新機能ガイド
Open Server™ 15.7 および SDK 15.7 Windows、Linux、および UNIX 版

ドキュメント ID：DC00071-01-1570-01
改訂：2011 年 9 月 9 日

製品のプラットフォームと互換性

表 1 に、Open Server™ と SDK をサポートするプラットフォームを示します。

表 1: Open Server および SDK をサポートするプラットフォーム

プラットフォーム
HP-UX Itanium 32 ビット版
HP-UX Itanium 64 ビット版
IBM AIX 32 ビット
IBM AIX 64 ビット
Linux x86 32 ビット版
Linux x86-64 64 ビット版
Linux on POWER 32 ビット版
Linux on POWER 64 ビット版
Microsoft Windows x86 32 ビット版
Microsoft Windows x86-64 64 ビット版
Sun Solaris SPARC 32 ビット版
Sun Solaris SPARC 64 ビット版
Sun Solaris x86 32 ビット版
Sun Solaris x86-64 64 ビット版

注意 上記のプラットフォームで Open Server と SDK のすべてのコンポーネントが使用できるとは限りません。各プラットフォームで使用できるコンポーネントについては、「製品のコンポーネント」(6 ページ) を参照してください。

表 2 に、Open Server および SDK 製品が構築およびテストされているプラットフォーム、コンパイラ、サードパーティ製品を示します。

表 2: Open Server と SDK のプラットフォーム互換性の一覧

プラットフォーム	オペレーティング・システム・レベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos バージョン	LDAP (Light-weight Directory Access)	Secure Sockets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
HP-UX Itanium 32 ビット版	HP 11.31	HP ANSI C A.06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2	5.14 (DBI 1.616)	該当なし	該当なし
HP-UX Itanium 64 ビット	HP 11.31	HP ANSI C A.06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2	該当なし	5.3.6	2.6 (DBAPI 2.0)

プラットフォーム	オペレーティング・システム・レベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos バージョン	LDAP (Lightweight Directory Access)	Secure Sockets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
IBM AIX 32 ビット	AIX 6.1	XL C 10.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.1	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.2)	5.14 (DBI 1.616)	該当なし	該当なし
IBM AIX 64 ビット	AIX 6.1	XL C 10.1	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.2)	該当なし	5.3.6	2.6 (DBAPI 2.0)
Linux x86 32 ビット版	Red Hat Enterprise Linux 5.3	gcc 4.1.2 20060404 kernel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2	該当なし	該当なし	該当なし
Linux x86-64 64 ビット版	Red Hat Enterprise Linux 5.3 (Nahant Update 4)	gcc 4.1.2 20060404 kernel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2	5.14 (DBI 1.616)	5.3.6	2.6 (DBAPI 2.0)
Linux on POWER 32 ビット版	Red Hat Enterprise Linux 5.3	XL C 10.1	サポート予定なし	MIT 1.4.1	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	CSI-OpenSSL 0.9.8l CSI-Crypto 2.6.3	5.14 (DBI 1.616)	該当なし	該当なし
Linux on POWER 64 ビット版	Red Hat Enterprise Linux 5.3	XL C 10.1	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	CSI-OpenSSL 0.9.8l CSI-Crypto 2.6.3	該当なし	5.3.6	2.6 (DBAPI 2.0)

プラットフォーム	オペレーティング・システム・レベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos バージョン	LDAP (Light-weight Directory Access)	Secure Sockets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
Microsoft Windows x86 32 ビット版	Windows 2008 R2 Service Pack 1 Windows XP Service Pack 1 (ODBC/ OLE DB のみ)	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 4.0、MIT 2.6.4	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0)	該当なし	該当なし	該当なし
Microsoft Windows x86 64 ビット	Windows 2008 R2 Service Pack 1 Windows XP Service Pack 1 (ODBC/ OLE DB のみ)	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 2.1	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	CSI-OpenSSL 0.9.8l CSI-Crypto 2.6.3 M2	該当なし	該当なし	2.6 (DBAPI 2.0)
Sun Solaris SPARC 32 ビット版	Solaris 10	Sun Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.2	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.2)	5.14 (DBI 1.616)	該当なし	該当なし
Sun Solaris SPARC 64 ビット	Solaris 10	Sun Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.2	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.2)	該当なし	5.3.6	2.6 (DBAPI 2.0)
Sun Solaris x86 32 ビット版	Solaris 10	Sun Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2	5.14 (DBI 1.616)	該当なし	該当なし
Sun Solaris x86-64 64 ビット	Solaris 10	Sun Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.16 (OpenSSL 0.9.8l を含む)	Certicom SSL Plus 5.2.2	該当なし	5.3.6	2.6 (DBAPI 2.0)

