méthode statique spécifie l'analyseur devant être utilisé pour les requêtes. Assurez-vous que la classe d'analyseur spécifié est accessible via CLASSPATH et compatible avec SAX 2.0.

Syntaxe — setParser *(String parserName)*

où *string* est le nom de la classe de l'analyseur.

Exemples

```
Xql.setParser("com.yourcompany.parser")
```

reSetParser

Description	Cette méthode statique rétablit l'analyseur par défaut fourni par Sybase, (*xerces.jar*, Version. 1.3.1).
Syntaxe	reSetParser
Exemples	Cet exemple rétablit l'analyseur par défaut de Sybase.

```
xql.resetParser()
```

CHAPITRE 8

Traitement XML spécialisé

Lorsque vous stockez des documents XML d'un type particulier dans Adaptive Server, il peut être pratique de les mettre à jour ou de les traiter grâce à des méthodes spécialisées. Une méthode consiste à écrire une classe Java dédiée aux mises à jour et au traitement de ce type de document XML. Ce chapitre prend l'exemple d'une classe Java conçue pour les documents OrderXML présentés au chapitre 6, "Introduction à XML dans la base de données".

Rubrique	Page
Classe OrderXml des documents Commande	155
Stockage des documents XML	160
Création et remplissage des tables SQL avec les données Commande	162
Utilisation de la technique de stockage d'éléments	164
Utilisation de la technique de stockage de documents	168
Utilisation de la technique de stockage hybride	173

Cette section décrit en premier lieu la classe OrderXML ainsi que ses méthodes, puis fournit un exemple simple qui illustre la manière dont sont stockés les documents XML ou les données qu'ils contiennent dans une base de données Adaptive Server.

Le code source et le Javadoc pour la classe OrderXML se trouvent dans les emplacements suivants :

- *$SYBASE/ASE-12_5/sample/JavaSql* (UNIX)
- *%SYBASE%\ASE-12_5\sample\JavaSql* (Windows NT)

Classe *OrderXml* des documents Commande

L'exemple ci-après utilise la classe OrderXML et ses méthodes pour les opérations de base sur les documents XML Commande.

OrderXML est une sous-classe de la classe JXml, qui est spécifique des documents XML Commande. Le constructeur OrderXML valide le document de la DTD Commande. Les méthodes de la classe OrderXml assurent le référencement et la mise à jour des éléments du document Commande.

Constructeur OrderXml(String)

Confirme que l'argument *String* contient un document XML Commande correct, puis construit un objet OrderXml contenant ce document. Supposez par exemple que "doc" est une variable de chaîne Java qui contient un document XML Commande, éventuellement contenu dans un fichier :

```
xml.order.OrderXml ox = new xml.order.OrderXml(doc);
```

OrderXml(date, customerid, server)

Les paramètres sont tous de type String.

Cette méthode suppose l'existence d'un ensemble de tables SQL contenant des données Commande. Cette méthode utilise JDBC pour exécuter une requête SQL d'extraction des données Commande correspondant aux paramètres *date* et *customerId* fournis. Puis, cette méthode assemble un document XML Commande à partir des données.

Le paramètre *server* identifie le système Adaptive Server sur lequel exécuter la requête.

- Si vous appelez la méthode dans un environnement client, spécifiez le nom du serveur.
- Si vous appelez la méthode dans le système Adaptive Server (dans une instruction SQL ou dans isql), spécifiez une chaîne vide ou la chaîne "jdbc:default:connection" pour indiquer que la requête doit être exécutée dans le système Adaptive Server courant.

Par exemple :

```
xml.order.OrderXml ox = new OrderXml("990704", "123",
                        "antibes:4000?user=sa");
```

void order2Sql(String ordersTableName, String server)

Extrait les éléments du document Commande et les stocke dans une table SQL créée par la méthode createOrderTable( ). *ordersTableName* est le nom de la table cible, et *server* est tel qu'il est décrit pour le constructeur OrderXml. Par exemple, si *ox* est une variable Java de type OrderXml :

```
ox.order2Sql("current_orders", "antibes:4000?user=sa");
```

Cet appel extrait les éléments du document Commande contenu dans *ox* et utilise JDBC pour les insérer dans les lignes et les colonnes de la table current_orders.

static void createOrderTable(String ordersTableName, String server)

static void createOrderTable (String ordersTableName, String server)

Crée une table SQL dont les colonnes conviennent au stockage des données Commande : customer_id, order_date, item_id, quantity et unit. ordersTableName est le nom de la nouvelle table. Le paramètre *server* est tel qu'il est décrit pour le constructeur OrderXml. Par exemple :

```
xml.order.OrderXml.createOrderTable
          ("current_orders", "antibes:4000?user=sa");
```

String getOrderElement(String elementName)

elementName correspond à "Date", "CustomerId" ou "CustomerName". La méthode renvoie le texte de l'élément. Par exemple, si *ox* est une variable Java de type OrderXml :

```
String customerId = ox.getOrderElement("CustomerId");
String customerName = ox.getOrderElement("CustomerName");
String date = ox.getOrderElement("Date");
```

void setOrderElement (String elementName, String newValue)

elementName est tel qu'il est décrit pour getOrderElement( ). La méthode définit cet élément à newValue. Par exemple, si *ox* est une variable Java de type OrderXml :

```
ox.setOrderElement("CustomerName", "Acme Alpha Consolidated");
ox.setOrderElement("CustomerId", "987a");
ox.setOrderElement("Date", "1999/07/05");
```

Chaîne getItemElement (int itemNumber, String elementName)

itemNumber est l'index d'un article de la commande. *elementName* correspond à "ItemId", "ItemName" ou "Quantity". La méthode renvoie le texte de l'article. Par exemple, si *ox* est une variable Java de type OrderXml :

```
String itemId = ox.getItemElement(2, "ItemId");
String itemName = ox.getItemElement(2, "ItemName");
String quantity = ox.getItemElement(2, "Quantity");
```

void setItemElement (int itemNumber, String elementName, String newValue)

itemNumber et *elementName* sont tels qu'ils sont décrits pour la méthode getItemElement. setItemElement définit l'élément à *newValue*. Par exemple, si *ox* est une variable Java de type OrderXml :

```
ox.setItemElement(2, "ItemId", "44");
ox.setItemElement(2, "ItemName", "cord");
ox.setItemElement(2, "Quantity", "3");
```

Chaîne getItemAttribute (int itemNumber, elementName, attributeName)

itemNumber et *elementName* sont tels qu'ils sont décrits pour getItemElement( ). *elementName* et *attributeName* sont de type String. *attributeName* doit être de type "unit". La méthode renvoie le texte de l'attribut d'unité de l'article.

Remarque Etant donné que les documents Commande n'ont actuellement qu'un seul attribut, le paramètre *attributeName* est inutile. Il est inclus pour illustrer le cas général.

Par exemple, si *ox* est une variable Java de type OrderXml :

```
String itemid - ox.getItemAttribute(2,"unit")
```

void setItemAttribute (int itemNumber, elementName, attributeName, newValue)

itemNumber, *elementName* et *attributeName* sont tels qu'ils sont décrits pour getItemAttribute( ). *elementName*, *attributeName* et *newValue* sont de type String. La méthode définit le texte de l'attribut d'unité de l'article à *newValue*. Par exemple, si *ox* est une variable Java de type OrderXml :

```
ox.setItemAttribute(2, "unit", "13");
```

Les paramètres sont tous de type String. La méthode ajoute un nouvel article au document, avec les valeurs d'élément données. Par exemple, si *ox* est une variable Java de type OrderXml :

```
ox.appendItem("77", "spacer", "5", "12");
```

void appendItem (newItemid, newItemName, newQuantity, newUnit)

Les paramètres sont tous de type String. La méthode ajoute un nouvel article au document, avec les valeurs d'élément données. Par exemple, si *ox* est une variable Java de type OrderXML :

```
ox.appendItem("77", "spacer", "5", "12");
```

void deleteItem(int itemNumber)

itemNumber est l'index d'un article de la commande. La méthode supprime cet article. Par exemple, si *ox* est une variable Java de type OrderXml :

```
ox.deleteItem(2);
```

Stockage des documents XML

Pour échanger des données à l'aide de documents XML dans Adaptive Server, vous devez être en mesure de stocker des documents XML ou les données qu'ils contiennent dans la base de données. Pour optimiser l'échange de données, il convient de se poser les questions suivantes :

- *Mappage et stockage* : quel type de correspondance entre les documents XML et les données SQL convient le mieux à votre système ?
- *Considérations relatives au client et au serveur* : est-il préférable que le mappage ait lieu sur le client ou sur le serveur ?
- *Accès au document XML en SQL* : comment accéder aux éléments d'un document XML dans le système SQL ?

Cette section développe chacune de ces considérations ; la suite du chapitre présente les classes et les méthodes utilisables avec des documents XML. Vous y trouverez :

- un exemple simple illustrant les principes de base du stockage de données et de l'échange de documents XML ;
- un exemple générique, personnalisable pour vos propres documents XML.

Mappage et stockage

Trois types de stockage de données XML sont permis dans Adaptive Server : le **stockage d'éléments**, le **stockage de documents** et le **stockage hybride**, qui est une combinaison des deux précédents.

- Stockage d'éléments : cette méthode permet d'extraire des éléments de données d'un document XML et de les stocker sous forme de lignes et de colonnes de données dans Adaptive Server.

  Par exemple, à partir du document XML Commande, vous pouvez créer des tables SQL avec des colonnes pour chaque élément de la commande : Date, CustomerId, CustomerName, ItemId, ItemName, Quantity et Units. Vous traitez ensuite ces données SQL à l'aide d'opérations SQL normales :

  - Pour générer un document XML à partir des données SQL Commande, extrayez-les et créez un document XML à partir de celles-ci.

    - Pour stocker un document XML avec de nouvelles données Commande, extrayez les éléments de ce document et mettez à jour les tables SQL à partir de ces données.
- Stockage de documents : cette méthode permet de stocker un document XML complet dans une colonne SQL.
- Ainsi, à partir du document Commande, vous pouvez créer une ou plusieurs tables SQL comportant une colonne pour les documents Commande. Le type de données de cette colonne sera par exemple :
    - text SQL, ou
    - une classe Java générique conçue pour les documents XML, ou
    - une classe Java spécialement conçue pour les documents XML Commande.
- Stockage hybride : cette méthode permet de stocker un document XML dans une colonne SQL et de placer certains de ses éléments de données dans des colonnes distinctes pour pouvoir y accéder plus rapidement et plus facilement.

Encore une fois, si l'on reprend l'exemple Commande, vous pouvez créer des tables SQL comme vous le feriez normalement pour le stockage de documents, puis inclure (ou ajouter ultérieurement) une ou plusieurs colonnes dans lesquelles stocker les éléments extraits des documents Commande.

Avantages et inconvénients des options de stockage

Chaque option de stockage présente des avantages et des inconvénients. Choisissez les options les mieux adaptées à l'opération que vous effectuez.

- Si vous stockez des éléments, toutes les données du document XML sont disponibles comme des données SQL normales sur lesquelles vous pouvez effectuer des requêtes et des mises à jour à l'aide d'opérations SQL. Cependant, le stockage d'éléments présente l'inconvénient de devoir assembler et désassembler les documents XML à échanger, ce qui crée une charge de travail supplémentaire.
- Le stockage de documents quant à lui élimine la nécessité d'assembler et de désassembler les données à échanger. Cependant, vous devez utiliser des méthodes Java pour référencer ou mettre à jour les éléments des documents XML lorsqu'ils sont en SQL, ce qui est plus long et moins pratique que l'accès SQL direct au stockage d'éléments.

- Le stockage hybride combine les avantages du stockage d'éléments et de documents, mais le stockage redondant des données extraites impose un coût et une complexité non négligeables.

Considérations relatives au client et au serveur

Vous pouvez exécuter des méthodes Java sur un client ou sur un serveur, auquel cas il vous faudra envisager le stockage d'éléments ou le stockage hybride. Le stockage de documents limite, voire supprime, le traitement du document.

- Stockage d'éléments : si vous faites correspondre des éléments d'un document XML à des données SQL, dans la plupart des cas, le document XML sera plus grand que les données SQL. Il est généralement plus efficace d'assembler et de désassembler le document XML sur le client et de ne transférer que les données SQL entre le client et le serveur.
- Stockage hybride : si vous stockez le document XML complet et des éléments extraits, il est généralement préférable d'extraire les données du serveur, au lieu de les transférer à partir du client.

Création et remplissage des tables SQL avec les données Commande

Dans cette section, plusieurs tables sont conçues pour contenir les données des documents XML Commande et nous permettront d'illustrer les techniques de stockage d'éléments, de documents et hybride.

Tables de stockage d'éléments

Les instructions suivantes créent les tables SQL customers, orders et items, dont les colonnes correspondent aux éléments des documents XML Commande.

```
création d'une table customers
    (customer_id varchar(5) not null unique,
    customer_name varchar(50) not null)

create table orders
    (customer_id varchar(5) not null,
    order_date datetime not null,
```

```
    item_id varchar(5) not null,
    quantity int not null,
    unit smallint default 1)

create table items
    (item_id varchar(5) unique,
    item_name varchar(20))
```

Il n'est pas nécessaire d'avoir créé spécialement ces tables pour y placer les documents XML Commande.

Les instructions SQL suivantes remplissent les tables avec les données du document XML exemple Commande (pour plus d'informations, reportez-vous à la section "Exemple de document XML", page 124) :

```
insert into customers values("123", "Acme Alpha")

insert into orders values ("123", "1999/05/07",
    "987", 5, 1)

insert into orders values ("123", "1999/05/07",
    "654", 3, 12)

insert into orders values ("123", "1999/05/07",
    "579", 1, 1)

insert into items values ("987", "Widget")

insert into items values ("654", "Medium connecter")

insert into items values ("579", "Type 3 clasp")
```

Utilisez l'instruction select pour extraire les données des tables :

```
select order_date as Date, c.customer_id as CustomerId,
    customer_name as CustomerName,
    o.item_id as ItemId, i.item_name as ItemName,
    quantity as Quantity, o.unit as unit
 from customers c, orders o, items i
    where c.customer_id=o.customer_id and
    o.item_id=i.item_id
```

Date	CustomerId	CustomerName	ItemId	ItemName	Quantity	Unit
4 juillet 1999	123	Acme Alpha	987	Coupleur	5	1
4 juillet 1999	123	Acme Alpha	654	Connecteur	3	12
4 juillet 1999	123	Acme Alpha	579	Fermoir	1	1

Tables de stockage de documents et tables de stockage hybride

L'instruction SQL suivante crée une table SQL permettant de stocker des documents XML Commande complets, avec ou sans les éléments extraits (pour le stockage hybride).

```
create table order_docs
    (id char(10) unique,
    customer_id varchar(5) null,

-- For an extracted "CustomerId" element
    order_doc xml.order.OrderXml)
```

Utilisation de la technique de stockage d'éléments

Cette section décrit la technique de stockage d'éléments utilisée pour créer une passerelle entre le langage XML et le langage SQL.

- La section "Assemblage de documents Commande à partir de données SQL", page 164 traite de l'assemblage d'un document XML Commande à partir de données SQL.
- La section "Conversion d'un document XML Commande en données SQL", page 166 traite du désassemblage d'un document XML Commande en données SQL.

Assemblage de documents Commande à partir de données SQL

Dans cet exemple, les méthodes Java génèrent un document XML Commande à partir des données SQL contenues dans les tables créées à la section "Création et remplissage des tables SQL avec les données Commande", page 162.

Une méthode de constructeur de la classe OrderXml mappe les données. Un exemple d'appel à ce constructeur se présentera comme suit :

```
new xml.order.OrderXml("990704", "123",
"antibes:4000?user=sa");
```

Cette méthode de constructeur utilise des opérations JDBC internes pour :

- exécuter une requête SQL sur les données Commande ;
- générer un document XML Commande à partir des données ;
- renvoyer l'objet OrderXml qui contient le document Commande.

Vous pouvez appeler le constructeur OrderXml dans le client ou dans Adaptive Server.

- Si vous appelez le constructeur OrderXml dans le client, il exécute les opérations JDBC en se servant de jConnect pour établir une connexion à Adaptive Server et traiter la requête SQL. Il lit ensuite le jeu de résultats de cette requête et génère le document Commande sur le client.
- Si vous appelez le constructeur OrderXml dans Adaptive Server, il exécute les opérations JDBC en utilisant le pilote JDBC natif pour établir une connexion à la base de données Adaptive Server courante et traiter la requête SQL. Il lit ensuite le jeu de résultats et génère le document Commande dans Adaptive Server.

Génération d'un document Commande sur le client

Conçu pour être mis en œuvre sur le client, main( ) appelle le constructeur de la classe OrderXML pour générer un document XML Commande à partir des données SQL. Ce constructeur exécute une instruction select pour la date et l'ID client fournis et assemble un document XML Commande à partir du résultat.

```
import java.io.*;
import util.*;
public class Sql2OrderClient {
       public static void main (String args[]) {
          try{
             xml.order.Order order =
             new xml.order.OrderXml("990704", "123","antibes:4000?user=sa");
             FileUtil.string2File("Order-sql2Order.xml",order.getXmlText());
          } catch (Exception e) {
                System.out.println("Exception:");
                e.printStackTrace();
          }
       }
 }
```

Génération d'un document Commande sur le serveur

Conçu pour l'environnement serveur, le script SQL suivant appelle le constructeur de la classe OrderXml pour générer un document XML Commande à partir des données SQL :

```
declare @order xml.order.OrderXml
select @order =
    new xml.order.OrderXml('990704', '123','')
insert into order_docs (id, order_doc) values("3",
    @order)
```

Conversion d'un document XML Commande en données SQL

Dans cette section, les éléments sont extraits d'un document XML Commande et stockés dans les lignes et les colonnes des tables SQL Commandes. Les exemples fournis illustrent cette procédure dans les environnement serveur et client.

Les éléments sont convertis à l'aide de la méthode Java order2Sql( ) de la classe OrderXml. Supposons que xmlOrder soit une variable Java de type OrderXml : xmlOrder.order2Sql("orders_received", "antibes:4000?user=sa") ;

L'appel order2Sql() extrait les éléments du document XML Commande contenu dans la variable *xmlOrder*, puis utilise des opérations JDBC pour insérer ces données dans la table SQL orders_received. Vous pouvez appeler cette méthode à partir du client ou d'Adaptive Server :

- Lorsqu'elle est appelée à partir du client, order2Sql() extrait les éléments du document XML Commande dans le client, utilise jConnect pour établir une connexion au système Adaptive Server, puis utilise l'instruction Transact-SQL insert pour insérer les données extraites dans la table.
- Lorsqu'elle est appelée à partir du serveur, order2Sql() extrait les éléments du document XML Commande dans Adaptive Server, utilise le pilote JDBC natif pour établir une connexion au système Adaptive Server courant, puis utilise l'instruction Transact-SQL insert pour insérer les données extraites dans la table.

Conversion du document XML sur le client

Lorsqu'elle est appelée depuis le client, la méthode main( ) de la classe Order2SqlClient crée la table orders_received dont les colonnes permettent de stocker les données Commande. Puis elle extrait les éléments du document XML Commande contenu dans le fichier *Order.xml* et les place dans les lignes et les colonnes de la table *orders_received*. Elle exécute ces actions en appelant la méthode statique OrderXml.createOrderTable( ) et la méthode d'instance order2Sql( ).

```
import util.*;
import xml.order.*;
import java.io.*;
import java.sql.*;
import java.util.*;
public class Order2SqlClient {
       public static void main (String args[]) {
          try{
             String xmlOrder =
                FileUtil.file2String("order.xml");
             OrderXml.createOrderTable("orders_received",
                "antibes:4000?user=sa");
             xmlOrder.order2Sql("orders_received",
                "antibes:4000?user=sa");
          } catch (Exception e) {
          System.out.println("Exception:");
          e.printStackTrace();
          }
       }
 }
```

Conversion du document XML sur le serveur

Lorsqu'il est appelé depuis le serveur, le script SQL suivant appelle le constructeur OrderXml pour générer un document XML Commande à partir des tables SQL, puis appelle la méthode OX.sql2Order( ), qui extrait les données Commande du document XML généré et les insère dans la table orders_received.

```
declare @xmlorder OrderXml
select @xmlorder = new OrderXml('19990704', '123','')
select @xmlorder>>order2Sql('orders_received', '')
```

Utilisation de la technique de stockage de documents

La technique de stockage de documents consiste à stocker un document XML complet dans une colonne SQL. Cette approche vous évite d'avoir à mapper les données entre SQL et XML lorsque des documents sont stockés et extraits, mais l'accès aux éléments stockés peut être long et peu pratique.