凡例：該当なし = そのプラットフォーム版でスクリプトが使用できない、または SDK と連動しない。

注意 最新の Open Server および SDK の動作確認サポートについては、Sybase Platform Certification サイトを参照してください。(http://certification.sybase.com/ucr/search.do)

製品のコンポーネント

Open Server と SDK 15.7 には、Bulk-Library の select into ロギング、ラージ・オブジェクトのストアド・プロシージャ・パラメータのサポート、Adaptive Server® での非実体化カラムのサポート、jConnect™ for JDBC™ と Adaptive Server のドライバおよびプロバイダに対するアップデートなどの新機能が導入されています。Open Server と SDK 15.7 は、Adaptive Server で使用するための Perl、PHP、Python のスクリプト言語もサポートしています。新機能の全リストについては、このマニュアルの最初の目次を参照してください。

15.7 でリリースされた製品は次のとおりです。

- Open Server
- Software Developer's Kit
- SDK DB-Library Kerberos Authentication Option

Open Server

Open Server は、Open Client™ または jConnect for JDBC ルーチンを通じて送信されたクライアント要求に応答するカスタム・サーバを作成するために使用できる API とサポート・ツールのセットです。表 3 に、Open Server のコンポーネントとそのコンポーネントをサポートしているプラットフォームを示します。

表 3: Open Server のコンポーネントとサポートされるプラットフォーム

Open Server のコンポーネント	プラットフォーム
Open Server Server-Library	すべてのプラットフォーム
Open Server Client-Library	すべてのプラットフォーム
言語モジュール	すべてのプラットフォーム

Software Developer's Kit

Software Developer's Kit (SDK) は、クライアント・アプリケーションの開発に使用できるライブラリとユーティリティのセットです。表 4 に、SDK のコンポーネントとそのコンポーネントがサポートされるプラットフォームを示します。

表 4: SDK のコンポーネントとサポートされるプラットフォーム

SDK のコンポーネント	プラットフォーム
Open Client Client-Library	すべてのプラットフォーム
Open Client DB-Library™	すべてのプラットフォーム
Embedded SQL™/C (ESQL/C)	すべてのプラットフォーム
Embedded SQL/COBOL (ESQL/COBOL)	• HP HP-UX Itanium 32 ビット版 • HP HP-UX Itanium64 ビット版 IBM AIX 64 ビット • Linux x86 32 ビット版 • Linux x86-64 64 ビット版 • Linux on POWER 32 ビット版 • Linux on POWER 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Sun Solaris SPARC 32 ビット版 • Sun Solaris SPARC 64 ビット版 • Sun Solaris x86 32 ビット版 Sun Solaris x86-64 64 ビット版
Extended Architecture (XA)	• HP HP-UX Itanium 32 ビット版 • HP HP-UX Itanium64 ビット版 • IBM AIX 32 ビット • IBM AIX 64 ビット • Linux x86-64 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Sun Solaris SPARC 32 ビット版 • Sun Solaris SPARC 64 ビット版 • Sun Solaris x86 32 ビット版 • Sun Solaris x86-64 64 ビット版
jConnect for JDBC	すべてのプラットフォーム

SDK のコンポーネント	プラットフォーム
Sybase 製 Adaptive Server Enterprise ODBC ドライバ	• HP HP-UX Itanium64 ビット版 • IBM AIX 64 ビット • Linux on POWER 64 ビット版 • Linux x86 32 ビット版 • Linux x86-64 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Sun Solaris SPARC 64 ビット版 • Sun Solaris x86-64 64 ビット版
Sybase 製 Adaptive Server Enterprise OLE DB プロバイダ	• Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版
Adaptive Server Enterprise ADO.NET Data Provider	• Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版
言語モジュール	すべてのプラットフォーム
Adaptive Server Enterprise (拡張モジュール Python 版)	• HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット • Linux x86-64 64 ビット版 • Linux on POWER 64 ビット版 • Microsoft Windows x86-64 64 ビット版 • Sun Solaris SPARC 64 ビット版 • Sun Solaris x86-64 64 ビット版
Adaptive Server Enterprise (拡張モジュール PHP 版)	• HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット • Linux x86-64 64 ビット版 • Linux on POWER 64 ビット版 • Sun Solaris SPARC 64 ビット版 • Sun Solaris x86-64 64 ビット版
Perl 用 Adaptive Server Enterprise データベース・ドライバ	• HP-UX Itanium 32 ビット版 • IBM AIX 32 ビット • Linux x86-64 64 ビット版 • Linux on POWER 32 ビット版 • Sun Solaris SPARC 32 ビット版 • Sun Solaris x86 32 ビット版