Stockage des documents XML Commande dans des colonnes SQL

Cette section fournit des exemples de stockage de documents à partir du client et du serveur.

Insertion d'un document Commande à partir d'un fichier client

L'appel de ligne de commande suivant illustre la manière dont les données XML peuvent être insérées dans Adaptive Server à partir d'un fichier client. Le contenu du fichier *Order.xml* est copié (à l'aide du paramètre -I) dans Adaptive Server et le script SQL est exécuté (à l'aide du paramètre -Q). Le contenu du fichier *Order.xml* est utilisé comme valeur du paramètre point d'interrogation (?).

```
java util.FileUtil -A putstring -I "Order.xml" \
 -Q "insert into order_docs (id, order_doc) \
       values ('1', new xml.order.OrderXml(?)) " \
 -S "antibes:4000?user=sa"
```

Remarque L'appel du constructeur `new xml.order.OrderXml` valide le document XML Commande.

Insertion d'un document Commande généré sur le serveur

Lorsqu'elle est exécutée sur le serveur, la commande SQL suivante génère un document XML Commande à partir des données SQL et l'insère immédiatement dans la colonne de la table order_docs.

```
insert into order_docs (ID, order_doc)
       select "2", new xml.order.OrderXml("990704", "123","")
```

Accès aux éléments des documents XML Commande stockés

Nous avons créé la table order_docs, avec la colonne order_doc. Le type de données de la colonne order_doc est OrderXml, c'est-à-dire une classe Java contenant un document XML Commande.

La classe OrderXml contient plusieurs méthodes d'instance qui permettent de référencer et de mettre à jour les éléments du document XML Commande. Ces méthodes sont décrites à la section "Classe OrderXml des documents Commande", page 155.

La présente section utilise ces méthodes pour mettre à jour le document Commande.

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "Order.dtd">
<Order>
 <Date>1999/07/04</Date>
 <CustomerId>123</CustomerId>
 <CustomerName>Acme Alpha</CustomerName>
 <Item>
 <ItemId> 987</ItemId>
 <ItemName>Coupler</ItemName>
 <Quantity>5</Quantity>
 </Item><Item>
 <ItemId>654</ItemId>
 <ItemName>Connecter</ItemName>
 <Quantity unit="12">3</Quantity>
 </Item><Item>
 <ItemId>579</ItemId>
 <ItemName>Clasp</ItemName>
 <Quantity>1</Quantity>
 </Item>
 </Order>
```

Chaque document XML Commande possède exactement un paramètre *Date*, *CustomerId* et *CustomerName*, et zéro, un ou plusieurs paramètres *Item*, chacun possédant un paramètre *ItemId*, *ItemName* et *Quantity*.

Accès client aux éléments Commande

La méthode main( ) de la classe OrderElements est exécutée sur le client. Elle lit le fichier *Order.xml* dans une variable locale et construit un document OrderXml à partir de celle-ci. Puis, la méthode extrait les éléments "d'en-tête" (Date, CustomerId et CustomerName) ainsi que les éléments du premier article de la Commande, les imprime et met à jour les éléments de la commande avec les nouvelles valeurs.

```
import java.io.*;
import util.*;
public class OrderElements {
       public static void main ( String[] args) {
          try{

          String xml = FileUtil.file2String("Order.xml");
          xml.order.OrderXml ox =
              new xml.order.OrderXml(xml);

          // Get the header elements
          String cname = ox.getOrderElement("CustomerName");
          String cid = ox.getOrderElement("CustomerId");
          String date = ox.getOrderElement("Date");

          // Get the elements for item 1 (numbering from 0)
          String iName1 = ox.getItemElement(1, "ItemName");
          String iId1 = ox.getItemElement(1, "ItemId");
          String iQ1 = ox.getItemElement(1, "Quantity");
          String iU = ox.getItemAttribute(1, "Quantity", "unit");
          System.out.println("\nBEFORE UPDATE: ")
          System.out.println("\n "+date+ " "+ cname + " " +cid);
          System.out.println("\n "+ iName1+" "+iId1+" "
              + iQ1 + " " + iU + "\n");

          // Set the header elements
          ox.setOrderElement("CustomerName", "Best Bakery"
          ox.setOrderElement("CustomerId", "531");
          ox.setOrderElement("Date", "1999/07/31");

          // Set the elements for item 1 (numbering from 0)
          ox.setItemElement(1, "ItemName", "Flange");
          ox.setItemElement(1, "ItemId", "777");
          ox.setItemElement(1, "Quantity","3");
          ox.setItemAttribute(1, "Quantity", "unit", "13");

          //Get the updated header elements
          cname = ox.getOrderElement("CustomerName");
          cid = ox.getOrderElement("CustomerId");
          date = ox.getOrderElement("Date");
```

```
        // Get the updated elements for item 1
        //  (numbering from 0)
        iName1 = ox.getItemElement(1, "ItemName");
        iId1 = ox.getItemElement(1, "ItemId");
        iQ1 = ox.getItemElement(1, "Quantity");
        iU = ox.getItemAttribute(1, "Quantity", "unit");

        System.out.println("\nAFTER UPDATE: ");
        System.out.println("\n "+date+ " "+ cname + " " +cid);
        System.out.println("\n "+ iName1+" "+iId1+" "
            + iQ1 + " " + iU + "\n");

        //Copy the updated document to another file
        FileUtil.string2File("Order-updated.xml", ox.getXmlText())

        } catch (Exception e) {
        System.out.println("Exception:");
        e.printStackTrace();
        }
    }
}
```

Après la mise en œuvre des méthodes dans OrderElements, le document Commande stocké dans le fichier *Order-updated.xml* est :

```
<?xml version="1.0"?>
 <!DOCTYPE Order SYSTEM 'Order.dtd'>
 <Order>
    <Date>1999/07/31</Date>
    <CustomerId>531</CustomerId>
    <CustomerName>Best Bakery</CustomerName>
    <Item>
       <ItemId> 987</ItemId>
       <ItemName>Coupler</ItemName>
       <Quantity>5</Quantity>
    </Item>
    <Item>
       <ItemId>777</ItemId>
       <ItemName>Flange</ItemName>
       <Quantity unit="13">3</Quantity>
    </Item>
    <Item>
       <ItemId>579</ItemId
       <ItemName>Clasp</ItemName
       <Quantity>1</Quantity>
     </Item>
 </Order>
```

Accès serveur aux éléments Commande

L'exemple précédant illustre l'utilisation des méthodes get et set dans un environnement client. Vous pouvez également appeler ces méthodes dans des instructions SQL sur le serveur :

```
select order_doc>>getOrderElement("CustomerId"),
       order_doc>>getOrderElement("CustomerName"),
       order_doc>>getOrderElement("Date")
 from order_docs

select order_doc>>getItemElement(1, "ItemId"),
       order_doc>>getItemElement(1, "ItemName"),
       order_doc>>getItemElement(1, "Quantity"),
       order_doc>>getItemAttribute(1, "Quantity", "unit")
 from order_docs

update order_docs
 set order_doc = order_doc>>setItemElement(1, "ItemName",
       "Wrench")

update order_docs
 set order_doc = order_doc>>setItemElement(2, "ItemId", "967")

select order_doc>>getItemElement(1, "ItemName"),
       order_doc>>getItemElement(2, "ItemId")
 from order_docs

update order_docs
 set order_doc = order_doc>>setItemAttribute(2, "Quantity",
       "unit", "6")

select order_doc>>getItemAttribute(2, "Quantity", "unit")
 from order_docs
```

Ajout et suppression d'articles dans le document XML

La classe Order fournit des méthodes pour ajouter ou supprimer des articles dans le document Commande.

Pour ajouter un nouvel article dans le document Commande, utilisez la méthode appendItem( ), dont les paramètres spécifient les valeurs ItemId, ItemName, Quantity et units du nouvel article :

```
update order_docs
 set order_doc = order_doc>>appendItem("864",
    "Bracket", "3","12")
```

appendItem( ) est une méthode void qui modifie l'instance. Lorsque vous appelez une méthode de ce type dans une instruction update, vous la référencez comme illustré, comme s'il s'agissait d'une méthode dont la valeur est déterminée par la Commande renvoyant l'article mis à jour.

Pour supprimer un article de la commande, utilisez le paramètre deleteItem( ). deleteItem( ) spécifie le numéro de l'article à supprimer. La numérotation commence à zéro, de sorte que la commande suivante supprime de la ligne spécifiée le deuxième article.

```
update order_docs
    set order_doc = order_doc>>deleteItem(1)
 where id = "1"
```

Utilisation de la technique de stockage hybride

Avec la technique de stockage hybride, l'intégralité du document XML est stockée dans une colonne SQL et, simultanément, des éléments de ce document sont stockés dans des colonnes distinctes. Cette technique combine souvent les avantages et les inconvénients du stockage d'éléments et de documents.

La section "Utilisation de la technique de stockage de documents", page 168 montre comment stocker l'intégralité du document XML Commande dans la colonne order_docs.order_doc. Pour utiliser le stockage de documents, référencez et accédez à l'élément *CustomerId* comme suit :

```
select order_doc>>getOrderElement("CustomerID") from order_docs
 where order_doc>>getOrderElement("CustomerID") > "222"
```

Pour accéder à l'élément *CustomerId* plus rapidement et plus efficacement qu'avec l'appel de méthode, mais sans convertir la commande en lignes et en colonnes SQL, procédez comme suit :

1 Ajoutez une colonne dans la table order_docs pour l'élément *customer_id* :

```
alter table order_docs
 add customer_id varchar(5) null
```

2 Mettez à jour cette nouvelle colonne avec les *customerId* extraites.

```
update order_docs
 set customer_id =
 order_doc>>getOrderElement("CustomerId")
```

3 Référencez directement les valeurs *CustomerId* :

```
select customer_id from order_docs
where customer_id > "222"
```

Vous pouvez également définir un index sur la colonne.

Remarque Cette technique n'effectue aucune synchronisation entre la colonne customer_id extraite et l'élément *CustomerId* de la colonne order_doc, si vous mettez à jour l'une ou l'autre de ces valeurs.

CHAPITRE 9

XML pour les jeux de résultats SQL

Ce chapitre décrit la classe ResultSetXML qui vous permet de générer un document XML représentant un jeu de résultats SQL et d'accéder et mettre à jour ce document XML.

Rubrique	Page
Classe ResultSetXML	175
Génération d'un document ResultSet dans le client	185
Génération d'un jeu de résultats dans Adaptive Server	186

Le code source de la classe ResultSetXml réside dans les répertoires suivants :

- *$SBASE/ASE-12_5/sample/JavaSql* (UNIX)
- *%SYBASE%\ASE-12_5\sample\JavaSql* (Windows NT)

Vous pouvez utiliser la classe ResultSetXML pour traiter les jeux de résultats SQL avec XML et comme exemple d'écriture de code Java pour accéder à XML. Dans le chapitre 8, "Traitement XML spécialisé",, vous trouverez un autre exemple de code Java.

Classe ResultSetXML

La classe ResultSetXml est une sous-classe de la classe JXml qui valide un document avec la DTD XML ResultSet et fournit des méthodes d'accès et de mise à jour des éléments du document XML ResultSet.

ResultSetXml(String)

Confirme que l'argument contient un document XML ResultSet correct et construit un objet *ResultSetXml* contenant ce document. Par exemple, si *doc* est une variable Java de type String contenant un document XML ResultSet :

```
xml.resultset.ResultSetXml rsx =
          new xml.resultset.ResultSetXml(doc);
```

Constructeur : ResultSetXml (query, cdataColumns, colNames, server)

- Les paramètres sont tous de type String.
- Le paramètre *query* peut être n'importe quelle requête SQL renvoyant un jeu de résultats.
- Le paramètre *server* identifie le système Adaptive Server sur lequel exécuter la requête. Si vous appelez la méthode dans un environnement client, spécifiez le nom du serveur.

  Si vous appelez la méthode dans Adaptive Server (dans une instruction SQL ou isql), spécifiez une chaîne vide ou la chaîne "jdbc:default:connection" pour indiquer que la requête doit être exécutée dans le système Adaptive Server courant.
- Le paramètre *cdata columns* désigne la colonne correspondant aux sections XML CDATA.
- Le paramètre *colNames* détermine si le document XML résultant doit spécifier les attributs "name" dans les éléments "Column"

Exemple de ResultXml

La méthode établit une connexion au serveur, exécute la requête, extrait le jeu de résultats SQL et construit un objet *ResultSetXml* à partir de ce jeu de résultats.

Par exemple :

```
xml.resultset.ResultSetXml rsx =
       new xml.resultset.ResultSetXml
       ("select 1 as 'a', 2 as 'b', 3 ", "none", "yes",
       "antibes:4000?user=sa");
```

Cet appel de constructeur établit une connexion au serveur spécifié dans le dernier argument, évalue la requête SQL fournie dans le premier argument et renvoie un document XML ResultSet contenant les données du jeu de résultats de la requête.

String toSqlScript (resultTableName, columnPrefix, goOption)

- Les paramètres sont tous de type String.
- Le paramètre *resultTableName* est le nom de la table correspondant aux instructions create et insert. (Les jeux de résultats SQL ne spécifient pas de nom de table car ils peuvent être dérivés de jointures ou d'unions.)
- Le paramètre *columnPrefix* est le préfixe à utiliser dans les noms des colonnes générées qui sont requis pour les colonnes non nommées du jeu de résultats.
- Le paramètre *goOption* indique si le script doit inclure les commandes go (qui sont requises dans isql mais pas dans JDBC).

  La méthode renvoie un script SQL avec une instruction create et la liste des instructions insert qui recréent la liste des données du jeu de résultats.

  Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
rsx>>toSqlScript("systypes_copy", "column_", "yes")
```

String getColumn(int rowNumber, int columnNumber)

- *rowNumber* est l'index d'une ligne dans le jeu de résultats.
- *columnNumber* est l'index d'une colonne dans le jeu de résultats. La méthode renvoie le texte de la colonne spécifiée.

  Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
select rsx>>getColumn(3, 4)
```

String getColumn(int rowNumber, String columnName)

- *rowNumber* est l'index d'une ligne dans le jeu de résultats.
- *columnNumber* est l'index d'une colonne dans le jeu de résultats.

La méthode renvoie le texte de la colonne spécifiée.

Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
select rsx>>getColumn(3, "name")
```

void setColumn (int rowNumber, int columnNumber, newValue)

Les paramètres *rowNumber* et *columnNumber* sont tels qu'ils sont décrits pour getColumn( ).

La méthode définit le texte de la colonne spécifiée à *newValue*.

Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
select rsx = rsx>>setColumn(3, 4, "new value")
```

void setColumn (int rowNumber, String columnName, newValue)

Les paramètres *rowNumber* et *columnName* sont tels qu'ils sont décrits pour getColumn( ).

La méthode définit le texte de la colonne spécifiée à *newValue*.

Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
select rsx = rsx>>setColumn(3, "name", "new value")
```

Boolean allString (int ColumnNumber, String compOp, String comparand)

- *columnNumber* est l'index d'une colonne dans le jeu de résultats.
- *compOp* est un opérateur de comparaison SQL (<, >, =, !=, <=, >=).
- *comparand* est une valeur de comparaison.

La méthode renvoie une valeur indiquant si la comparaison spécifiée est vraie pour toutes les lignes du jeu de résultats.

Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
if rsx>>allString(3, "<", "compare value")...
```

Cette condition est vraie si, pour toutes les lignes du jeu de résultats représenté par *rsx*, la valeur de la colonne 3 est inférieure à "compare value". Cette comparaison est de type String. Des méthodes similaires peuvent être utilisées pour d'autres types de données.

Boolean someString (int ColumnNumber, String compOp, String comparand)

- *columnNumber* est l'index d'une colonne dans le jeu de résultats.
- *compOp* est un opérateur de comparaison SQL (<, >, =, !=, <=, >=).
- *comparand* est une valeur de comparaison.

La méthode renvoie une valeur indiquant si la comparaison spécifiée est vraie pour une ligne du jeu de résultats.

Par exemple, si *rsx* est une variable Java de type ResultSetXml :

```
if rsx>>someString(3, "<", "compare value")...
```

Cette condition est vraie si, pour une ligne du jeu de résultats représenté par *rsx*, la valeur de la colonne 3 est inférieure à "compare value".

Exemple personnalisable pour des jeux de résultats différents

Cette section explique comment stocker des documents XML ou les données qu'ils contiennent dans une base de données Adaptive Server en utilisant la classe ResultSet et ses méthodes pour gérer les jeux de résultats. Vous pouvez personnaliser la classe ResultSet pour votre application de base de données.

Notez la différence entre les types de documents ResultSet et Order :

- Le type de document Order est un exemple simplifié conçu pour une application de gestion des bons de commande et ses méthodes Java sont conçues pour un ensemble spécifique de tables SQL pour les données des bons de commande. Reportez-vous à la section "Classe OrderXml des documents Commande", page 155.

- Le type de document ResultSet est conçu pour de nombreux types de jeux de résultats SQL et les méthodes Java correspondantes incluent les paramètres des différents types de requêtes SQL.

Dans cet exemple, vous allez créer et utiliser des documents XML ResultSet contenant les mêmes données que les documents XML Order.

Dans un premier temps, créez la table orders et ses données :

```
create table orders
       (customer_id varchar(5) not null,
       order_date datetime not null,
       item_id varchar(5) not null,
       quantity int not null,
       unit smallint default 1)
insert into orders values ("123", "1999/05/07", "987", 5, 1)
insert into orders values ("123", "1999/05/07", "654", 3, 12)
insert into orders values ("123", "1999/05/07", "579", 1, 1)
```

Créez également la table SQL suivante pour stocker des documents XML ResultSet complets :

```
create table resultset_docs
    (id char(5),
    rs_doc xml.resultsets.ResultSetXml)
```

Type de document ResultSet

Les documents ResultSet sont composés d'une section ResultSetMetaData suivie d'une section ResultSetData, comme le montre la formulation suivante :

```
<?xml version="1.0"?>
 <!DOCTYPE ResultSet SYSTEM 'ResultSet.dtd'>
 <ResultSet>
<ResultSetMetaData>
…
</ResultSetMetaData>
<ResultSetData>
…
</ResultSetData>
</ResultSet>
```

La section ResultSetMetaData d'un document XML ResultSet est composée des métadonnées SQL renvoyées par les méthodes de la classe JDBC ResultSet. La section ResultSetMetaData de l'exemple de jeu de résultats se présente comme suit :

```
<ResultSetMetaData
    getColumnCount="7">
  <ColumnMetaData
        getColumnDisplaySize="25"
        getColumnLabel="Date"
        getColumnName="Date"
        getColumnType="93"
        getPrecision="0"
        getScale="0"
        isAutoIncrement="false"
        isCurrency="false"
        isDefinitelyWritable="false"
        isNullable="false"
           isSigned="false" />
    <ColumnMetaData
        getColumnDisplaySize="5"
        getColumnLabel="CustomerId"
        getColumnName="CustomerId"
        getColumnType="12"
        getPrecision="0"
        getScale="0"
        isAutoIncrement="false"
        isCurrency="false"
        isDefinitelyWritable="false"
        isNullable="false"
           isSigned="false" />
    <ColumnMetaData
        getColumnDisplaySize="50"
        getColumnLabel="CustomerName"
        getColumnName="CustomerName"
        getColumnType="12"
        getPrecision="0"
        getScale="0"
        isAutoIncrement="false"
        isCurrency="false"
        isDefinitelyWritable="false"
        isNullable="false"
           isSigned="false" />
    <ColumnMetaData
        getColumnDisplaySize="5"
        getColumnLabel="ItemId"
```

```
            getColumnName="ItemId"
            getColumnType="12"
            getPrecision="0"
            getScale="0"
            isAutoIncrement="false"
            isCurrency="false"
            isDefinitelyWritable="false"
            isNullable="false"
               isSigned="false" />
      <ColumnMetaData
            getColumnDisplaySize="20"
            getColumnLabel="ItemName"
            getColumnName="ItemName"
            getColumnType="12"
            getPrecision="0"
            getScale="0"
            isAutoIncrement="false"
            isCurrency="false"
            isDefinitelyWritable="false"
            isNullable="false"
               isSigned="false" />
      <ColumnMetaData
            getColumnDisplaySize="11"
            getColumnLabel="Quantity"
            getColumnName="Quantity"
            getColumnType="4"
            getPrecision="0"
            getScale="0"
            isAutoIncrement="false"
            isCurrency="false"
            isDefinitelyWritable="false"
            isNullable="false"
               isSigned="true" />
      <ColumnMetaData
            getColumnDisplaySize="6"
            getColumnLabel="unit"
            getColumnName="unit"
            getColumnType="5"
            getPrecision="0"
            getScale="0"
            isAutoIncrement="false"
            isCurrency="false"
            isDefinitelyWritable="false"
            isNullable="false"
               isSigned="true" />
   </ResultSetMetaData>
```

Les noms des attributs de ColumnMetaData sont simplement les noms des méthodes de la classe JDBC ResultSetMetaData et les valeurs de ces attributs sont les valeurs renvoyées par ces méthodes.

La section ResultSetData d'un document XML ResultSet est une liste d'éléments *Row*, chacun correspondant à une liste d'éléments *Column*. Le texte d'un élément *Column* est renvoyé par la méthode JDBC getString( ) de la colonne. La section ResultSetData de notre exemple se présente ainsi :

```
<ResultSetData>
     <Row>
       <Column name="Date">1999-07-04 00:00:00.0</Column>
       <Column name="CustomerId">123</Column>
       <Column name="CustomerName">Acme Alpha</Column>
       <Column name="ItemId">987</Column>
       <Column name="ItemName">Coupler</Column>
       <Column name="Quantity">5</Column>
       <Column name="unit">1</Column>
     </Row>
     <Row>
       <Column name="Date">1999-07-04 00:00:00.0</Column>
       <Column name="CustomerId">123</Column>
       <Column name="CustomerName">Acme Alpha</Column>
       <Column name="ItemId">654</Column>
       <Column name="ItemName">Connecter</Column>
       <Column name="Quantity">3</Column>
       <Column name="unit">12</Column>
     </Row>
     <Row>
       <Column name="Date">1999-07-04 00:00:00.0</Column>
       <Column name="CustomerId">123</Column>
       <Column name="CustomerName">Acme Alpha</Column>
       <Column name="ItemId">579</Column>
       <Column name="ItemName">Fermoir</Column>
       <Column name="Quantity">1</Column>
       <Column name="unit">1</Column>
     </Row>
   </ResultSetData>
 <ResultSet>
```

DTD XML du type de document ResultSetXml

La DTD du type de document XML ResultSet est celle-ci :

```
<!ELEMENT ResultSet (ResultSetMetaData,
      ResultSetData)>
 <!ELEMENT ResultSetMetaData (ColumnMetaData)+>
   <!ATTLIST ResultSetMetaData getColumnCount CDATA
      #IMPLIED>
 <!ELEMENT ColumnMetaData  EMPTY>
 <!ATTLIST ColumnMetaData
   getCatalogName CDATA #IMPLIED
   getColumnDisplaySize CDATA #IMPLIED
   getColumnLabel CDATA #IMPLIED
   getColumnName CDATA #IMPLIED
   getColumnType CDATA #REQUIRED
   getColumnTypeName CDATA #IMPLIED
   getPrecision CDATA #IMPLIED
   getScale CDATA #IMPLIED
   getSchemaName CDATA #IMPLIED
   getTablename CDATA #IMPLIED
   isAutoIncrement (true|false) #IMPLIED
   isCaseSensitive (true|false) #IMPLIED
   isCurrency (true|false) #IMPLIED
   isDefinitelyWritable (true|false) #IMPLIED
   isNullable (true|false) #IMPLIED
   isReadOnly (true|false) #IMPLIED
   isSearchable (true|false) #IMPLIED
   isSigned (true|false) #IMPLIED
   isWritable (true|false) #IMPLIED
   >
<!ELEMENT ResultSetData (Row)*>
<!ELEMENT Row (Column)+>
<!ELEMENT Column (#PCDATA)>
<!ATTLIST Column
   null (true | false) "false"
   name CDATA #IMPLIED
```

Utilisation de la technique de stockage de documents

Cette section utilise la table orders pour illustrer la technique de mappage entre des données SQL et des documents XML ResultSet.

Dans la section "Assemblage d'un document XML ResultSet à partir de données SQL", page 185, nous générons un document XML ResultSet à partir des données SQL, en supposant que nous sommes l'auteur du document XML ResultSet. Nous utilisons le document XML ResultSet obtenu pour décrire la DTD ResultSet.

Assemblage d'un document XML ResultSet à partir de données SQL

Vous pouvez utiliser des méthodes Java pour évaluer une requête donnée et générer un jeu de résultats XML à partir des données de la requête. L'exemple suivant utilise une méthode de constructeur de la classe ResultSetXml : Par exemple :

```
new xml.resultset.ResultSetXml
    ("select 1 as 'a', 2 as 'b', 3 ", "none",
    "yes", "antibes:4000?user=sa");
```

La méthode se sert d'opérations JDBC internes pour exécuter la requête d'argument, puis construit le document XML ResultSet à partir des données de la requête.

Ce constructeur peut être appelé dans un client ou dans Adaptive Server :

- Si vous appelez le constructeur dans un client, spécifiez un paramètre serveur identifiant le système Adaptive Server à utiliser pour évaluer la requête. La requête est évaluée dans Adaptive Server, mais le document XML est assemblé dans le client.
- Si vous appelez le constructeur dans Adaptive Server, spécifiez une valeur null ou *jdbc:default:connection* pour le serveur. La requête est évaluée dans le serveur courant et le document XML est assemblé dans ce dernier.

Génération d'un document ResultSet dans le client

La méthode main( ) de la classe OrderResultSetClient est appelée dans un environnement client. main( ) appelle le constructeur de la classe ResultSetXml pour générer un document XML ResultSet. Le constructeur exécute la requête, extrait ses métadonnées et ses données à l'aide des méthodes JDBC ResultSet et assemble un document XML ResultSet.

```
import java.io.*;
import util.*;
public class OrderResultSetClient {
    public static void main (String[] args) {
       try{
          String orderQuery = "select order_date as Date,"
              + "c.customer_id as CustomerId, "
              + "customer_name as CustomerName, "
              + "o.item_id as ItemId, i.item_name as ItemName, "
              + "quantity as Quantity, o.unit as unit "
```

```
            + "from customers c, orders o, items i "
            + "where c.customer_id=o.customer_id and
            + " o.item_id=i.item_id " ;
         xml.resultset.ResultSetXml rsx
               = new xml.resultset.ResultSetXml(orderQuery,
                  "none", "yes", "external",
                  "antibes:4000?user=sa");
         FileUtil.string2File("OrderResultSet.xml",rsx.getXmlText());
      } catch (Exception e) {
         System.out.println("Exception:");
         e.printStackTrace();
      }
   }
}
```

Génération d'un jeu de résultats dans Adaptive Server

Le script SQL suivant appelle le constructeur de la classe ResultSetXml dans un environnement serveur :

```
declare @rsx xml.resultset.ResultSetXml
select @rsx = new xml.resultset.ResultSetXml
       ("select 1 as 'a', 2 as 'b', 3 ", "none", "yes", "");
insert into resultset_docs values ("1", @rsx)
```

Traduction du document XML ResultSet dans le client

La méthode main( ) de ResultSetXml est exécutée dans un environnement client. Elle copie le fichier *OrderResultSet.xml*, construit un objet *ResultSetXml* contenant le contenu de ce fichier et appelle la méthode toSqlScript( ) de cet objet pour générer un script SQL qui recrée les données du jeu de résultats. La méthode stocke le script SQL dans le fichier *order-resultset-copy.sql*.

```
import java.io.*;
import jcs.util.*;
public class ResultSet2Sql{
    public static void main (String[] args) {
       try{
          String xml = FileUtil.file2String("OrderResultSet.xml");
             xml.resultset.ResultSetXml rsx
```

```
            = new xml.resultset.ResultSetXml(xml);
         String sqlScript
            = rsx.toSqlScript("orderresultset_copy", "col_","no");
         FileUtil.string2File("order-resultset-copy.sql",sqlScript);
            ExecSql.statement(sqlScript,"antibes:4000?user=sa");
      } catch (Exception e) {
         System.out.println("Exception:");
         e.printStackTrace();
      }
   }
}
```

Le script SQL généré par ResultSet2Sql est le suivant :

```
set quoted_identifier on
 create table orderresultset_copy (
    Date datetime not null,
    CustomerId varchar (5) not null,
    CustomerName varchar (50) not null,
    ItemId varchar (5) not null,
    ItemName varchar (20) not null,
    Quantity integer not null,
    unit smallint not null
 )
insert into orderresultset_copy values (
    '1999-07-04 00:00:00.0',   '123',
   'Acme Alpha',    '987', 'Widget', 5,  1 )
 insert into orderresultset_copy values (
    '1999-07-04 00:00:00.0',  '123',
   'Acme Alpha',  '654',
   'Connecteur moyen',  3,  12 )
 insert into orderresultset_copy values (
   '1999-07-04 00:00:00.0',  '123',
   'Acme Alpha',  '579',  'Fermoir Type 3',  1,  1 )
```

Le script SQL inclut la commande set quoted_identifier on lorsque le script SQL généré utilise des identificateurs entre guillemets.

Traduction du document XML ResultSet dans Adaptive Server

Le script SQL suivant appelle la méthode toSqlScript( ) dans Adaptive Server, puis crée et remplit une table avec une copie des données du jeu de résultats.

```
declare @rsx xml.resultset.ResultSetXml
select @rsx = rs_doc from resultset_docs where id=1
select @script = @rsx>>toSqlScript("resultset_copy",
    "column_", "no")
declare @I integer
select @I = jcs.util.ExecSql.statement(@script, "")
```

Utilisation de la technique de stockage de documents

Cette section fournit des exemples de stockage de documents XML ResultSet dans des colonnes SQL ainsi que des techniques de référencement et de mise à jour des éléments des colonnes.

Stockage d'un document XML ResultSet dans une colonne SQL

Le script SQL suivant génère un document XML ResultSet et le stocke dans une table :

```
declare @query java.lang.StringBuffer
select @query = new java.lang.StringBuffer()
 -- Les commandes "appends" suivantes génèrent une instruction SQL select
       dans la variable @query
 -- Nous utilisons un cache StringBuffer et la méthode append de sorte que
       la requête @query peut être aussi longue qu'il le faut.
select @query>>append("select order_date as Date,
       c.customer_id as CustomerId, ")
select @query>>append("customer_name as CustomerName, ")
select @query>>append("o.item_id as ItemId, i.item_name as ItemName, ")
select @query>>append("quantity as Quantity, o.unit as unit " )
select @query>>append("from customers c, orders o, items i ")
select @query>>append("where c.customer_id=o.customer_id and
       + "o.item_id=i.item_id  ")
declare @rsx xml.resultset.ResultSetXml
select  @rsx = new xml.resultset.ResultSetXml
       (@query>>toString(), 'none', 'yes', '')
insert into resultset_docs values("1",  @rsx)
```

Accès aux colonnes des documents ResultSet stockés

A la section "Stockage d'un document XML ResultSet dans une colonne SQL", page 188, vous insérez un document XML ResultSet complet dans la colonne rs_doc de la table resultset_docs. Cette section donne des exemples d'utilisation des méthodes de la classe ResultSetXml pour référencer et mettre à jour un document ResultSet stocké.

Appel client

La méthode main( ) de ResultSetElements est exécutée dans un environnement client. Elle copie le fichier *OrderResultSet.xml*, construit un document ResultSetXml à partir de celui-ci, puis accède aux colonnes du document ResultSet et les met à jour.

```
import java.io.*;
import util.*;
public class ResultSetElements {
    public static void main (String[] args) {
       try{
          String xml =
             FileUtil.file2String("OrderResultSet.xml");
          xml.resultset.ResultSetXml rsx
                 = new xml.resultset.ResultSetXml(xml);
          // Obtenir les colonnes contenant les infos sur le client et la date
          String cname = rsx.getColumn(0, "CustomerName");
          String cid   = rsx.getColumn(0, "CustomerId");
          String date = rsx.getColumn(0, "Date");
          // Obtenir les éléments pour l'article 1 (numérotation à partir de 0)
          String iName1 = rsx.getColumn(1, "ItemName");
          String iId1 = rsx.getColumn(1, "ItemId");
          String iQ1 = rsx.getColumn(1, "Quantity");
          String iU = rsx.getColumn(1,  "unit");
          System.out.println("\nBEFORE UPDATE: ");
          System.out.println("\n   "+date+ "  "+ cname + " " +
             cid);
          System.out.println("\n   "+ iName1+" "+iId1+" "
             + iQ1 + " " + iU + "\n");
          //  Obtenir les éléments pour l'article 1 (numérotation à partir de 0)
          rsx.setColumn(1, "ItemName", "Flange");
          rsx.setColumn(1, "ItemId", "777");
          rsx.setColumn(1, "Quantity","3");
          rsx.setColumn(1,  "unit", "13");
          // Obtenir les éléments mis à jour pour l'article 1 (numérotation
             à partir de 0) iName1 = rsx.getColumn(1, "ItemName");
          iId1 = rsx.getColumn(1, "ItemId");
```

```
            iQ1 = rsx.getColumn(1, "Quantity");
            iU = rsx.getColumn(1,  "unit");
            System.out.println("\nAFTER UPDATE: ");
            System.out.println("\n   "+date+ "  "+ cname + " " +
               cid);
            System.out.println("\n   "+ iName1+" "+iId1+" "
               + iQ1 + " " + iU + "\n");

            //  Copier le document mis à jour dans un autre fichier
            FileUtil.string2File("Order-updated.xml",
               rsx.getXmlText());
            } catch (Exception e) {
            System.out.println("Exception:");
            e.printStackTrace();
            }
        }
 }
```

La méthode FileUtil.string2File( ) stocke le document ResultSet mis à jour dans le *Order-updated.xml*. Le ResultSetMetaData du document mis à jour n'est pas modifié. La section ResultSetData mise à jour du document se présente comme suit, avec de nouvelles valeurs pour le deuxième article.

```
<ResultSetData>
     <Row>
       <Column name="Date">1999-07-04 00:00:00.0</Column>
       <Column name="CustomerId">123</Column>
       <Column name="CustomerName">Acme Alpha</Column>
       <Column name="ItemId">987</Column>
       <Column name="ItemName">Gadget</Column>
       <Column name="Quantity">5</Column>
       <Column name="unit">1</Column>
     </Row>
     <Row>
       <Column name="Date">1999-07-04 00:00:00.0</Column>
       <Column name="CustomerId">123</Column>
       <Column name="CustomerName">Acme Alpha</Column>
       <Column name="ItemId">777</Column>
       <Column name="ItemName">Flange</Column>
       <Column name="Quantity">3</Column>
       <Column name="unit">13</Column>
     </Row>
     <Row>
       <Column name="Date">1999-07-04 00:00:00.0</Column>
       <Column name="CustomerId">123</Column>
       <Column name="CustomerName">Acme Alpha</Column>
       <Column name="ItemId">579</Column>
```

```
      <Column name="ItemName">Fermoir Type 3</Column>
      <Column name="Quantity">1</Column>
      <Column name="unit">1</Column>
    </Row>
  </ResultSetData>
</ResultSet>
```

Script serveur

Le script SQL de la section "Stockage d'un document XML ResultSet dans une colonne SQL", page 188 vous a permis de stocker des documents XML ResultSet complets dans la colonne rs_doc de la table resultset_docs. Les commandes SQL suivantes, exécutées dans un environnement serveur, référencent et mettent à jour les colonnes contenues dans ces documents.

Vous pouvez sélectionner des colonnes en spécifiant leur nom ou leur numéro :

- Sélection des colonnes de la ligne 1 par spécification de leur nom :

```
select rs_doc>>getColumn(1, "Date"),
 rs_doc>>getColumn(1, "CustomerId"),
 rs_doc>>getColumn(1, "CustomerName"),
 rs_doc>>getColumn(1, "ItemId"),
 rs_doc>>getColumn(1, "ItemName"),
 rs_doc>>getColumn(1, "Quantity"),
 rs_doc>>getColumn(1, "unit")
  from resultset_docs
```

- Sélection des colonnes de la ligne 1 par spécification de leur numéro :

```
select rs_doc>>getColumn(1, 0),
 rs_doc>>getColumn(1, 1),
 rs_doc>>getColumn(1, 2),
 rs_doc>>getColumn(1, 3),
 rs_doc>>getColumn(1, 4),
 rs_doc>>getColumn(1, 5),
 rs_doc>>getColumn(1, 6)
 from resultset_docs
```

Spécifiez des colonnes et des lignes inexistantes. Ces références renvoient des valeurs NULL.

```
Select rs_doc>>getcolumn(1, "itemid"),
    rs_doc>>getcolumn(1, "xxx"),
    rs_doc>>getcolumn(1, "Quantity"),
    rs_doc>>getcolumn(99,  "unit"),
    rs_doc>>getColumn(1,  876)
 from resultset_docs
```

Mettez à jour les colonnes du document ResultSet stocké :

```
update resultset_docs
set rs_doc = rs_doc>>setColumn(1, "ItemName",
"Wrench")
where id= "1"
update resultset_docs
 set rs_doc = rs_doc>>setColumn(1, "ItemId", "967")
 where id="1"
update resultset_docs
 set rs_doc = rs_doc>>setColumn(1,  "unit", "6")
 where id="1"
select rs_doc>>getColumn(1, "ItemName"),
    rs_doc>>getColumn(1, "ItemId"),
    rs_doc>>getColumn(1,  "unit")
 from resultset_docs
 where id="1"
```

Comparaisons quantifiées dans les documents ResultSet stockés

ResultSetXml contient deux méthodes, allString( ) et someString( ), qui permettent d'effectuer des recherches quantifiées dans les colonnes d'un document ResultSetXml. Pour illustrer ces deux méthodes, créez d'abord quelques lignes exemples dans la table order_results.

La table order_results a été initialisée avec une ligne, dont la colonne id est égale à "1" et dont la colonne rs_doc contient la commande initiale utilisée dans tous les exemples.

Les instructions suivantes copient cette ligne deux fois, en affectant les valeurs "2" et "3" à la colonne id des nouvelles lignes. La table order_results possède désormais trois lignes, avec des colonnes id définies à "1", "2" et "3", ainsi que la commande initiale.