SDK DB-Library Kerberos Authentication Option

Sybase SDK DB-Library™ Kerberos Authentication Option では、MIT Kerberos セキュリティ・メカニズムを DB-Library で使用できます。これは、次のプラットフォームで使用できます。

- Linux x86 32 ビット版
- Microsoft Windows x86 32 ビット版
- Solaris SPARC 32 ビット版
- Solaris SPARC 64 ビット版

Open Client 15.7 と Open Server 15.7 の機能

この項では、Open Client 15.7 と Open Server 15.7 の新機能ついて説明します。

ラージ・オブジェクトのロケータのサポート

Open Client および Open Server バージョン 15.7 は、ラージ・オブジェクト (LOB) のロケータをサポートしています。LOB ロケータには、データ自体ではなく、Adaptive Server 内の LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。

Adaptive Server 15.7 には、LOB ロケータを使用して LOB データを操作するための Transact-SQL® コマンドおよび関数が含まれています。これらのコマンドや関数は、Client-Library からの言語コマンドとして呼び出すことができます。『ASE Transact-SQL ユーザーズ・ガイド』の「第 21 章 ロー内／ロー外の LOB」を参照してください。

Client-Library の変更

CS_LOCATOR データ型は LOB ロケータをサポートしています。cs_locator_alloc() および cs_locator_drop() API は、CS_LOCATOR 変数のメモリをそれぞれ割り付けおよび割り付け解除します。CS_LOCATOR 変数からの情報を取得するために、cs_locator() が追加されています。

Client-Library ルーチンである cs_convert() と ct_bind() は、CS_LOCATOR 変数を処理できるように強化されています。

CS_LOCATOR

CS_LOCATOR は、ロケータの値とオプションのプリフェッチされたデータを格納する opaque データ型です。受信ロケータを CS_LOCATOR 変数にバインドする前に、cs_locator_alloc() を使用してこの変数のメモリを割り付けてください。そうしないとエラーが発生します。変数が無用になった場合は、cs_locator_drop() を使用してそのメモリを解放します。

CS_LOCATOR 変数は再使用できますが、Adaptive Server 内の現在のロケータ値はトランザクションが終了すると無効になります。

CS_LOCATOR の型定数は次のとおりです。

- CS_TEXTLOCATOR_TYPE − text LOB 用
- CS_IMAGELOCATOR_TYPE − image LOB 用
- CS_UNITEXTLOCATOR_TYPE − unitext LOB 用

ロケータのプリフェッチされたデータやロケータ値の文字表現を CS_LOCATOR 変数から取得するには、cs_convert() を使用します。CS_LOCATOR を CS_CHAR に変換すると、ロケータの 16 進数値が文字列として返されます。ロケータを CS_TEXT_TYPE、CS_IMAGE_TYPE、または CS_UNITEXT_TYPE に変換すると、ロケータのプリフェッチされたデータが返されます。

表 5: サポートされている LOB ロケータの変換

	CS_TEXT_ LOCATOR	CS_IMAGE_ LOCATOR	CS_UNITEXT_ LOCATOR
CS_CHAR_TYPE	X	X	X
CS_TEXT_TYPE	X		
CS_IMAGE_TYPE		X	
CS_UNITEXT_TYPE			X
CS_TEXT_LOCATOR	X		
CS_IMAGE_LOCATOR		X	
CS_UNITEXT_LOCATOR			X

凡例：X = サポートされている変換

ロケータのデータ型を使用するときは、次の内容が適用されます。

- ct_bind() は CS_DATAFMT の *maxlength* 値を無視します。これは、Client-Library がロケータのデータ型の長さを固定と見なすためです。ロケータを使用して送信される、オプションのプリフェッチされたデータに必要なメモリは、その長さ全体に対して内部で割り付けられます。*maxlength* の値がプリフェッチされたデータの長さに影響することはありません。
- 受信 LOB ロケータは CS_CHAR_TYPE にバインドできます。ただし、ロケータを CS_TEXT_TYPE、CS_IMAGE_TYPE、または CS_UNITEXT_TYPE に直接バインドすることはできません。

cs_locator()

プリフェッチされたデータ、サーバ内の LOB の全長、ロケータのポインタの文字表現などの情報を CS_LOCATOR 変数から取得します。

構文

```
CS_RETCODE cs_locator(ctx, action, locator, type, buffer, buflen,
  outlen)

CS_CONTEXT      *ctx;
CS_INT           action;
CS_LOCATOR      *locator;
CS_INT           type;
CS_VOID         *buffer;
CS_INT          buflen;
CS_INT           *outlen;
```