```
insert into resultset_docs(id, rs_doc)
select "2", rs_doc
 from resultset_docs where id="1"
insert into resultset_docs (id, rs_doc) select "3", rs_doc
 from resultset_docs where id="1"
```

Les instructions suivantes modifient la ligne dont la colonne id est définie à "1" pour que le paramètre ItemId des trois articles soit égal à "100", "110" et "120" :

```
update resultset_docs
 set rs_doc = rs_doc>>setColumn(0, "ItemId", "100")
 where id="1"
update resultset_docs
 set rs_doc = rs_doc>>setColumn(1, "ItemId", "110")
 where id="1"
update resultset_docs
 set rs_doc = rs_doc>>setColumn(2, "ItemId", "120")
 where id="1"
```

L'instruction update suivante modifie la ligne dont la colonne id est définie à "3" pour que le paramètre ItemId de son deuxième article (à partir de 0) soit égal à "999" :

```
update resultset_docs
 set rs_doc = rs_doc>>setColumn(2, "ItemId", "999")
 where id="3"
```

L'instruction select suivante affiche la colonne id et les trois valeurs du paramètre ItemId pour chaque ligne :

```
select id, rs_doc>>getColumn(0, "ItemId"),
    rs_doc>>getColumn(1, "ItemId"),
    rs_doc>>getColumn(2, "ItemId")
 from resultset_docs
```

Les résultats de l'instruction select sont :

```
1     100      110        120
2     987      654        579
3     987      654        999
```

Notez que :

- la ligne dont la colonne id est définie à "2" contient les données de la commande initiale ;
- la ligne dont la colonne id est définie à "1" a été modifiée pour que *toutes* les valeurs du paramètre ItemId dans cette ligne soient inférieures à "200" ;
- la ligne dont la colonne id est définie à "3" a été modifiée pour que *certaines* valeurs du paramètre ItemId dans cette ligne soient supérieures ou égales à "9999".

L'exemple suivant illustre ces quantifications avec les méthodes allString( ) et someString( ) :

```
select id, rs_doc>>allString(3, "<", "200") as "ALL test"
 from resultset_docs
select id, rs_doc>>someString(3, ">=", "999") as "SOME test"
 from resultset_docs
select id as "id for ALL test" from resultset_docs
 where rs_doc>>allString(3, "<", "200")>>booleanValue() = 1
select id as "id for SOME test" from resultset_docs
 where rs_doc>>someString(3, ">=", "999")>>booleanValue() = 1
```

Les deux premières instructions montrent le quantificateur dans la liste select et donnent les résultats suivants :

ID	Test "all"	Test "certaines"
1	vrai	faux
2	faux	faux
3	faux	vrai

Les deux dernières instructions montrent le quantificateur de la clause where et donnent les résultats suivants :

- ID pour le test "toutes" = "3"
- ID pour le test "certaines" = "1"

Dans les exemples utilisant la méthode du quantificateur dans la clause where, il convient de noter les points suivants :

- Les exemples de clause where comparent les résultats de la méthode avec la valeur entière 1. SQL ne supporte pas le type de données Booléen comme une valeur de fonction, mais traite les valeurs booléennes comme les valeurs entières 1 et 0, pour vrai et faux.
- Les exemples de clause where utilisent la méthode booleanValue( ). Les méthodes allString( ) et someString( ) renvoient le type java.lang.Boolean, qui n'est pas compatible avec le type SQL integer. La méthode Java booleanValue( ) renvoie la valeur booléenne simple à partir de l'objet booléen qui est compatible avec le type SQL integer. Ce comportement résulte de la fusion des systèmes des types de données SQL et Java.

Les méthodes de quantificateur renvoient java.lang.Boolean au lieu du type Java boolean, de sorte qu'elles peuvent renvoyer la valeur NULL lorsque la colonne sort de l'intervalle admis, ce qui est cohérent avec le traitement SQL lorsqu'une valeur sort de l'intervalle admis.

Les instructions suivantes montrent des références de quantificateur spécifiant la colonne 33, qui n'existe pas dans les données :

```
select id, rs_doc>>allString(33, "<", "200") as "ALL test"
 from resultset_docs
select id as "id for ALL test" from resultset_docs
 where rs_doc>>allString(33, "<", "200")>>booleanValue() = 1
```

ID	Test "all"
1	NULL
2	NULL
3	NULL

ID pour le test "all " = (vide).

CHAPITRE 10

Débogage de Java dans la base de données

Ce chapitre décrit le débogueur Java de Sybase et explique comment l'utiliser lors du développement de Java dans Adaptive Server.

Nom	Page
Introduction au débogage de Java	197
Utilisation du débogueur	198
Didacticiel de débogage	205

Introduction au débogage de Java

Le débogueur Java de Sybase permet de tester les classes Java et de résoudre les problèmes qui risquent de se poser.

Fonctionnement du débogueur

Le débogueur Java de Sybase est une application Java exécutée sur une machine cliente. Il se connecte à la base de données à l'aide du pilote JDBC de Sybase jConnect.

Le débogueur débogue les classes exécutées dans la base de données. Vous pouvez passer en revue le code source Java des fichiers à condition qu'il réside sur le disque de votre machine cliente. (N'oubliez pas que les classes compilées sont installées dans la base de données mais que le code source ne l'est pas.)

Conditions requises pour utiliser le débogueur Java

Pour utiliser le débogueur Java, vous avez besoin :

- d'un environnement d'exécution Java tel que Sun Microsystems Java Runtime Environment ou le JDK Sun Microsystems complet sur votre machine ;
- du code source de votre application sur la machine cliente.

Applications possibles du débogueur

Le débogueur Java de Sybase permet de :

- suivre l'exécution : vous parcourez pas à pas le code source d'une classe exécutée dans la base de données. Vous pouvez également parcourir vers l'avant et vers l'arrière la pile des fonctions appelées ;
- définir des points d'arrêt : vous exécutez le code jusqu'à ce que vous rencontriez un point d'arrêt et vous interrompez l'exécution à ce niveau du code ;
- définir des conditions d'arrêt : les points d'arrêt incluent des lignes de code, mais vous pouvez également définir les conditions d'arrêt du code. Par exemple, vous pouvez arrêter le code sur une ligne lorsqu'elle est exécutée pour la dixième fois ou seulement si une variable a une valeur donnée. Vous pouvez également arrêter le code chaque fois qu'une certaine exception est générée par l'application Java ;
- parcourir le contenu des classes : vous pouvez parcourir les classes installées dans la base de données qu'utilise le serveur ;
- vérifier et définir des variables : vous pouvez vérifier si les valeurs des variables sont modifiées après l'arrêt de l'exécution à un point d'arrêt ;
- vérifier des expressions et arrêter l'exécution : vous pouvez vérifier les valeurs de nombreuses expressions.

Utilisation du débogueur

Cette section explique comment utiliser le débogueur Java. La section suivante fournit un didacticiel simple.

Démarrage du débogueur et connexion à la base de données

Le débogueur est le fichier JAR *Debug.jar* installé dans le répertoire d'installation Adaptive Server sous *$SYBASE/$SYBASE_ASE/debugger*. S'il n'y figure pas déjà, ajoutez ce fichier comme le premier élément de votre variable d'environnement CLASSPATH.

Debug.jar contient de nombreuses classes. Pour démarrer le débogueur, appelez la classe sybase.vm.Debug, qui comprend une méthode main( ). Il existe trois façons de démarrer le débogueur :

- Exécutez le script *jdebug* situé sous *$SYBASE/$SYBASE_ASE/debugger*.

  La section "Didacticiel de débogage", page 205 fournit une session de débogage exemple utilisant le script *jdebug*.

- Sur la ligne de commande, entrez :

  ```
  java sybase.vm.Debug
  ```

  Dans la fenêtre Connect, entrez un URL, un nom de connexion utilisateur et un mot de passe pour établir une connexion à la base de données.

- A partir de Sybase Central :

  a Démarrez Sybase Central et ouvrez le dossier Utilities, sous Adaptive Server Enterprise.

  b Cliquez deux fois sur l'icône du débogueur Java dans le volet de droite.

  c Dans la fenêtre Connect, entrez un URL, un nom de connexion utilisateur et un mot de passe pour établir une connexion à la base de données.

Compilation des classes à déboguer

Les compilateurs Java, tels que javac de Sun Microsystems, peuvent compiler des classes Java avec différents niveaux d'optimisation. Vous pouvez compiler du code Java de telle manière que les informations utilisées par les débogueurs soient conservées dans les fichiers de classe compilés.

Si vous compilez votre code source sans utiliser d'option de débogage, vous pouvez passer en revue le code et utiliser des points d'arrêt. Cependant, vous ne pouvez pas vérifier les valeurs des variables locales.

Pour compiler des classes à déboguer à l'aide du compilateur javac, utilisez l'option -g :

```
javac -g ClassName.java
```

Réquisition d'une machine virtuelle Java

Lorsque vous vous connectez à une base de données à partir du débogueur, la fenêtre Connection présente toutes les machines virtuelles Java actives sous le nom de connexion utilisateur. S'il n'y en a aucune, le débogueur passe en *mode d'attente*. Le mode d'attente fonctionne de la manière suivante :

- Chaque fois qu'une nouvelle machine virtuelle Java est démarrée, elle apparaît dans la liste.
- Vous pouvez déboguer la nouvelle machine virtuelle Java ou attendre qu'une autre apparaisse.
- Une fois que, en traversant la liste, vous avez dépassé une machine virtuelle Java, vous ne pouvez plus revenir dessus et choisir de la déboguer. Si toutefois vous souhaitez ultérieurement vous y attacher, vous devez vous déconnecter de la base de données puis vous reconnecter. La machine virtuelle Java apparaît alors comme active et vous pouvez vous y attacher.

La fenêtre Source

La fenêtre Source :

- affiche du code source Java, avec des numéros de ligne et des indicateurs de point d'arrêt (un astérisque dans la colonne de gauche) ;
- affiche l'état d'exécution dans la zone d'état en bas de la fenêtre ;
- donne accès à d'autres fenêtres du débogueur à partir du menu.

Fenêtres de débogueur

Le débogueur peut afficher les fenêtres suivantes :

- Fenêtre Breakpoints : affiche la liste des points d'arrêt courants.
- Fenêtre Calls : affiche la pile d'appels courante.
- Fenêtre Classes : affiche la liste des classes actuellement chargées dans la machine virtuelle Java. De plus, cette fenêtre affiche la liste des méthodes et la liste des variables statiques de la classe actuellement sélectionnée. Cette fenêtre permet de définir des points d'arrêt au niveau de l'accès à une méthode ou lorsqu'une variable statique est écrite.

- Fenêtre Connection : s'affiche au démarrage du débogueur. Vous pouvez également l'afficher si vous souhaitez vous déconnecter de la base de données.
- Fenêtre Exceptions : permet de définir une exception particulière sur laquelle l'exécution s'interrompra ou d'arrêter l'exécution sur toutes les exceptions.
- Fenêtre Inspection : affiche les variables statiques courantes et permet de les modifier. Vous pouvez également vérifier la valeur des expressions Java, notamment :
  - les variables locales ;
  - les variables statiques ;
  - les expressions utilisant l'opérateur point ;
  - les expressions utilisant les sous-scripts [] ;
  - les expressions utilisant les parenthèses, les opérateurs arithmétiques ou les opérateurs logiques.

    Par exemple, les expressions suivantes sont utilisables :

    ```
    x[i].field
    q + 1
    i == 7
    (i + 1)*3
    ```
- Fenêtre Locals : affiche les variables locales courantes et permet de les modifier.
- Fenêtre Status : affiche des messages décrivant l'état d'exécution de la machine virtuelle Java.

Options

Toutes les options utilisables pour passer en revue le code source apparaissent dans le menu Run. Les options suivantes sont disponibles :

Fonction	Raccourci clavier	Description
Run	F5	Poursuit l'exécution jusqu'au point d'arrêt suivant, jusqu'à la sélection de l'option Stop ou jusqu'à la fin.

Fonction	Raccourci clavier	Description
Step Over	F7 ou espace	Passe à la ligne suivante de la méthode courante. Si la ligne invoque une autre méthode, le débogueur ne rentre pas dans celle-ci. Passez outre également les points d'arrêt des méthodes ignorées.
Step Into	F8 ou i	Passe à la ligne de code suivante. Si la ligne invoque une autre méthode, le débogueur rentre dans cette méthode.
Step Out	F11	Continue l'exécution de la méthode courante et s'interrompt à la ligne suivante de la méthode appelante.
Stop		Interrompt l'exécution.
Run to Selected	F6	S'arrête après l'exécution de la ligne actuellement sélectionnée.
Home	F4	Sélectionne la ligne sur laquelle l'exécution est arrêtée.

Définition des points d'arrêt

Lorsque vous définissez un point d'arrêt dans le débogueur, la machine virtuelle Java arrête l'exécution sur ce point. Une fois l'exécution arrêtée, vous pouvez vérifier et modifier les valeurs des variables et d'autres expressions pour mieux comprendre l'état du programme. Puis vous pouvez suivre l'exécution pas à pas pour identifier les problèmes.

La définition de points d'arrêt aux endroits adéquats est essentielle pour localiser les problèmes d'exécution.

Le débogueur Java permet de définir des points d'arrêt non seulement sur une ligne de code, mais également en utilisant de nombreuses autres conditions. Cette section explique comment définir des points d'arrêt en utilisant des conditions.

Arrêt sur un numéro de ligne

Lorsque vous définissez un point d'arrêt sur une ligne de code, l'exécution s'arrête chaque fois que cette ligne de code est exécutée.

Pour définir un point d'arrêt sur une ligne de code donnée, procédez comme suit :

- Dans la fenêtre Source, sélectionnez la ligne et appuyez sur la touche F9.

  Vous pouvez aussi cliquer deux fois sur la ligne.

Lorsqu'un point d'arrêt est défini sur un numéro de ligne, il est identifié dans la fenêtre Source à l'aide d'un astérisque dans la colonne de gauche. Si la fenêtre Breakpoints est ouverte, la méthode et le numéro de ligne sont affichés dans la liste des points d'arrêt.

Vous pouvez activer et désactiver le point d'arrêt en cliquant deux fois ou en appuyant sur la touche F9.

Arrêt sur une méthode de classe

Lorsque vous définissez un point d'arrêt sur une méthode, ce point est défini sur la première ligne de code de la méthode contenant une instruction exécutable.

Pour définir un point d'arrêt sur une méthode statique, procédez comme suit :

1 Dans la fenêtre Source, sélectionnez Break→ New. La fenêtre Break At s'affiche.

2 Entrez le nom de la méthode sur laquelle vous souhaitez arrêter l'exécution. Par exemple :

```
JDBCExamples.selecter
```

arrête l'exécution à chaque accès à la méthode JDBCExamples.selecter( ).

Lorsqu'un point d'arrêt est défini sur une méthode, il est identifié dans la fenêtre Source par un astérisque dans la colonne de gauche de la ligne correspondant au point d'arrêt. Si la fenêtre Breakpoints est ouverte, la méthode s'affiche dans la liste des points d'arrêt.

Utilisation de compteurs avec les points d'arrêt

Si vous définissez un point d'arrêt sur une ligne qui se trouve dans une boucle ou dans une méthode fréquemment appelée, il se peut que la ligne soit exécutée plusieurs fois avant que la condition qui vous intéresse se présente. Le débogueur permet d'associer un compteur à un point d'arrêt afin que l'exécution ne s'arrête que lorsque la ligne est exécutée un certain nombre de fois.

Pour associer un compteur à un point d'arrêt, procédez comme suit :

1 Dans la fenêtre Source, sélectionnez Break→Display. La fenêtre Breakpoints s'affiche.

2 Dans la fenêtre Breakpoints, cliquez sur un point d'arrêt pour le sélectionner.

3 Sélectionnez Break→Count. Une fenêtre s'affiche pour vous permettre d'entrer le nombre d'itérations. Entrez un entier. L'exécution s'arrêtera lorsque la ligne aura été exécutée le nombre de fois spécifié.

Utilisation de conditions avec les points d'arrêt

Le débogueur permet d'associer une condition à un point d'arrêt de manière que l'exécution ne s'arrête que lorsque la ligne est exécutée et que la condition est remplie.

Pour associer une condition à un point d'arrêt, procédez comme suit :

1 Dans la fenêtre Source, sélectionnez Break→Display. La fenêtre Breakpoints s'affiche.

2 Dans la fenêtre Breakpoints, cliquez sur un point d'arrêt pour le sélectionner.

3 Sélectionnez Break→Condition. Une fenêtre s'affiche pour vous permettre d'entrer une expression. L'exécution s'arrêtera lorsque la condition sera remplie.

   Les expressions utilisées dans ce cas sont les mêmes que celles qui sont utilisables dans la fenêtre Inspection, notamment :

   - les variables locales ;
   - les variables statiques ;
   - les expressions utilisant l'opérateur point ;
   - les expressions utilisant les sous-scripts [] ;
   - les expressions utilisant les parenthèses, les opérateurs arithmétiques ou les opérateurs logiques.

Arrêt lorsque l'exécution n'est pas interrompue

A une seule exception près, les points d'arrêt ne peuvent être définis que lorsque l'exécution du programme est interrompue. Si vous supprimez tous les points d'arrêt et que vous exécutez le programme en cours de débogage jusqu'à son terme, vous ne pouvez plus définir de point d'arrêt sur une ligne ou au début d'une méthode. En outre, si un programme est exécuté en boucle, l'exécution se poursuit sans interruption.

Pour déboguer votre programme dans l'une de ces conditions, sélectionnez Run→Stop dans la fenêtre Source. Ceci a pour effet d'arrêter l'exécution à la prochaine ligne de code Java exécutée. Vous pouvez ensuite définir des points d'arrêt sur d'autres points du code.

Déconnexion de la base de données

Une fois que le programme a été exécuté jusqu'à son terme ou à tout moment du débogage, vous pouvez vous déconnecter de la base de données à partir de la fenêtre Connect. Ensuite, fermez la fenêtre Source et reconnectez-vous à la base de données une fois le débogage terminé.

Didacticiel de débogage

Cette section décrit pas à pas une session de débogage simple.

Notes préliminaires

Le code source de la classe utilisée dans ce didacticiel est disponible sous *$SYBASE/$SYBASE_ASE/sample/JavaSql/manual-examples/JDBCExamples.java*.

Avant d'exécuter le débogueur, compilez le code source à l'aide de la commande javac, avec l'option -g.

Pour obtenir des informations détaillées sur la compilation et l'installation des classes Java dans la base de données, reportez-vous à la section "Création de classes Java et de fichiers JAR", page 16.

Démarrage du débogueur Java et connexion à la base de données

Vous pouvez démarrer le débogueur et vous connecter à la base de données à l'aide d'un script, d'options de ligne de commande ou de Sybase Central. Dans ce didacticiel, nous utiliserons *jdebug*. Vous pouvez utiliser n'importe quelle base de données.

Procédez comme suit :

1 Démarrez Adaptive Server.

2 Si aucune requête Java n'a encore été exécutée sur votre serveur, exécutez-en une pour initialiser le sous-système Java et démarrer une machine virtuelle Java.

3 Exécutez le script *$SYBASE/$SYBASE_ASE/debugger/jdebug*. *jdebug* vous invite à entrer :

   a le nom de la machine d'Adaptive Server ;

   b le numéro de port de la base de données ;

   c votre nom de connexion ;

   d votre mot de passe ;

   e un autre chemin vers *Debug.jar* s'il ne figure pas déjà dans votre CLASSPATH.

Une fois la connexion établie, la fenêtre du débogueur affiche la liste des machines virtuelles Java disponibles ou le message "Waiting for a VM".

Réquisition d'une machine virtuelle Java

Pour vous connecter à une machine virtuelle Java à partir de votre session utilisateur, procédez comme suit :

1 Une fois que le débogueur a démarré, connectez-vous à la base de données exemple à partir de isql en tant qu'utilisateur sa :

```
$SYBASE/bin/isql -Usa -P
```

Remarque Vous ne pouvez pas démarrer l'exécution Java à partir du débogueur. Pour démarrer une machine virtuelle Java, exécutez une opération Java à partir d'une autre connexion avec le même nom d'utilisateur.

2 Exécutez le code Java à l'aide des instructions suivantes :

```
select JDBCExamples.serverMain('createtable')
select JDBCExamples.serverMain('insert')
select JDBCExamples.serverMain('select')
```

La machine virtuelle Java de Sybase démarre pour extraire les objets Java de la table. Le débogueur arrête immédiatement l'exécution du code Java.

La fenêtre Connection du débogueur affiche les machines virtuelles Java appartenant à l'utilisateur dans le format suivant :

```
VM#: "nom_connexion, spid:spid#"
```

3 Dans la fenêtre Connection du débogueur, sélectionnez une machine virtuelle Java, puis cliquez sur Attach to VM. Le débogueur s'attache à la machine virtuelle Java et la fenêtre Source s'affiche. La fenêtre Connection se ferme.

Ensuite, faites apparaître le code source de la méthode dans la fenêtre Source. Le code source est disponible sur le disque.

Chargement du code source dans le débogueur

Le débogueur recherche les fichiers de code source. Vous devez donc permettre au débogueur d'accéder au sous-répertoire *$SYBASE/$SYBASE_ASE/sample/JavaSql/manual-examples/* afin qu'il puisse rechercher le code source de la classe actuellement exécutée dans la base de données.

Pour ajouter un emplacement de code source sur le débogueur, procédez comme suit :

1 Dans la fenêtre Source, sélectionnez File→Source Path. La fenêtre Source Path s'affiche.

2 Dans la fenêtre Source Path, sélectionnez Path→Add. Dans la zone de texte, entrez l'emplacement suivant :

```
$SYBASE/$SYBASE_ASE/sample/JavaSql/
       manual-examples/
```

Le code source de la classe JDBCExamples s'affiche dans la fenêtre et la première ligne de la méthode Query serverMain( ) est mise en surbrillance. Le débogueur Java a arrêté l'exécution du code sur ce point.

Vous pouvez maintenant fermer la fenêtre Source Path.

Passage en revue du code source

Il existe plusieurs façons de parcourir pas à pas le code source dans le débogueur Java. Les différentes possibilités offertes par la méthode serverMain( ) sont présentées ci-après.

Lorsque l'exécution est suspendue sur une ligne jusqu'à ce que des instructions soient fournies, on dit que l'exécution est **interrompue** sur la ligne. La ligne est un **point d'arrêt**. L'exécution pas à pas du code requiert la définition de points d'arrêt explicites ou implicites dans le code et l'exécution du code jusqu'à ces points d'arrêt.

Dans la section précédente, le débogueur a dû arrêter l'exécution de JDBCExamples.serverMain( ) à la première instruction :

Exemples

Voici quelques exercices pratiques :

1 Rentrer dans une fonction : appuyez sur la touche F7 pour passer à la ligne suivante de la méthode courante.

2 Appuyez sur la touche F8 pour rentrer dans la fonction doAction( ), à la ligne 99.

3 Poursuivez l'exécution jusqu'à une ligne sélectionnée. Vous êtes désormais entré dans la fonction doAction( ). Cliquez sur la ligne 155 et appuyez sur la touche F6 pour exécuter cette ligne et cet arrêt :

```
String workString = "Action(" + action + ")";
```

4 Créez un point d'arrêt et poursuivez l'exécution jusqu'à ce point : sélectionnez la ligne 179 et appuyez sur la touche F9 pour définir un point d'arrêt sur cette ligne lorsque vous exécutez isql `select JDBCExamples.serverMain('select')` :

```
workString + = selecter(con);
```

Appuyez sur la touche F5 pour poursuivre l'exécution jusqu'à cette ligne.

5 Test : essayez plusieurs méthodes d'exécution pas à pas du code. Terminez en appuyant sur la touche F5 pour mettre fin à l'exécution.

Une fois que vous avez mis fin à l'exécution, la fenêtre Interactive SQL Data s'affiche :

```
Action(select) - Row with id = 1: name(Joe Smith)
```

Vérification et modification des variables

Vous pouvez lire les valeurs des variables locales (déclarées dans une méthode) et des variables statiques de classe dans le débogueur.

Vérification des variables locales

Vous pouvez lire les valeurs des variables locales d'une méthode lorsque vous parcourez pas à pas le code pour mieux comprendre le fonctionnement du programme.

Pour lire et modifier la valeur d'une variable, procédez comme suit :

1 A partir de la fenêtre Breakpoint, définissez un point d'arrêt sur la première ligne de la méthode selecter( ). Cette ligne est la suivante :

```
String sql = "select name, home from xmp where
       id=?";
```

2 Dans Interactive SQL, entrez de nouveau l'instruction suivante pour exécuter la méthode :

```
select JDBCExamples.serverMain('select')
```

La requête ne s'exécute que jusqu'au point d'arrêt.

3 Appuyez sur la touche F7 pour passer à la ligne suivante. La variable *sql* est maintenant déclarée et initialisée.

4 Dans la fenêtre Source, sélectionnez Window→Locals. La fenêtre Locals s'affiche.

La fenêtre Locals montre qu'il existe plusieurs variables locales. La variable *sql* est définie à zéro. Les autres variables sont répertoriées comme hors de portée, ce qui signifie qu'elles ne sont pas encore initialisées.

Vous devez ajouter les variables à la liste dans la fenêtre Inspect.

5 Dans la fenêtre Source, appuyez plusieurs fois sur la touche F7 pour parcourir pas à pas le code. Ceci a pour effet de faire apparaître les valeurs des variables dans la fenêtre Locals.

Si une variable locale n'est pas un entier simple ni une autre quantité, dès qu'elle est définie, le signe + apparaît en regard. Ceci signifie que la variable locale comporte des champs avec des valeurs. Vous pouvez développer une variable locale en cliquant deux fois sur le signe + ou en positionnant le curseur sur la ligne et en appuyant sur la touche Enter.

6 Terminez l'exécution de la requête pour finir cet exercice.

Modification des variables locales

Vous pouvez également modifier les valeurs des variables à partir de la fenêtre Locals.

Pour modifier une variable locale, procédez comme suit :

1 Dans la fenêtre Source, définissez un point d'arrêt sur la ligne suivante de la méthode selecter( ) de la classe serverMain :

```
String sql = "select name, home from xmp where
       id=?";
```

2 Poursuivez l'exécution après cette ligne.

3 Ouvrez la fenêtre Locals. Sélectionnez la variable *id* et sélectionnez Local→Modify. Vous pouvez également positionner le curseur sur la ligne et appuyer sur la touche Enter.

4 Entrez la valeur 2 dans la zone de texte et cliquez sur OK pour confirmer la nouvelle valeur. La variable *id* est définie à 2 dans la fenêtre Locals.

5 Dans la fenêtre Source, appuyez sur la touche F5 pour mettre fin à l'exécution de la requête. Dans la fenêtre Interactive SQL Data, un message d'erreur s'affiche pour indiquer qu'aucune ligne n'a été trouvée.

Vérification des variables statiques

Vous pouvez également lire les valeurs des variables de classe (variables statiques).

Pour lire une variable statique, procédez comme suit :

1 Dans la fenêtre Source, sélectionnez Window→Classes. La fenêtre Classes s'affiche.

2 Sélectionnez une classe dans la zone de gauche. Les méthodes et les variables statiques de la classe sont affichées dans les zones de droite.

3 Sélectionnez Static→Inspect. La fenêtre Inspect s'affiche. Elle répertorie les variables disponibles pour la lecture.

CHAPITRE 11

Accès au réseau avec java.net

Adaptive Server 12.5 supporte le package java.net qui permet de créer des applications réseau et d'accéder à différents types de serveurs externes.

Rubrique	Page
Présentation	211
Classes java.net	212
Configuration de java.net	212
Exemple d'utilisation	213
Remarques à l'attention de l'utilisateur	219
Complément d'information	219

Adaptive Server java.net est compatible avec l'API Java 1.2.

Présentation

L'intégration de java.net dans Adaptive Server vous permet de créer des applications réseau Java client au sein du serveur. Vous pouvez créer dans Adaptive Server une application réseau Java client connectable à tous les serveurs qui configure Adaptive Server pour fonctionner en tant que client vis-à-vis des serveurs externes. Reportez-vous à la section "Exemple d'utilisation", page 213.

Vous pouvez utiliser java.net aux diverses fins suivantes :

- télécharger des documents depuis n'importe quelle adresse Internet ;
- envoyer des messages électroniques à partir du serveur ;
- se connecter au serveur externe pour enregistrer un document et exécuter des fonctions au niveau des fichiers : enregistrement d'un document, modification d'un document etc. ;
- accéder aux documents via XML.

Classes java.net

Le Tableau 11.1 répertorie les classes java.net supportées par Sybase.

Tableau 11-1 : Classes java.net supportées

Classe	Supportée	Conditions spéciales
InetAddress	Oui	Aucune
Socket	Oui	Ne supporte pas le constructeur déconseillé "Socket (chaîne hôte, port int, flux, booléen)" lorsque stream = false.
URL	Oui	Absence d'URL de fichier
HttpURLConnection	Oui	Aucune
URLConnection	Oui	Absence d'URL de fichier
URLDecoder	Oui	Aucune
URLEncoder	Oui	Aucune
DatagramPacket	Non	
DatagramSocket	Non	
MulticastSocket	Non	
ServerSocket	Non	

Vous pouvez utiliser n'importe quelle classe supportée dans java.net pour écrire des applications clientes Adaptive Server.

Configuration de java.net

La procédure ci-après permet d'activer java.net.

❖ **Activation de java.net**

1 Activez la machine virtuelle Java.

```
sp_configure "enable java", 1
```

2 Spécifiez le nombre de sockets à ouvrir (0 étant la valeur par défaut). Le paramètre de configuration du nombre de sockets étant dynamique, il est inutile de redémarrer Adaptive Server à la suite de sa modification. Par exemple, pour ouvrir 10 sockets, tapez ce qui suit :

```
sp_configure "number of java sockets", 10
```

3 Adaptez le volume de mémoire allouée à la machine virtuelle Java. Si vous envisagez la transmission en continu de textes volumineux depuis et vers le serveur, il convient d'augmenter en conséquence la quantité de mémoire mis à la disposition de la machine virtuelle Java. Les paramètres entrant en ligne de compte sont les suivants :

- size of global fixed heap
- size of process object heap
- size of shared class heap

Pour plus d'informations sur ces paramètres, reportez-vous au Chapitre 5, "Paramètres de configuration" du Sybase *Guide d'administration système*.

Exemple d'utilisation

Cette section fournit des exemples d'utilisation des classes socket et des classes URL. Vous pouvez :

- accéder à un document externe avec XQL, à l'aide de la classe URL ;
- enregistrer du texte en dehors d'Adaptive Server ;
- utiliser l'URL de classe MailTo pour envoyer un document par messagerie.

Utilisation des classes socket

Les classes socket permettent d'effectuer des transferts réseau plus sophistiqués que les classes URL. La classe Socket permet de se connecter à un port précis d'un quelconque hôte réseau spécifié et d'utiliser les classes InputStream et OutputStream pour lire et écrire les données.

Enregistrement de texte en dehors d'Adaptive Server

Cet exemple décrit comment configurer une application cliente dans Adaptive Server. Adaptive Server version 12.5 ne supportant pas l'accès direct à un fichier, cet exemple fournit une solution à cette restriction.

Vous pouvez écrire votre propre serveur externe, exécutant des opérations de fichier et vous y connecter depuis Adaptive Server en utilisant un socket créé à partir d'une classe Socket.

En principe, le client se connecte au serveur et transmet en continu le texte, tandis que le serveur reçoit le flux et le transmet à un fichier.

Cet exemple décrit comment installer une application Java dans Adaptive Server, à l'aide de java.net. Cette application fait office de client pour un serveur externe.

❖ **Le processus client se déroule comme suit :**

1. Réception d'un flux InputStream.
2. Création d'un socket à l'aide de la classe Socket pour se connecter au serveur.
3. Création d'un flux OutputStream sur le socket.
4. Lecture du flux InputStream et écriture de celui-ci dans le flux OutputStream :

```
public static void writeOut(InputStream fin) throws
       Exception{
Socket socket = new Socket("localhost", 1718);
OutputStream fout = new
BufferedOutputStream(socket.getOutputStream());byte []
       buffer = new byte [10];
int bytes_read;
while (bytes_read = fin.read(buffer)) ! = -1{
       fout.write(buffer, 0, bytes_read);
}
fout.close();
}
```

Compilation de ce programme.

❖ **Le processus serveur se déroule comme suit :**

1. Création d'un socket de serveur à l'aide de la classe SocketServer pour une connexion d'écoute sur un port.
2. Utilisation du socket de serveur pour l'établissement d'une connexion.
3. Réception d'un flux InputStream.

4 Lecture du flux InputStream et écriture de celui-ci dans un flux FileOutputStream.

Remarque Dans cet exemple, le serveur n'utilise pas de threads ; par conséquent, il ne peut recevoir une connexion qu'à partir d'un seul client à la fois.

```
public class FileServer {
public static void main (string[] args) throws
IOException{
 Socket client = accept (1718);
 try{
  InputStream in = client.getInputStream ();
  FileOutputStream fout = new
    FileOutputStream("chastity.txt");
    byte[] buffer = new byte [10];
    int bytes_read;
    while (bytes_read = in.read(buffer))!= -1){
     fout.write(buffer, 0, bytes_read);
 }
    fout.close();
 }finally {
  client.close ();
 }
}
static Socket accept (int port) throws
       IOException{
    System.out.prinln
       ("Starting on port" + port);
    ServerSocket server = new
       ServerSocket (port);
    System.ou.println ("Waiting");
    Socket client = server.accept ();
    System.out.println ("Accepted from" +
       client.getInetAddress ());
    server.close ();
    return client;
    }
}
```

Compilation de ce programme.

Pour utiliser cette combinaison client/serveur, installez le client dans Adaptive Server et démarrez le serveur externe comme suit :

```
witness% java FileServer &
[2] 28980
witness% Starting on port 1718
Waiting
```

Appelez le client à partir d'Adaptive Server.

```
create table t(c1 text)
go
insert values into t1 ("samplestring")
go

select TestStream2File.writeOut(c1) from t
go
```

Utilisation de la classe URL

Vous pouvez utiliser la classe URL pour :

- envoyer un message électronique ;
- télécharger un document HTTP depuis un serveur Web. Il peut s'agir d'un document statique ou construit dynamiquement par le serveur Web ;
- accéder à un document externe avec XQL.

Envoi d'un document par messagerie à l'aide de l'URL de classe MailTo

L'envoi d'un document par messagerie est un bon exemple d'utilisation de la classe URL. Avant toute chose, veillez à ce que votre client soit connecté à un programme de messagerie électronique, tel que sendmail.

1 Créez un objet URL.

2 Définissez un objet URLConnection.

3 Créez un objet OutputStream à partir de l'objet URL.

4 Rédigez un message électronique. Par exemple :

```
public static void sendIt() throws Execption{
System.getProperty("mail.host",
"salsa.sybase.com");
URL url = new URL(mailto:"name@sybase.com");
```

```
 URLConnection conn = url.openConnection();
 PrintStream out = new
  PrintStream(conn.getOutputStream(), true);
out.print ("From: kennys@sybase.com"+"\r\n");
out.print ("Subject: Works Great!"+"\r\n");
out.print ("Thanks for the example - it works
great!"+"\r\n");
out.close();
System.out.printIn("Messsage Sent");
}
```

5 Installez la classe MailTo nécessaire pour l'envoi de messages électroniques à partir de la base de données :

```
select MailTo.sendIt()
Message Sent!
```

Ces opérations nécessitent une connexion au serveur.

Téléchargement d'un document HTTP

Une autre utilisation de la classe URL consiste à télécharger un document à partir d'un URL HTTP. Avant toute chose, veillez à ce votre client doit connecté à un serveur Web. Vous pouvez effectuer les opérations suivantes dans le code client :

- créer un objet URL ;
- créer un objet InputStream à partir de l'objet URL ;
- utiliser la fonction de lecture de l'objet InputStream pour lire le document.

Pour utiliser le code exemple suivant, vous devez :

- lire la totalité du document dans la mémoire d'Adaptive Server ;
- créer un nouveau flux InputStream sur le document dans la mémoire d'Adaptive Server.

Par exemple :

```
public static InputStream url_test()
    throws Exception

{
URL u = new URL("http://www.xxxxxx.com/");
Reader in = new InputStreamReader(u.openStream()));
int n=0, off;
char c[]=new char[50000];
```

```
for(off=0;(off<c.length-512)
&&((n=in.read(c,off,512))!=-1;off+=n)
for(off=0; off < c.length; off ++) {
b[off]=(byte)c[off];
in.close();
ByteArrayInputStream test =
    new ByteArrayInputStream(b,0,off);
return (InputStream) test}
```

Une fois la nouvelle classe InputStream créée, installez-la et utilisez-la pour lire un fichier texte dans la base de données et insérer des données dans une table, comme l'illustre l'exemple suivant.

```
create table t (cl text)
insert into t values (

    URLprocess.readURL)
select datalength(cl) from
--------------
       34136
```

Accès à un document externe avec XQL

Vous pouvez accéder à un document externe à l'aide de la fonction de requête XQL d'Adaptive Server qui analyse et interroge à la fois les documents XML.

Transmettez le document XML à l'analyseur XQL en tant que InputStream. Vous pouvez utiliser la classe URLProcess pour transmettre le document XML soit à la méthode XQL parse, soit à la méthode XQL query.

La classe URLProcess est disponible à l'emplacement suivant :

```
select xml.Xql.query("//ItemID",
    URLProcess.readURL
        ("http://www.myserver.com/xmltest.xml"))
```

- *$SYBASE/ASE-12_5/sample/JavaSql* pour les environnements UNIX
- *%SYBASE\ASE-12_5\sample\JavaSql* pour les environnements Windows NT

Remarques à l'attention de l'utilisateur

Certains aspects de java.net doivent être abordés avec attention :

- La plupart des objets associés avec java.net ne sont pas sérialisables ; en d'autres termes, il est impossible de les insérer dans des tables.
- Vous risquez de rencontrer l'exception "Too many open files" alors que vous n'avez ouvert qu'un petit nombre de fichiers. Vérifiez le paramètre de configuration Number of Java Sockets.
- La plupart des fonctions liées aux E/S utilisent des E/S avec buffer, ce qui signifie que vous devrez probablement vider vos données explicitement. La classe PrintWriter est un exemple d'une classe dont les données ne sont pas vidées automatiquement.

Complément d'information

Documents de référence :

- Java Examples in a Nutshell: A Desktop Quick Reference. David Flanagan, O'Reilly 1997
- Java Network Programming: Complete guide to networking, streams, and distributed computing. Hughes, Shoffner, Hamner, Bellur, Manning 1997

Ces ouvrages existent sous forme imprimée. Vous trouverez davantage de documents de référence au sujet de Java sur le site Web java.sun.com.

CHAPITRE 12

Rubriques de référence

Ce chapitre fournit des informations sur plusieurs rubriques de référence.

Rubrique	Page
Affectation	221
Conversions autorisées	223
Transfert d'objets Java-SQL vers les clients	223
Packages, classes et méthodes d'API Java supportés	224
Appel de SQL à partir de Java	227
Commandes Transact-SQL à partir des méthodes Java	228
Mappage de type de données entre Java et SQL	232
Identificateurs Java-SQL	233
Noms de classe et de package Java-SQL	234
Déclarations de colonnes Java-SQL	235
Déclarations de variables Java-SQL	235
Références de colonnes Java-SQL	236
Références de membres Java-SQL	237
Appels de méthode Java-SQL	238

Affectation

Cette section définit les règles d'affectation des éléments de données SQL dont les types de données sont des classes Java-SQL.

Chaque affectation transfère une *instance source* vers un *élément de données cible* :

- Pour une instruction insert qui spécifie une table comportant une colonne Java-SQL, cette colonne est l'élément de données cible et la valeur d'insertion est l'instance source.
- Pour une instruction update de mise à jour d'une colonne Java-SQL, cette colonne est l'élément de données cible et la valeur de mise à jour est l'instance source.

- Pour une instruction select ou fetch d'affectation à une variable ou à un paramètre, la variable ou le paramètre est l'élément de données cible et la valeur extraite est l'instance source.

Remarque Si la source est une variable ou un paramètre, elle fait référence à un objet de la machine virtuelle Java. Si la source est une référence de colonne contenant une sérialisation, les règles relatives aux références de colonne (reportez-vous à la section Références de colonnes Java-SQL, page 236) produisent une référence à un objet de la machine virtuelle Java. Ainsi, la source est une référence à un objet de la machine virtuelle Java.

Règles d'affectation pour la compilation

1 Soit SC et TC les noms de classe de compilation de la source et de la cible. Soit SC_T et TC_T des classes nommées SC et TC dans la base de données associée à la cible. De la même manière, soit SC_S et TC_S des classes nommées SC et DT dans la base de données associée à la source.

2 SC_T doit être identique à TC_T ou doit être une sous-classe de TC_T.

Règles d'affectation pour l'exécution

Supposons que DT_SC soit identique à DT_TC ou en soit une sous-classe.

- Soit RSC le nom de classe d'exécution de la valeur source. Soit RSC_S la classe nommée RSC dans la base de données associée à la source. Soit RSC_T le nom d'une classe RSC_T installée dans la base de données associée à la cible. S'il n'y a pas de classe RSC_T, une exception est générée. Si RSC_T n'est pas identique à TC_T et n'est pas une sous-classe de TC_T, une exception est générée.
- Si les bases de données associées à la source et à la cible diffèrent, l'objet source est sérialisé par sa classe courante, RSC_S, et cette sérialisation est désérialisée par la classe RSC_T qui sera associée à la base de données associée à la cible.
- Si la cible est une variable ou un paramètre SQL, la source est copiée par sa référence vers la cible.
- Si la cible est une colonne Java-SQL, la source est sérialisée et cette sérialisation est copiée vers la cible.

Conversions autorisées

Vous pouvez utiliser convert pour changer le type de données d'une expression en procédant de l'une des manières suivantes :

- Lorsque le type de données Java est un type d'objet Java, convertissez les types Java en types de données SQL, comme indiqué à la section "Mappage de type de données entre Java et SQL", page 232. L'action de la fonction convert est le mappage Java-SQL requis.
- Convertissez les types de données SQL en types Java, comme indiqué à la section "Mappage de type de données entre Java et SQL", page 232. L'action de la fonction convert est le mappage SQL-Java requis.
- Convertissez toute classe Java-SQL installée dans le système SQL en n'importe quelle autre classe Java-SQL installée dans le système SQL si le type de données de compilation de l'expression (classe source) est une sous-classe ou une superclasse de la classe cible. Sinon, une exception est générée.

Le résultat de la conversion est associé à la base de données courante.

Pour plus d'informations sur l'utilisation de la fonction convert pour les sous-types Java, reportez-vous à la section "Utilisation de la fonction de conversion SQL pour les sous-types Java".

Transfert d'objets Java-SQL vers les clients

Lorsqu'une valeur dont le type de données est un type d'objet Java-SQL est transférée d'Adaptive Server vers un client, la conversion des données de l'objet dépend du type de client :

- S'il s'agit d'un client isql, la méthode toString( ) de l'objet ou une méthode similaire est appelée et le résultat est tronqué à varchar, puis transféré au client.

Remarque Le nombre d'octets transférés au client dépend de la valeur de la variable globale @@stringsize. La valeur par défaut est 50 octets. Pour plus d'informations, reportez-vous à la section "Représentation des instances Java", page 35.

- S'il s'agit d'un client Java utilisant jConnect version 4.0 ou supérieure, le serveur transmet la sérialisation de l'objet au client. Cette sérialisation est désérialisée de façon transparente par jConnect pour produire une copie de l'objet.
- S'il s'agit d'un client bcp :
  - Si l'objet est une colonne déclarée comme in row, la valeur sérialisée contenue dans la colonne est transférée vers le client comme valeur varbinary d'une longueur définie par la taille de la colonne.
  - Sinon, la valeur sérialisée de l'objet (le résultat de la méthode writeObject de l'objet) est transférée vers le client comme valeur de type image.

Packages, classes et méthodes d'API Java supportés

Adaptive Server supporte la plupart des classes et des méthodes dans l'API Java. De plus, Adaptive Server peut imposer des restrictions liées à la sécurité et des limitations de mise en oeuvre. Par exemple, Adaptive Server ne gère pas tous les utilitaires de création et de manipulation de thread de java.lang.Thread.

Les packages supportés sont installés avec Adaptive Server et sont toujours disponibles. L'utilisateur ne peut pas les installer.

Cette section répertorie :

- les packages et les classes Java supportés
- les packages Java non supportés
- les méthodes java.sql non supportées

Les packages et les classes Java supportés

- java.io
  - Externalizable
  - DataInput
  - DataOutput
  - ObjectInputStream
  - ObjectOutputStream
  - Serializable

- java.lang – reportez-vous à la section "Méthodes et interfaces java.sql non supportées", page 226 pour visualiser la liste des classes non supportées dans java.lang.
- java.math
- java.net – reportez-vous au chapitre 11, "Accès au réseau avec java.net".
- java.sql – reportez-vous à la section "Méthodes et interfaces java.sql non supportées", page 226 pour visualiser la liste des méthodes et interfaces non supportées dans java.sql.
- java.text
- java.util
- java.util.zip

Packages et classes Java non supportés

- java.applet
- java.awt
- java.awt.datatransfer
- java.awt.event
- java.awt.image
- java.awt.peer
- java.beans
- java.lang.Thread
- java.lang.ThreadGroup
- java.rmi
- java.rmi.dgc
- java.rmi.registry
- java.rmi.server
- java.security
- java.security.acl
- java.security.interfaces

Méthodes et interfaces *java.sql* non supportées

- Connection.commit( )
- Connection.getMetaData( )
- Connection.nativeSQL( )
- Connection.rollback( )
- Connection.setAutoCommit( )
- Connection.setCatalog( )
- Connection.setReadOnly( )
- Connection.setTransactionIsolation( )
- DatabaseMetaData.* – la commande DatabaseMetaData est supportée, sauf pour les méthodes suivantes :
  - deletesAreDetected()
  - getUDTs()
  - insertsAreDetected()
  - updatesAreDetected()
  - othersDeletesAreVisible()
  - othersInsertsAreVisible()
  - othersUpdatesAreVisible()
  - ownDeletesAreVisible()
  - ownInsertsAreVisible()
  - ownUpdatesAreVisible()
- PreparedStatement.setAsciiStream( )
- PreparedStatement.setUnicodeStream( )
- PreparedStatement.setBinaryStream( )
- ResultSetMetaData.getCatalogName( )
- ResultSetMetaData.getSchemaName( )
- ResultSetMetaData.getTableName( )
- ResultSetMetaData.isCaseSensitive( )
- ResultSetMetaData.isReadOnly( )

- ResultSetMetaData.isSearchable( )
- ResultSetMetaData.isWritable( )
- Statement.getMaxFieldSize( )
- Statement.setMaxFieldSize( )
- Statement.setCursorName( )
- Statement.setEscapeProcessing( )
- Statement.getQueryTimeout( )
- Statement.setQueryTimeoutt( )

Appel de SQL à partir de Java

Adaptive Server fournit un pilote JDBC natif, java.sql, qui met en oeuvre les spécifications JDBC 1.1. Ce pilote est décrit à l'adresse Internet suivante http://www.javasoft.com. java.sql permet aux méthodes Java exécutées dans Adaptive Server d'effectuer des opérations SQL.

Considérations spéciales

java.sql.DriverManager.getConnection( ) accepte les URL suivants :

- null
- "" (la chaîne NULL)
- jdbc:default:connection

Lorsque SQL est appelé à partir de Java, certaines restrictions s'appliquent :

- Une requête SQL qui exécute des actions de mise à jour (update, insert ou delete) ne peut pas utiliser les utilitaires de java.sql pour appeler d'autres opérations SQL qui exécutent également des actions de mise à jour.
- Les triggers déclenchés par SQL à l'aide des utilitaires de java.sql ne peuvent pas générer de jeu de résultats.
- java.sql ne peut pas être utilisé pour exécuter des procédures stockées étendues ni des procédures stockées à distance.

Commandes Transact-SQL à partir des méthodes Java

Vous pouvez utiliser certaines commandes Transact-SQL dans les méthodes Java appelées dans le système SQL. Le tableau 12-1 répertorie les commandes Transact-SQL et indique si elles sont utilisables ou non dans les méthodes Java. Vous trouverez d'autres informations sur la plupart de ces commandes dans Adaptive Server Enterprise - Manuel de référence de Sybase.

Tableau 12-1 : Support des commandes Transact-SQL

Commande	Etat
alter database	Non
alter role	Non
alter table	Oui
begin ... end	Oui
begin transaction	Non
break	Oui
case	Oui
checkpoint	Non
commit	Non
compute	Non
connect - disconnect	Non
continue	Oui
create database	Non
create default	Non
create existing table	Non
create function	Oui
create index	Non
create procedure	Non
create role	Non
create rule	Non
create schema	Non
create table	Oui
create trigger	Non
create view	Non
cursors	Non Seuls les "curseurs serveur" (server cursors) sont supportés, c'est-à-dire les curseurs déclarés et utilisés dans une procédure stockée.
dbcc	Non
declare	Oui

Commande	Etat
disk init	Non
disk mirror	Non
disk refit	Non
disk reinit	Non
disk remirror	Non
disk unmirror	Non
drop database	Non
drop default	Non
drop function	Oui
drop index	Non
drop procedure	Non
drop role	Non
drop rule	Non
drop table	Oui
drop trigger	Non
drop view	Non
dump database	Non
dump transaction	Non
execute	Oui
goto	Oui
grant	Non
clauses group by et having	Oui
if…else	Oui
insert table	Oui
kill	Non
load database	Non
load transaction	Non
online database	Non
order by Clause	Oui
prepare transaction	Non
print	Non
raiserror	Oui
readtext	Non
return	Oui
revoke	Non
rollback trigger	Non
rollback	Non

Commande	Etat
save transaction	Non
set	Reportez-vous au tableau 12-2 pour les options set.
setuser	Non
shutdown	Non
truncate table	Oui
union Operator	Oui
update statistics	Non
update	Oui
use	Non
waitfor	Oui
where Clause	Oui
while	Oui
writetext	Non

Le tableau 12-2 répertorie les options de commande set et indique si elles sont utilisables dans des méthodes Java.

Tableau 12-2 : Support des options de la commande set

Option de commande set	Etat
ansinull	Oui
ansi_permissions	Oui
arithabort	Oui
arithignore	Oui
chained	Non Voir la remarque 1.
char_convert	Non
cis_rpc_handling	Non
close on endtran	Non
cursor rows	Non
datefirst	Oui
dateformat	Oui
fipsflagger	Non
flushmessage	Non
forceplan	Oui
identity_insert	Oui
language	Non
lock	Oui
nocount	Oui
noexec	Non

Option de commande set	Etat
offsets	Non
or_strategy	Oui
parallel_degree	Oui Voir la remarque 2.
parseonly	Non
prefetch	Oui
process_limit_action	Oui Voir la remarque 2.
procid	Non
proxy	Non
quoted_identifier	Oui
replication	Non
role	Non
rowcount	Oui
scan_parallel_degree	Oui Voir la remarque 2.
self_recursion	Oui
session_authorization	Non
showplan	Oui
sort_resources	Non
statistics io	Non
statistics subquerycache	Non
statistics time	Non
string_rtruncation	Oui
stringsize	Oui
table count	Oui
textsize	Non
transaction iso level	Non. Voir la remarque 1.
transactional_rpc	Non

Remarque (1) Les commandes set avec les options chained ou transaction isolation level ne sont autorisées que si la configuration qu'elles spécifient est déjà en vigueur. En d'autres termes, les commandes set sont autorisées lorsqu'elles sont sans effet. Ceci permet de gérer les techniques de codage habituelles des procédures stockées.

Remarque (2) Les commandes set relatives au degré de parallélisation sont autorisées mais sont sans effet. Ceci permet de gérer les procédures stockées définissant le degré de parallélisation pour d'autres contextes.

Mappage de type de données entre Java et SQL

Adaptive Server mappe les types de données SQL sur les types Java (mappage de type de données SQL-Java) et les types scalaires Java sur les types de données SQL (mappage de type de données Java-SQL). Le tableau 12-3 présente le mappage de type de données SQL-Java.

Tableau 12-3 : Mappage des types de données SQL sur les types Java

Type SQL	Type Java
char	String
varchar	String
nchar	String
nvarchar	String
texte	String
numeric	java.math.BigDecimal
decimal	java.math.BigDecimal
money	java.math.BigDecimal
smallmoney	Java.math.BigDecimal
bit	boolean
tinyint	byte
smallint	short
integer	int
real	float
float	double
double precision	double
binary	byte[ ]
varbinary	byte[ ]
image	byte[ ]
datetime	java.sql.Timestamp
smalldatetime	java.sql.Timestamp

Le tableau 12-4 présente le mappage de type de données Java-SQL.

Tableau 12-4 : Mappage des types scalaires Java sur les types de données SQL

Type scalaire Java	Type SQL
boolean	bit
byte	tinyint
short	smallint
int	integer
long	integer
float	real
double	double

Identificateurs Java-SQL

Description

Les identificateurs Java-SQL sont des sous-ensembles d'identificateurs Java qui peuvent être référencés dans le système SQL.

Syntaxe

identificateur_java_sql ::= caractère alphabétique | tiret bas (_)
[caractère alphabétique | chiffre arabe | tiret bas (_) |
dollar ($)]

Utilisation

- Les identificateurs Java-SQL peuvent avoir une longueur maximale de 255 octets s'ils sont entre guillemets. Sinon, leur longueur ne doit pas dépasser 30 octets.
- Le premier caractère de l'identificateur doit être un caractère alphabétique (majuscule ou minuscule) ou un tiret bas (_). Les caractères qui suivent peuvent être des caractères alphabétiques (majuscules ou minuscules), des chiffres, des symboles dollar ($) ou des tirets bas (_).
- Dans les identificateurs Java-SQL, la distinction majuscules/minuscules est toujours appliquée.

Identificateurs délimités

- Les identificateurs délimités sont des noms d'objet entre guillemets. L'utilisation d'identificateurs délimités pour les identificateurs Java-SQL permet de contourner certaines restrictions sur les noms des identificateurs Java-SQL.

Remarque Vous pouvez mettre les identificateurs Java-SQL entre guillemets, que l'option set quoted_identifier soit activée on ou désactivée off.

- Les identificateurs délimités permettent d'utiliser des mots réservés SQL pour les packages, les classes, les méthodes, etc. Chaque fois que vous utilisez un identificateur délimité dans une instruction, vous devez le mettre entre guillemets. Par exemple :

```
create table t1
       (c1 char(12)
       c2 p1."select".p2."jar")
```

- Seuls les identificateurs Java-SQL sont entre guillemets, pas le nom qualifié.

Voir aussi

Pour plus d'informations sur les identificateurs, reportez-vous au chapitre 5, section "Rubriques Transact-SQL" du *Manuel de référence*.

Noms de classe et de package Java-SQL

Description

Pour référencer une classe ou un package Java-SQL, utilisez la syntaxe suivante :

Syntaxe

nom_classe_java_sql ::=
[*nom_package_java_sql.*]*identificateur_java_sql*

nom_package_java_sql ::=
[*nom_package_java_sql.*]*identificateur_java_sql*

Paramètres

nom_classe_java_sql
Nom qualifié d'une classe Java-SQL dans la base de données courante.

nom_package_java_sql
Nom qualifié d'un package Java-SQL dans la base de données courante.

identificateur_java_sql
Reportez-vous à la section Identificateurs Java-SQL.

Utilisation

S'agissant des noms de classe Java-SQL :

- Une référence de nom de classe fait toujours référence à une classe de la base de données courante.
- Si vous spécifiez un nom de classe Java-SQL sans référencer le nom de package, seule une classe Java-SQL portant ce nom doit exister dans la base de données courante, et son package doit être le package par défaut (anonyme).
- Si un type de données SQL défini par l'utilisateur et une classe Java-SQL possèdent la même séquence d'identificateurs, Adaptive Server utilise le nom de type de données SQL défini par l'utilisateur et ignore le nom de la classe Java-SQL.

S'agissant des noms de package Java-SQL :

- Si vous spécifiez un nom de sous-package Java-SQL, vous devez le référencer avec son nom de package :

  nom_package_java_sql.nom_sous-package_java_sql

- N'utilisez les noms de package Java-SQL que comme qualificatifs de noms de classe ou de sous-package et pour supprimer des packages de la base de données à l'aide de la commande remove java.

Déclarations de colonnes Java-SQL

Description

Pour déclarer une colonne Java-SQL lorsque vous créez ou que vous modifiez une table, utilisez la syntaxe suivante :

Syntaxe

colonne_java_sql ::= nom_colonne nom_classe_java_sql

Paramètres

colonne_java_sql
Spécifie la syntaxe des déclarations de colonnes Java-SQL.

nom_colonne
Nom de la colonne Java-SQL.

nom_classe_java_sql
Nom d'une classe Java-SQL dans la base de données courante. Il s'agit de la "classe déclarée" de la colonne.

Utilisation

- La classe déclarée doit mettre en oeuvre l'interface Serializable ou Externalizable.
- Une colonne Java-SQL est toujours associée à la base de données courante.
- Une colonne Java-SQL ne peut pas être spécifiée comme :
  - non-NULL
  - unique
  - une clé primaire

Voir aussi

N'utilisez la déclaration de colonne Java-SQL que lorsque vous créez ou modifiez une table. Pour plus d'informations sur create table et alter table, reportez-vous au document *Manuel de référence*.

Déclarations de variables Java-SQL

Description

Utilisez les déclarations de variables Java-SQL pour déclarer des paramètres de variable et de procédure stockée pour les types de données correspondant à des classes Java-SQL.

Syntaxe

variable_java_sql ::= @*nom_variable nom_classe_java_sql*

paramètre_java_sql ::= @*nom_paramètre nom_classe_java_sql*

Paramètres

variable_java_sql
Spécifie la syntaxe d'une variable Java-SQL dans une procédure stockée SQL.

paramètre_java_sql
Spécifie la syntaxe d'un paramètre Java-SQL dans une procédure stockée SQL.

nom_classe_java_sql
Nom d'une classe Java-SQL dans la base de données courante.

Utilisation

Les paramètres *variable_java_sql* ou *paramètre_java_sql* sont toujours associés à la base de données qui contient la procédure stockée.

Voir aussi

Pour plus d'informations sur les déclarations de variables, reportez-vous au document *Manuel de référence*.

Références de colonnes Java-SQL

Description

Pour référencer une colonne Java-SQL, utilisez la syntaxe suivante :

Syntaxe

référence_colonne ::=
[[[*nom_basededonnées*.]*propriétaire*.]*nom_table*.]*nom_colonne*
| *nom_basededonnées*..*nom_table*.*nom_colonne*

Paramètres

référence_colonne
Référence à une colonne dont le type de données est une classe Java-SQL.

Utilisation

- Si la valeur de la colonne est NULL, la référence de la colonne est également NULL.
- Si la valeur de la colonne est une sérialisation Java S et que le nom de sa classe est CS :
  - Si la classe CS n'existe pas dans la base de données courante ou si CS n'est pas le nom d'une classe de la base de données associée à la sérialisation, une exception est générée.

Remarque La base de données associée à la sérialisation est généralement la base de données qui contient la colonne. Cependant, les sérialisations contenues dans les tables de travail et dans les tables temporaires créées avec "insert into #tempdb" sont associées à la base de données dans laquelle la sérialisation a été initialement stockée.

- La valeur de la référence de colonne est :

  ```
  CSC.readObject(S)
  ```

  où CSC est la référence de colonne. Si l'expression génère une exception Java non détectée, une exception est générée.

  L'expression produit une référence à un objet de la machine virtuelle Java, qui est rattachée à la base de données associée à la sérialisation.

Références de membres Java-SQL

Description

Pour référencer le champ ou la méthode d'une classe ou d'une instance de classe, utilisez la syntaxe suivante :

Syntaxe

référence_membre ::= *référence_membre_classe* | *référence_membre_instance*

référence_membre_classe ::= *nom_classe_java_sql.nom_méthode*

référence_membre_instance ::= *expression_instance>>nom_membre*

expression_instance ::= *référence_colonne* | *nom_variable* | *nom_paramètre* | *appel_méthode* | *référence_membre*

nom_membre ::= *nom_champ* | *nom_méthode*

Paramètres

référence_membre
Expression décrivant un champ ou une méthode d'une classe ou d'un objet.

référence_membre_classe
Expression décrivant la méthode statique d'une classe Java-SQL.

référence_membre_instance
Expression décrivant la méthode statique ou dynamique ou le champ d'une classe Java-SQL.

nom_classe_java_sql
Nom qualifié d'une classe Java-SQL de la base de données courante.

expression_instance
Expression dont le type de données est une classe Java-SQL.

nom_membre
Nom d'un champ ou d'une méthode de la classe ou de l'instance de classe.

Utilisation

- Si un membre référence un champ d'une instance de classe, l'instance a une valeur NULL et la référence de membre Java-SQL est la cible d'une instruction fetch, select ou update, alors une exception est générée.

  Sinon, la référence de membre Java-SQL a une valeur NULL.

- La qualification avec des chevrons (>>) ou un point (.) est prioritaire sur n'importe quel autre opérateur, notamment les opérateurs d'addition (+) ou d'égalité (=), par exemple :

  ```
  X>>A1>>B1 + X>>A1>>B2
  ```

  Dans cette expression, l'addition est exécutée après que les membres aient été référencés.

- Le champ ou la méthode désignés par une référence de membre est associé à la même base de données que celle de sa classe Java-SQL ou de son instance de classe Java-SQL.

  Si le type Java d'une référence de membre est l'un des types scalaires Java (tels que boolean, byte, etc.), le type de données SQL correspondant de la référence est obtenu en mappant le type Java sur son type SQL correspondant.

  Si le type Java d'une référence de membre est un type d'objet, le type de données SQL est le même type d'objet ou la même classe Java.

Appels de méthode Java-SQL

Description

Pour appeler une méthode Java-SQL renvoyant une valeur unique, utilisez la syntaxe suivante :

Syntaxe

appel_méthode ::= *référence_membre* ([*paramètres*])
| new *nom_classe_java_sql* ([*paramètres*])

paramètres ::= *paramètre* [(, *paramètre*)...]

paramètre ::= *expression*

Paramètres

appel_méthode
Appel de méthode statique, de méthode d'instance ou de constructeur de classe. Un appel de méthode peut être utilisé dans une expression où une valeur du type de données de la méthode est requise.

référence_membre
Référence de membre dénotant une méthode.

paramètre
Liste des paramètres à transmettre à la méthode. S'il n'y a aucun paramètre, incluez des parenthèses vides.

Utilisation

Remplacement de méthode

- Lorsque des méthodes homonymes se trouvent dans la même classe ou la même instance, le problème est résolu conformément aux règles de surcharge de méthodes Java.

Type de données d'appels de méthode

- Le type de données d'un appel de méthode est déterminé de la manière suivante :
  - Si un appel de méthode spécifie new, son type de données est celui de sa classe Java-SQL.
  - Si un appel de méthode spécifie une référence de membre dénotant une méthode de type, le type de données de l'appel de méthode est ce type.
  - Si un appel de méthode spécifie une référence de membre dénotant une méthode statique void, le type de données de l'appel de méthode est de type SQL integer.
  - Si un appel de méthode spécifie une référence de membre dénotant une méthode d'instance void d'une classe, le type de données de l'appel de méthode est celui de la classe.
- Pour inclure un paramètre dans une référence de membre lorsque ce paramètre est une instance Java-SQL associée à une autre base de données, vous devez vous assurer que le nom de classe associé à l'instance Java-SQL est inclus dans les deux bases de données. Sinon, une exception est générée.

Résultats de l'exécution

- Le résultat de l'exécution d'un appel de méthode dépend des conditions suivantes :
  - Si un appel de méthode spécifie une référence de membre dont la valeur d'exécution est NULL (autrement dit, une référence à un membre d'une instance NULL), le résultat est NULL.
  - Si un appel de méthode spécifie une référence de membre dénotant une méthode de type, le résultat est la valeur renvoyée par la méthode.

- Si un appel de méthode spécifie une référence de membre dénotant une méthode statique void, le résultat est la valeur NULL.
- Si un appel de méthode spécifie une référence de membre dénotant une méthode d'instance void d'une instance de classe, le résultat est une référence à cette instance.
- L'appel de méthode et le résultat de l'appel de méthode sont associés à la même base de données.
- Adaptive Server ne transmet pas la valeur NULL comme valeur de paramètre à une méthode dont le type Java est scalaire.

Glossaire

Ce glossaire définit les termes et expressions Java et Java-SQL utilisés dans ce manuel. Pour une définition de la terminologie SQL et Adaptive Server, reportez-vous au document *Glossaire Adaptive Server*.

affectation

Terme générique désignant les transferts de données spécifiés par les commandes Transact-SQL select, fetch, insert et update. Une affectation définit une valeur source dans un élément de données cible.

archive Java (JAR)

Format indépendant de la plate-forme conçu pour collecter des classes dans un seul fichier. Format non tributaire de la plate-forme permettant de regrouper les classes dans le même fichier. Voir aussi **(fichier) JAR enregistré**.

bytecode

Code octet. Forme compilée du code source Java exécuté par la machine virtuelle Java.

classe

Elément de base des programmes Java contenant un ensemble de déclarations et de méthodes de champs. Une classe est la copie maître qui détermine le comportement et les attributs de chaque instance de cette classe. Une définition de classe est la définition d'un type de données actives qui spécifie un ensemble légal de valeurs et définit un ensemble de méthodes qui gèrent ces valeurs. Reportez-vous à la section **instance de classe**.

classe déclarée

Type de données déclaré d'un élément de données Java-SQL. Il s'agit du type de données de la valeur d'exécution ou d'un supertype de celle-ci.

classe Java-SQL

Classe Java publique installée dans le système Adaptive Server. Elle est composée d'un ensemble de définitions de variables et de méthodes.

Une instance de classe est composée d'une instance de chacun des champs de la classe. Le type des instances de classe est principalement déterminé par le nom des classes.

Une sous-classe est une classe déclarée pour étendre (au maximum) une autre classe. Cette dernière est appelée la superclasse directe de la sous-classe. Une sous-classe possède toutes les variables et les méthodes de ses superclasses directes et indirectes et elle est interchangeable avec celles-ci.

classes installées — Les classes et les méthodes installées Java ont été placées dans le système Adaptive Server par l'utilitaire installjava.

classes synonymes — Classes Java-SQL portant le même nom qualifié mais installées dans différentes bases de données.

colonne Java-SQL — Colonne SQL dont le type de données est une classe Java-SQL.

conversion élargissante — Opération Java permettant de convertir une référence d'instance de classe en une référence d'instance d'une superclasse de cette classe. Cette opération est écrite en SQL avec la fonction convert. Voir aussi **conversion rétrécissante**.

conversion rétrécissante — Opération Java permettant de convertir une référence d'instance de classe en une référence d'instance d'une sous-classe de cette classe. Cette opération est écrite en SQL avec la fonction convert. Voir aussi **conversion élargissante**.

déclaration de type de données (DTD) — En langage XML, chaque document correct possède une DTD qui décrit les éléments disponibles dans ce type de document. Une DTD peut être encapsulée dans le document XML ou référencée par celui-ci.

document correct — En langage XML, un document correct possède une DTD et s'y conforme. On parle aussi de **document respectant les règles XML**.

document respectant les règles XML — XML fournit un ensemble de règles syntaxiques génériques qui doivent être respectées. Les caractéristiques requises d'un document bien structuré sont : tous les éléments ont des balises de début et de fin, les valeurs d'attribut sont entre guillemets, tous les éléments sont correctement imbriqués.

eXtensible Markup Language (XML) — Métalangage conçu pour les applications Web qui vous permet de définir vos propres balises et attributs de marquage pour différents types de documents. Le langage XML est un sous-ensemble du langage SGML.

eXtensible Query Language (XQL) — Langage de marquage conçu pour interroger les documents XML stockés dans une base de données relationnelle. Adaptive Server fournit un moteur de requête XQL que vous pouvez installer dans Adaptive Server ou exécuter comme programme autonome.

eXtensible Style Language (XSL) — Langage de marquage conçu pour formater les documents XML en HTML ou d'autres documents XML avec différents attributs et balises.

externalisation — Une externalisation d'instance Java est un flux d'octets qui contient suffisamment d'informations pour que la classe puisse reconstruire l'instance. L'externalisation est définie par l'interface externalisable. Toutes les classes Java-SQL doivent être externalisables ou sérialisables. Reportez-vous à la section **sérialisation**.

fichier de classes
Fichier de type "class" (par exemple, *myclass.class*) contenant le bytecode (code octet) compilé d'une classe Java. Voir **fichier Java** et **archive Java (JAR)**.

(fichier) JAR enregistré
Reportez-vous à la section **archive Java (JAR)**.

fichier Java
Fichier de type "java" (par exemple, *myfile.java*) contenant du code source Java. Voir **fichier de classes** et **archive Java (JAR)**.

Hypertext Markup Language (HTML)
Sous-ensemble du langage SGML conçu pour le Web.

instance de classe
Valeur du type de données de la classe qui contient une valeur pour chaque champ de la classe et qui en accepte toutes les méthodes.

interface
Ensemble nommé de déclarations de méthode. Une classe peut faire appel à une interface si la classe définit toutes les méthodes déclarées dans l'interface.

Java Database Connectivity (JDBC)
API Java-SQL et partie standard des Java Class Libraries qui contrôlent le développement d'applications Java. JDBC fournit des utilitaires similaires à ceux d'ODBC.

Java Development Kit (JDK)
Gamme d'outils de Sun Microsystems permettant d'écrire et de tester des programmes Java à partir du système d'exploitation.

Machine virtuelle Java (Java VM)
Interpréteur Java qui traite Java dans le serveur. Il est appelé par la mise en œuvre SQL.

mappable
Un type de données Java est mappable s'il est :

- répertorié dans la première colonne du tableau 12-3, page 232 ;
- une classe Java-SQL publique installée dans le système Adaptive Server.

Un type de données SQL est mappable s'il est :

- répertorié dans la première colonne du tableau 12-4, page 232 ;
- une classe Java-SQL publique intégrée ou installée dans le système Adaptive Server.

Une méthode Java est mappable si tous ses types de données paramètre et résultat sont mappables.

mappage de type de données Java-SQL
Conversions entre les types de données Java et SQL. Reportez-vous à la section "Mappage de type de données entre Java et SQL", page 232.

mappage de type de données SQL-Java
Conversions entre les types de données Java et SQL. Reportez-vous à la section "Mappage de type de données entre Java et SQL", page 232.

mappage de types de données — Conversions entre les types de données Java et SQL.

méthode — Ensemble d'instructions contenues dans une classe Java permettant d'exécuter une tâche. Une méthode peut être déclarée statique, auquel cas elle est appelée méthode de classe. A défaut, il s'agit d'une méthode d'instance. Il est possible de référencer des méthodes de classe en qualifiant le nom de méthode avec le nom de classe ou le nom d'une instance de la classe. Les méthodes d'instance sont référencées par qualification du nom de méthode avec le nom d'une instance de la classe. Le corps d'une méthode d'instance peut référencer les variables locales de cette instance.

méthode de classe — Reportez-vous à la section **méthode statique**.

méthode d'instance — Méthode appelée qui référence une instance spécifique d'une classe.

méthode statique — Méthode appelée sans référence d'un objet. Les méthodes statiques influent sur l'ensemble de la classe, pas seulement sur une instance de celle-ci. Également appelée expression de mot-clé.

objet Java — Instance d'une classe Java contenue dans la mémoire de la machine virtuelle Java. Les instances Java référencées en SQL sont les valeurs des colonnes Java ou des objets Java.

package — Ensemble de classes connexes. Une classe spécifie un package ou fait partie d'un package anonyme par défaut. Une classe peut utiliser des instructions Java import pour spécifier d'autres packages dont les classes peuvent être référencées.

procédure — Procédure stockée SQL ou méthode Java avec un type de résultat *void*.

public — Champs et méthodes publics, tels que définis en Java.

sérialisation — La sérialisation d'une instance Java est un flux d'octets contenant suffisamment d'informations pour identifier sa classe et reconstruire l'instance. Toutes les classes Java-SQL doivent être externalisables ou sérialisables. Reportez-vous à la section **externalisation**.

signature de fonction SQL — Type de données SQL de chacun des paramètres d'une fonction SQLJ.

signature de méthode Java — Type de données Java de chacun des paramètres d'une méthode Java.

signature de procédure SQL — Type de données SQL de chacun des paramètres d'une procédure SQLJ.

sous-classe Classe hiérarchiquement inférieure à une autre classe. Elle hérite des attributs et du comportement des classes supérieures. Une sous-classe est interchangeable avec ses superclasses. La classe au-dessus de la sous-classe est sa superclasse directe. Voir aussi **superclasse**, **conversion rétrécissante** et **conversion élargissante**.

superclasse Classe hiérarchiquement supérieure à une ou plusieurs classes. Elle transmet ses attributs et son comportement aux classes inférieures. Elle n'est pas interchangeable avec ses sous-classes. Voir aussi **sous-classe**, **conversion rétrécissante** et **conversion élargissante**.

types de données Java Classes Java définies par l'utilisateur ou provenant de l'API JavaSoft ou types de données primitives Java tels que boolean, byte, short et int.

Unicode Jeu de caractères 16 bits défini par le code ISO 10646 qui gère plusieurs langues.

variable En langage Java, une variable est locale par rapport à une classe, aux instances de cette classe ou à une méthode. Une variable déclarée statique est locale par rapport à la classe. Les autres variables déclarées dans la classe sont locales par rapport aux instances de la classe. Ces variables sont appelées des champs de la classe. Une variable déclarée dans une méthode est locale pour cette méthode.

variable Java-SQL Variable SQL dont le type de données est une classe Java-SQL.

visible Une classe Java installée dans un système SQL est visible en SQL si elle est déclarée public ; un champ ou une méthode d'une instance Java est visible en SQL si elle est public et mappable. Les classes, les champs et les méthodes visibles peuvent être référencés en SQL. Les autres classes, champs et méthodes ne le peuvent pas, y compris les classes private, protected ou friendly et les champs et méthodes private, protected ou friendly ou qui ne sont pas mappables.

Index

Symboles

[ ] (crochets)
  dans les instructions SQL xvii
() (parenthèses)
  dans les instructions SQL xvii
, (virgule)
  dans les instructions SQL xvii
>> (chevrons)
  pour qualifier les champs et les méthodes Java 238
{} (accolades)
  dans les instructions SQL xvii

A

accès au réseau, java.net 211
accès serveur aux éléments Commande 172
accolades ({})
  dans les instructions SQL xvii
activation de java.net, procédure 212
activation Java 15
Adaptive Server
  installation de XQL 135
  module de connexion 30, 89
adresse Web
  W3C, Document Object Model (DOM) 123
  W3C, Extensible Markup Language (XML) 123
  W3C, Extensible Stylesheet Language (XSL) 123
  World Wide Web Consortium (W3C) 123
affectation 221
allString booléenne, méthode Java 178
alter table
  commande 135
  syntaxe 30
analyse XML avec SAX 132
analyseur XML 132, 133
  application portable utilisant les interfaces SAX et DOM 132
  domaine public 132
  interface standard 132
  licence gratuite 132
API Java 9
  accès à partir de SQL 9
  package supporté 224–227
  support par Sybase 9
appel
  méthode 33
  méthode Java 87
  méthode Java, appel direct 87
  méthode Java, avec SQLJ 88
  SQL à partir de Java 227, 231
appel de méthode 238
  type de données 239
appel méthode Java, avec SQLJ 87
appendItem, méthode Java 172
application autonome
  exemple 146
  utilisant XQL 145
application de présentation, utilisation de XSL pour 130
arbre d'analyse syntaxique
  assemblage avec DOM 132
  construction et modification avec DOM 132
  construire ou modifier 132
  génération d'une représentation Java 132
  génération du texte d'un document à partir de 132
argument
  interclasse 57
arrêt
  en utilisant des conditions 203
  lorsque l'exécution n'est pas interrompue 205
  sur un numéro de ligne 202
  sur une méthode de classe 203
  utilisation de compteurs 203
assemblage de documents Commande 164
attribut imbriqué dans les balises d'élément 126

autorisation
Java 7, 26
JDBC 66
routine SQLJ 85

B

balise
définie par l'utilisateur 125
HTML, mise entre crochets incohérente 128
HTML, paragraphe 128
personnalisation dans XML 123
XML strictement imbriquée 125
balise d'élément
attribut imbriqué 126
définie par l'utilisateur 125
HTML, incohérence 128
imbrication stricte 126
personnalisation 126
balise d'élément incohérente, HTML 128
base de données de travail 57
base de données temporaire 57
bases de données multiples 54
besoins en mémoire
configuration 137
Java Services 137
pour le moteur de requêtes 137
besoins en mémoire, paramètres Java Services 133
bookstore.xml
commande : validate 145
DTD, conforme 145
exemple auteurs 144
exemple XML 144
nom de fichier 144
page Web 146

C

chaîne 47
longueur nulle 47
chevrons
pour qualifier les champs et les méthodes Java 32, 238
classe définie par l'utilisateur, création 16
classe exemple 58–60
address2Line 59
adresse 58
emplacement 12
JDBCExamples 66–81
JDBCExamples, méthodes 67–73
JDBCExamples, présentation 66
Misc 60
OrderXml 155
ResultSet 179
classe HoldString 141
classe InputStream 216
classe Java 155
classe URL, utilisation 214
Commande 172
comme type de données 3, 27
création 16
DatagramPacket 212
DatagramSocket 212
définie par l'utilisateur 9, 14
enregistrement 23
exemple SQLJ 86
HoldString 141
HttpURLConnection 212
InetAddress 212
InputStream 213, 216, 218
installation 17–21
JXml 156, 175
MailTo 216
mise à jour 19
MulticastSocket 212
OrderXml 155, 156, 175
OutputStream 213, 216
PrintWriter 219
référencement d'autres classes 20
ResultSet 179
ResultSetXml 175
sauvegarde dans un fichier JAR 17
ServerSocket 212, 214
Socket 212
sous-type 40
supportée 9
URL 212, 216, 217
URLConnection 212
URLDecoder 212
URLEncoder 212
version d'exécution 14

classe Java nécessaire à l'installation 14
emplacement de la version d'exécution 14
classe java.net
HttpURLConnection 212
InetAddress 212
Socket 212
URL 212
URLConnection 212
URLDecoder 212
URLEncoder 212
classe java.net, voir classe Java
classe Java-SQL
dans les bases de données multiples 53
installation 17–21
classe JDBCExamples 76–81
classe Socket, utilisation 213
classe SQLJExamples 116
classe URL
accès au serveur externe avec XQL 216
classe Java 212, 216, 217
envoi d'un message électronique 216
insertion de données dans une table 216
téléchargement d'un document HTTP 216
utilisation 216
classe, voir classe Java
CLASSPATH
programme autonome 134
variable d'environnement 134
variable d'environnement pour UNIX et NT 134
xerces.jar, xml.zip, runtime.zip 134
clause
group by 51
order by 51
where 41, 49, 52
where, effet sur le traitement 140
where, non utilisée pour le stockage du jeu de résultats 141
client
bcp 223, 224
isql 223
codage caractère, voir jeu de caractères
code exemple
DTD, exemple de commande 129
HTML, exemple de commande 127
XML, exemple d'article 126
XML, exemple d'informations 126
XML, exemple de commande DTD 129
code Java
compilation 16
écriture 16
colonne
déclaration 235
référencement 236
colonne Java-SQL
option de stockage 28
com 153
com.sybase.xml.xql.store.SybMemXmlStream, XQL
interface 153
com.sybase.xml.xql.Xql
méthode, spécifique à 149, 150, 153, 154
com.sybase.xml.xql.XqlDriver
commande 133
fichier local 142
programme autonome 142
requête dans un document XML 142
syntaxe 143
utilisation 142
commande
aide 143
alter table 30, 135
charindex 132
clause where 140
com.sybase.xml.xql.XqlDriver 133
create function SQLJ 90
create procedure (SQLJ) 96, 99
create table 28, 29
create table, syntaxe 28, 29
déboguer 143
debug 143
drop function 96
DTD, code exemple 129
FileInputStream() 146
help 143
infile 143
insertion 131
insertion de la clause "values" 131
Java, voir commande
new (nouveau) 135
outfile 143
parse 145
parse() 135, 136
patindex 132

qstring 143
query 145
query() 141
remove java 22, 234
select 141
set, autorisée dans méthode Java 230
set, mise à jour 49
SQLJ create procedure 96
substring 132
table 180
update (mise à jour) 135
URL 146
valid 145
validate 143, 145
validation 143
writetext 131
Commande, classe Java 172
compilateur Java 199
compilation du code Java 16
complément d'information
à propos de Java 11
XML 123
configuration 212
configuration des besoins en mémoire 137
connexion, établissement 69
constructeur 30, 31, 48
OrderXml 165, 167
constructeur Java
OrderXml 156
constructeur Java ResultSetXm 176
construction d'un arbre d'analyse syntaxique avec un analyseur XML 132
conventions
syntaxe Java-SQL xvi
syntaxe Transact-SQL xvii
conventions syntaxiques
Java-SQL xvi
Transact-SQL xvii
conversion 223
élargissante 41
rétrécissante 41
conversion de données 164, 166
conversion de type de données 223
conversion rétrécissante 41
création
application cliente 211
application réseau, java.net 211
classe définie par l'utilisateur 16
et remplissage de tables SQL 162
feuille de style XSL 130
table 28
crochets [ ]
dans les instructions SQL xvii

D

DatagramPacket, classe Java 212
débogage
Java 197–210
débogueur
compilation des classes 199
conditions requises 198
déconnexion 205
définition de conditions d'interruption 198
définition de points de rupture 198
démarrage 199
emplacement 198
fonctionnement 197
mode d'attente 200
option 201
parcourir le contenu des classes 198
réquisition d'une machine virtuelle Java 200
suivi d'exécution 198
vérification des expressions et interruption de l'exécution 198
vérification et définition des variables 198
débogueur fenêtre
Breakpoints 200
Calls 200
Classes 200
Connection 201
Exceptions 201
Inspection 201
Locals 201
Source 200
déclaration de colonne 235
déclaration de variable 235
déclaration XML, pour spécifier le jeu de caractères 125
définition de la table 86

désactivation de la conversion des jeux de caractères 131
désactivation Java 15
didacticiel de débogage 205–210
  chargement du code source 207
  code source 205
  démarrage du débogueur 206
  exemple 208
  modification des variables locales 210
  passage en revue du code source 208
  réquisition d'une machine virtuelle Java 206
  vérification des variables 208
  vérification des variables locales 209
  vérification des variables statiques 210
document Commande
  génération sur le client 165
  génération sur le serveur 166
document HTML cible, à partir de XSL 130
Document Object Model, voir DOM
document respectant les règles XML 126
Document Type Definition, voir DTD
document XML
  accès aux éléments 169
  accès en XQL 160
  balise de marquage imbriquée 125
  client ou serveur 160
  code exemple DTD 129
  code exemple, commande 124
  correct avec une DTD 130
  création à partir d'Adaptive Server 122
  création à partir des données SQL 121
  en tant que données caractère 125
  exécution sur le serveur 168
  exemple, informations 126
  insertion dans la base de données 131
  lecture à partir de la base de données 131
  mappage et stockage 160
  mise à jour 136
  partie 125
  présentation 130
  recherche avec XQL 122
  recherche sur le Web 122
  requête 142
  respectant les règles 126
  sans instructions de présentation 126
  stockage 160
  stockage dans Adaptive Server 121, 122
  stocké comme fichier système 145
  stocké sur le Web 145
  suppression 136
  valeur de jeu de caractères 131
document XML correct 130
document, type ResultSet 180
document, validation 145
DOM
  assemblage d'un arbre d'analyse syntaxique 132
  construction de l'arbre d'analyse syntaxique d'un document 132
  génération d'un arbre d'analyse syntaxique 132
  interface XML standard 132
  modification de l'arbre d'analyse syntaxique d'un document 132
  objet renvoyé par SAX 132
  portable sur les analyseurs XML 132
DOM, Document Object Model 132
données
  conversion à partir d'un document XML 166
  sélection avec XQL 133
données texte, XML 125
données XML
  insertion à partir d'un fichier client 168
  stockage d'éléments 160
  stockage de documents, données XML 161
DTD 128
  document XML correct 130
  élément 129
  interne 130
  non obligatoire dans tous les documents 130
DTD (déclaration de type de données) 126
DTD, #IMPLIED 129
DTD, astérisque (*) 129
DTD, ATTLIST 129
DTD, déclaration de type de données 126
DTD, ELEMENT 129
DTD, élément #PCDATA DTD 129
DTD, point d'interrogation (?) 129
DTD, signe plus (+) 129

E

égal, opérateur 138
élément
  extraction 166
  référencement et mise à jour 169
élément de données permanent 36
élément de données temporaire 36
éléments Commande, accès serveur aux 172
enregistrement de texte en dehors d'Adaptive Server 213
environnement d'exécution Java 13
envoi d'un document par messagerie 213
exceptions 35
exécution, environnement 13
exemple
  pour routine SQLJ 86
exemple de code
  XML, exemple d'informations 126
exemple de commande
  code XML 124
  HTML 127
exemple de ResultXml, méthode Java 176
exemple XML, bookstore.xml 144
expression
  case 42, 95
expression booléenne, à l'intérieur d'un opérateur filtre 140
eXtensible Markup Language (XML), voir Extensible Markup Language
Extensible Markup Language, voir XML
Extensible Style Language, voir XSL
externalisation 235
extraction d'éléments 166

F

fenêtre Breakpoints 202
feuille de style, création 130
feuille de style, XSL 130
fichier
  JAR, compressé, installation 17
  JAR, création 17
  JAR, enregistrement 19
  JAR, installation 17
  JAR, non compressé, installation 17
fichier Java
  Debug.jar 199
fichier local, com.sybase.xml.xql.XqlDriver 142
fonction convert/de conversion 40, 223
fonction SQLJ 90–96
  suppression 96
  visualisation des informations 107

G

génération
  arbre d'analyse syntaxique Java 132
  génération de texte à partir de l'arbre d'analyse syntaxique 132
génération d'un document Commande sur le client 165
getString, méthode JDBC 183

H

HTML
  affichage des données Commande 126
  code exemple Commande 126, 127
  élément DTD 129, 130
  mise entre crochets incohérente des éléments 128
  page générée par javadoc 122
  sous-ensemble du langage SGML, voir Standardized General Markup Language
HTML (Hypertext Markup Language) 121
HttpURLConnection, classe Java 212
hybride
  table de stockage 164
Hypertext Markup Language (HTML) 121

I

identificateur 233
identificateur délimité 233
image, type de données 135, 136
imbrication de la DTD dans XML 129
implémentation SQLJ
  définie par Sybase 115
  différence entre SQLJ et Sybase 113
  fonctionnalités non supportées 114

fonctionnalités partiellement supportées 114
InetAddress, classe Java 212
informations relatives au langage XML sur le Web 123
informations, exemple de code XML 126
InputStream, classe Java 218
insertion
document XML dans la base de données 131
données dans une table 216
objet Java 30
insertion d'une commande, clause "values" 131
installation
classe Java 17, 21
fichier JAR compressé 17
fichier JAR non compressé 17
XQL dans Adaptive Server 135
instance Java, représentation 35

J

Java dans la base de données
avantages 1
caractéristiques clés 6
fonctionnalités 2
préparation 13–23
questions et réponses 6
Java Development Kit 7
Java Services
augmentation des paramètres de mémoire par défaut 133
table des besoins en mémoire 137
table, paramètres de mémoire 137
Java, SQL, utilisation conjointe 8
java.net 212, 213, 214, 219
accès au réseau 211
accès aux documents externes 213
accès aux documents via XML, JDBC 211
activation 212
aide 219
application cliente, configuration 213
classe 212
connexion via JDBC avec jConnect 211
création d'applications réseau 211
document de référence 219
écriture du serveur externe 213
enregistrement d'un document 211
enregistrement de texte à partir de Adaptive Server 213
envoi de documents par messagerie 213
envoi de messages électroniques 211
exemple 213
objet non sérialisable 219
procédure d'activation 212
processus client 214
processus serveur 214
recommandations 219
référence écrite 219
référence en ligne 219
téléchargement de documents 211
java.sql 227
javadoc, génération de pages HTML 122
Java-SQL 27
appel de méthode 238
casse 27
colonne 36, 52
création de tables 28
déclaration de colonne 235
déclaration de variable 235
identificateur 233
longueur des noms de classe 27
méthode non supportée 226
nom 27
nom de classe 234
nom de package 234
paramètre 36
référence de colonnes 236
référence de membre 237
résultat de fonction 36
transfert d'objets 223
transfert vers les clients 223
variable 36
variable statique 53
jConnect 211
JDBC 8
OrderXml dans le client 165
jConnect, utilisé par OrderXml dans le client 165
JDBC 63–81
accès aux données 66
autorisation 66
classe JDBCExamples 66
classe ResultSet 179

classe ResultSetMetaData 183
client 8, 65
concept 64
connexion 69
établissement d'une connexion 69
interface 10
paramètre de connexion par défaut 66
serveur 8, 65
terminologie 64
version supportée 14
JDBC client 8
JDBC serveur 8
jeu de caractères
client serveur 131
conversion ignorée 125
conversion, non prise en compte 131
déclaré correspondre au jeu réel 125
déclaré, réel 131
données XML 131
module d'extension Adaptive Server 89
spécification 125
spécification avec SAX 132
Unicode 30, 40, 89
UTF8, par défaut 125
XML 125, 131
jeu de résultats 111
non stocké dans la clause where 141
résultat inattendu 142

L

lecture d'un document XML à partir de la base de données 131

M

machine virtuelle Java 7, 13
MailTo, classe Java 216
mappage de type de données 38, 85, 108, 232
mappage de type de données Java et SQL 108
mappage de type de données JDBC 108
mappage, illustration avec table orders 184
messagerie électronique
java.net 211
message, envoi 211
méthode
appendItem 172
aseutils, com.sybase.xml.xql.Xql 152
com.sybase.xml.xql.store 153
exceptions 35
main() exécutée sur le client 170
order2Sql 166
OrderXml 156
pour référencer et mettre à jour les éléments 169
query, com.sybase.xml.xql.Xql 152
résultats de l'exécution 239
SQLJExamples.bestTwoEmps() 87
SQLJExamples.correctStates() 86, 99
SQLJExamples.job() 87
SQLJExamples.region() 86, 92
type 48
void 48, 98
méthode d'analyse 149
méthode d'instance 49
méthode de commande main 112
méthode Java
allString 192
allString booléenne 178
appel 33, 87
appel par référence 35, 52
Boolean someString 179
chaîne getItemElement 158
commande main 112
constructeur ResultSetXml 176
exceptions 35
exemple de ResultXml 176
getString, JDBC 183
instance 49
someString 192
static void createOrderTable 157
statique 50
string getColumn 177
string toSqlScript 177
toSqlScript() 186
type 47
void appendItem 159
void deleteItem(int itemNumber) 159
void order2Sql(String ordersTableName, String server) 157
void setColumn 178

void setItemElement 158
void setOrderElement 157
voir aussi méthode XQL
XQL 148
méthode Java, spécifique à com.sybase.xml.xql.Xql 149
méthode java.sql, non supportée 226
méthode main(), exécutée sur le client 170
méthode query, com.sybase.xml.xql.Xql 150
méthode, spécifique à 151
méthode statique 50, 85, 87, 96
méthode XML
parse(InputStream xml_document) 150
query(String query, String xmlDoc) 150
méthode XQL 148
parse(String xmlDoc) 149
query(String query, InputStream xmlDoc) 151
query(String query, JXml jxml) 152
query(String query, SybXmlStream xmlDoc) 151
setParser 153
SybFileXmlStream 153
SybXmlStream 152
méthode, voir aussi méthode XQL
méthode, voir méthode Java
mise à jour d'un document XML 136
mise à jour des objets Java 30
mot-clé
de style Java 98
distinct 51
moteur de requêtes
besoins en mémoire 137
comme programme autonome 133
hors du serveur ou dans le serveur 133
MulticastSocket, classe Java 212

N

navigation XQL 137
nom de classe 234
nom de fichier, bookstore.xml 144
nom de package 234
nombre de sockets Java, paramètre de configuration 219
norme 5
norme ANSI 5
norme SQLJ 84

O

objet de base de données
xmlcol 135, 138
xmlimage 138
XMLTEXT 135
XQL, langage de requêtes générique pour XML 137
objet Java 30
Open Client CT-Library 131
Open Client DB-Library 131
opérateur
descendant 138
égal 138
fils 138
filtre 139
sous-script 139
union 51
opérateur filtre, utilisant une expression booléenne 140
opération agencement 51
opération de chaîne de caractères dans SQL 132
opération de données XML
serveur 162
opération égalité 51
opération Java, appelée depuis SQL 8
opération XML
client 162
option
external name 92
language java 92
parameter style java 92
saxparser 143
option de stockage
avantages et inconvénients 161
dans la ligne 28
option de stockage, avantages et inconvénients 161
order2Sql, méthode 166
OrderXml 155
classe 155
classe exemple 155
classe Java 155, 175
code source 155
constructeur appelé à partir du serveur 167
exemple d'application 155
sous-classe de la classe JXml 156
OrderXml, méthode Java 156

ordre de recherche
  type de fonction 93

P

paramètre
  aide 143
  called on null input 91
  déterministe 91, 98
  entrée 100
  external name 98
  in 100
  infile 143
  inout 100
  Java-SQL 53
  language java 98
  modifies sql data 91, 98
  non déterministe 98
  out 100
  outfile 143
  parameter style java 98
  qstring 143
  returns null on null input, clause Java 91
  size of global fixed heap (machine virtuelle Java) 213
  size of process object heap (machine virtuelle Java) 213
  size of shared class heap (machine virtuelle Java) 213
  sortie 100
  style java 98
  validation 143
paramètre de configuration, nombre de sockets Java 219
paramètre de jeu de résultat dynamique 98
paramètre de la machine virtuelle Java
  size of global fixed heap 213
  size of process object heap 213
  size of shared class heap 213
paramètre de langage Java 98
paramètre de mémoire, Java Services, table 137
parenthèses ()
  dans les instructions SQL xvii
parse()
  commande 136
  renvoie sybase.aseutils.SybXmlStream 136
parse(), méthode Java, commande 135
parse(InputStream xmlll_document), méthode XML 150
parse(String xmlDoc), méthode XQL 149
personnalisation des éléments 126
pilote JDBC 14, 227
  client 8, 65
  jConnect 8
  serveur 8, 65
  utilisé par OrderXml dans le serveur 165
présentation
  informations XML avec XSL 130
  instructions fournies dans XSL 124
PrintWriter, classe Java 219
procédure
  activation de java.net 212
  création d'une routine SQLJ 84
procédure stockée SQLJ 96–98, 107
  fonctionnalité 96
  modification de données SQL 98
  suppression 107
  utilisation de paramètres d'entrée et de sortie 100
  visualisation des informations 107
procédure système
  helpjava 21
  sp_configure 15
  sp_depends 107
  sp_help 107, 108
  sp_helpjava 108
  sp_helprotect 108
processus
  processus client, java.net 214
  processus serveur, java.net 214
processus serveur 214
programme autonome, com.sybase.xml.xql.XqlDriver 142
propriété d'affectation
  élément de données Java-SQL 36

Q

questions et réponses 6

R

recherche dans les documents XML stockés sur le Web 122
référence de colonnes 236
référence de membre 237
référencement
  champ 32
  externe d'une DTD XML 129
remplacement de méthode 239
réorganisation des classes installées 23
requête dans XML avec com.sybase.xml.xql.XqlDriver 142
requête, structure 140
réquisition d'une machine virtuelle Java 200
restriction de Java dans la base de données 11
ResultSet
  accès aux colonnes de documents stockés 189
  assemblage d'un document à partir de SQL 185
  classe 175
  classe Java 179
  classe JDBC 181
  comparaison quantifiée dans les documents stockés 192
  DTD 184
  génération dans Adaptive Server 186
  génération dans le client 185
  méthode de recherche 192
  quantificateur dans la clause where 194
  quantificateur dans la liste de sélection 194
  script serveur 191
  sélection et mise à jour des colonnes 191–192
  stockage de document dans une colonne SQL 188
  traduction dans le client 186
  type de document 180
  type de données mappable 109
ResultSetData 179
ResultSetMetaData 179
ResultSetMetaData pour le résultat de l'exemple 181
ResultSetMetaData, classe JDBC 183
ResultSetXml 175
  accès à XML 175
  code source 175
  code source, XML 175
  écriture de code Java pour accéder à XML 175
  répertoire, XML 175
  similaire à la classe OrderXml 175
  sous-ensemble de la classe JXml 175
  traitement des jeux de résultats SQL 175
ResultSetXml(String), méthode Java 176

S

SAX
  génération d'événements 132
  interface XML standard 132
  portable sur les analyseurs XML 132
  renvoi d'un objet DOM 132
SAX (Simple API for XML) 132
saxparser, option com.sybase.xml.xql.XqlDriver 143
sécurité
  routine SQLJ 85
sélection de données avec XQL 133
sélection des objets Java 30
sérialisation 235, 236
ServerSocket, classe Java 212, 214
serveur externe, écriture avec java.net 213
SGML (Standardized General Markup Language) 122
shared class heap 212
signature de méthode Java 92, 98
signature de méthode Java explicite 110
signature de méthode Java implicite 110
signe @ 91
signe plus (+) dans une déclaration de type de données XML 129
Simple API for XML (SAX) 132
Socket, classe Java 212
source d'entrée, spécification avec SAX 132
sous-type 40
sous-type de classe 40–42
sp_helpjava
  syntaxe 21
  utilitysp_helpjava 21
spécification du jeu de caractères 125
SQL
  encapsulation 83, 88
  expression, objet Java inclus 8
  opération de chaîne de caractères 132
  signature de fonction 90
  signature de procédure 97
  table, création et remplissage 162

Standardized General Markup Language (SGML) 122
static void createOrderTable, méthode Java 157
stockage
  document 162
  élément 162
  hybride 162
stockage d'éléments 160, 162, 164, 167, 184–188
  extraction d'éléments et stockage 162
stockage de documents 161, 162, 168, 173, 188–195
  à partir du client et du serveur 168
  document entier 162
  table 164
stockage des documents XML 160
stockage hybride 161, 162, 173–174
  stockage d'un document dans une colonne XQL 162
  utilisation de la technique 173
stockage, documents 168
stockage, hybride 162
string getColumn, méthode Java 177
string getItemElement, méthode Java 158
string toSqlScript, méthode Java 177
structure, requête 140
supertype 40
suppression 30, 107
  document XML 136
  objet Java 30
suppression de classe 22
suppression de fichier JAR 22
surcharge de méthode 111
Sybase Central
  création d'une procédure ou fonction SQLJ 89
  gestion de procédures et fonctions SQLJ 89
  visualisation des propriétés d'une routine SQLJ 90
sybase.asciutils 152
sybase.aseutils.SybXmlStream, renvoyé par la commande parse() 136
SybXmlStream
  variable 149

T

table de stockage, de documents et hybride 164
table orders 180
  utilisation 184
tableau Java 100
technique du stockage hybride, utilisation 173
téléchargement
  classe installée 22
  JAR installé 22
texte, type de données 135, 136
toSqlScript(), méthode Java 186
traitement
  effet de la clause where 140
traitement avec SAX de manière incrémentielle 132
traitement XML spécialisé 155
Transact-SQL
  commande, dans méthode Java 228
type de document XML
  Order (Commande) 179
  ResultSet 179
type de données
  appel de méthode 239
  char 124
  classe Java 3
  conversion 223
  de compilation 42
  image 124, 135, 136
  Java primitif 93
  mappable ADT 108
  mappable Objet 108
  mappable Sortie 109
  mappage 232–??
  pour une colonne, conventions 27
  simplement mappable 108
  text 124
  texte 135, 136
  varchar 124
  version d'exécution 42
type de données de classe Java 93
type de données de compilation 42
type de données Java
  mappable ADT 108
  mappable Jeu de résultats 109
  mappable Objet 108
  mappable Sortie 109
  simplement mappable 108
type de données, mappage ??–232

U

Unicode 47
URL
  classe Java 214
URLConnection, classe Java 212
URLDecoder, classe Java 212
URLEncoder, classe Java 212
UTF8, jeu de caractères par défaut 125, 131
utilisation
  classe Java 25, 58
  classe Socket 213
  classe URL 214
  com.sybase.xml.xql.XqlDriver 142
  conjointe de Java et de SQL 8
  table orders 184
  technique du stockage hybride 173
utilitaire
  extractjava 22
  installjava 14, 17, 135
  installjava, option -f 18
  installjava, option -j 19
  installjava, option -new 19
  installjava, option update 20
  installjava, syntaxe 18

V

valeur de jeu de caractères par défaut, UTF8 131
valeur NULL
  dans fonction SQLJ 93
  instruction case 95
valeur NULL dans Java-SQL 43–46
  argument des méthodes 45
  utilisation de la fonction convert 46
variable 235
  Java-SQL 53
  statique 53
  SybXmlStream 149
  type de données 28
  valeur affectée 31
variable HTML
  ... 128
  <table>...</table>, structure 128
  bcolor, couleur 128
  CustomerID 127, 128
  CustomerName 128
  données 128
  ItemID 126
  ItemName 126
  Quantities 128
  Quantity 126
  séquence 126
  unités 126
variable statique 53
variable XML
  balise 125
  OrderXML 122
  XMLResultSet 122
version d'exécution
  type de données 42
vidage explicite des données 219
virgule (,) dans les instruction SQL xvii
visualisation des informations
  sur les classes installées 21
  sur les JAR installés 21
void appendItem, méthode Java 159
void deleteItem(int itemNumber), méthode Java 159
void order2Sql(String ordersTableName, String server), méthode Java 157
void setColumn, méthode Java 178
void setItemElement, méthode Java 158
void setOrderElement, méthode Java 157

W

Web, stockage de documents XML 145
writetext, commande 131

X

xerces.jar
  répertoire 135
XML 121
  accès aux documents avec java.net 211
  adapté pour l'échange de données 122
  analyseur 132
  analyseur, hors du serveur ou dans le serveur 133
  balise personnalisée 123
  code exemple DTD, imbrication 129

code exemple DTD, référencement externe 129
code source des classes exemples 122
comparaison avec le HTML 122
comparaison avec les langages SGML et HTML 123
complément d'information 123
conversion avec XSL 130
déclaration de jeu de caractères 131
déclaration de type de données (DTD) 126
document exemple 124
document source à partir de XSL 130
document Web pour des informations détaillées 123
DTD non obligatoire dans tous les documents 130
DTD, instruction 129, 130
élément DTD, restrictions 129, 130
exemple personnalisable 179
Extensible Markup Language 121
interprétable par les navigateurs et les processeurs HTML 123
jeu de caractères, client et serveur 131
option de stockage, avantages et inconvénients 161
outil écrit en Java 122
présentation 123
sous-ensemble du langage SGML 122
structure syntaxique stricte 123
traitement spécialisé 155
type de document spécifique d'une application 123
XML Query Language (XQL) 122
XML, analyseur 132
xml.zip, répertoire 135
xmlcol, objet de base de données 135, 138
xmlimage, objet de base de données 138
XMLTEXT, objet de base de données 135
XQL
à partir de zéro 144
affichage sous forme de document XML 133
analyse et requête 218
client JDBC 145
développement d'une application autonome 145
EJB 145
installation dans Adaptive Server 135
interface, com.sybase.xml.xql.store.SybMemXmlStream 153
JavaBeans 145
langage de requête reposant sur le chemin 122, 137
méthode d'analyse 218
méthode d'interrogation 218
navigation 137
opérateur 137
recherche dans les documents XML 122
système de numérotation 144
XQL (XML Query Language) 122
XSL 130
conversion XML 130
Extensible Style Language 124
informations de présentation XML 130
spécifications 130
utilisation avec les applications de présentation 130