パラメータ

- *ctx* − CS_CONTEXT 構造体を指すポインタ。
- *action* − 情報を設定するのか取得するのかを指定します。現時点で実行可能な唯一のアクションは CS_GET です。
- *locator* − ロケータ変数を指すポインタ。
- *type* − 取得または設定する情報の種類。記号値は次のとおりです。

値	対処法	*buffer が指す対象	説明
CS_LCTR_LOBLEN	CS_GET	CS_BIGINT	サーバ内の LOB データの全長を取得する。
CS_LCTR_LOCATOR	CS_GET	CS_CHAR	ロケータ値を文字列として取得する。
CS_LCTR_PREFETCHLEN	CS_GET	CS_INT	ロケータ変数に含まれている、プリフェッチされた LOB データの長さを取得する。
CS_LCTR_PREFETCHDATA	CS_GET	CS_CHAR	ロケータ変数に含まれている、プリフェッチされた LOB データを取得する。
CS_LCTR_DATATYPE	CS_GET	CS_INT	ロケータの型を取得する。有効な戻り値の型は、CS_TEXTLOCATOR_TYPE、CS_IMAGELOCATOR_TYPE、CS_UNITEXTLOCATOR_TYPE。

- *buffer* − データの格納先変数を指すポインタ。文字データは NULL で終了します。
- *buflen* − **buffer* のバイト単位の長さ。
- *outlen* − CS_INT 変数を指すポインタ。*outlen* が NULL 以外の場合、cs_locator() は **outlen* を、**buffer* に配置されたデータのバイト単位の長さに設定します。返されたデータが文字データ (プリフェッチされたデータやロケータ文字列など) の場合、**outlen* に返される長さには、NULL ターミネータが含まれます。cs_locator() が CS_TRUNCATED を返し、*outlen* が NULL でない場合、cs_locator() は必要なバッファ・サイズを **outlen* に返します。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_TRUNCATED	バッファが小さすぎるために結果がトランケートされている。
CS_FAIL	ルーチンが失敗。

cs_locator_alloc()

CS_LOCATOR データ型構造体を割り付けます。

構文

```
CS_RETCODE cs_locator_alloc(ctx, locator)

CS_CONTEXT *ctx;
CS_LOCATOR **locator;
```

パラメータ

- *ctx* − CS_CONTEXT 構造体を指すポインタ。
- *locator* − 割り付けられるロケータ変数のアドレス。**locator* を、新たに割り付けられた CS_LOCATOR 構造体のアドレスに設定します。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

cs_locator_drop()

CS_LOCATOR データ型構造体の割り付けを解除します。

構文

```
CS_RETCODE cs_locator_drop(ctx, locator)

CS_CONTEXT *ctx;
CS_LOCATOR *locator;
```

パラメータ

- *ctx* − CS_CONTEXT 構造体を指すポインタ。
- *locator* − 割り付け解除されるロケータ変数を指すポインタ。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗。

isql 拡張機能

isql は、LOB ロケータ値を 16 進数文字の形式で表示します。CS_LOCATOR に格納されている、プリフェッチされたデータは表示されません。

例 LOB データをロケータに変換し、ロケータの値を表示します。

```
1> set send_locator on
2> go

1> select * from testable
```

```
2> go
charcol         textcol
--------------- --------------------------------------------------
Hello           0x48656c6c6f20576f726c642e204865726520492061d2e2e
```

ラージ・オブジェクトのロケータに対する Open Server のサポート

LOB ロケータの機能が Server-Library に追加されているので、Open Server アプリケーションは LOB ロケータの言語コマンドをクライアントからバックエンド・サーバに渡すことができます。LOB ロケータをサーバからクライアント・アプリケーションに渡すために、Open Server アプリケーションでは CS_LOCATOR 変数のメモリを割り付け、LOB 情報をバインドしてサーバから受け取ります。

srv_bind() と srv_descfmt() は、CS_TEXT_LOCATOR_TYPE、CS_IMAGE_LOCATOR_TYPE、CS_UNITEXT_LOCATOR_TYPE を処理できるように強化されています。

ラージ・オブジェクトのロケータのサポート

次の接続機能は、LOB ロケータの送受信のサポートを示します。

- CS_DATA_LOBLOCATOR − クライアント・アプリケーションが CS_VERSION_157 によって初期化されたときに暗黙的に設定される読み取り専用要求機能です。Client-Library が LOB ロケータをサーバに送信できることを示します。
- CS_DATA_NOLOBLOCATOR − クライアント・アプリケーションが設定する応答機能。基本となる Client-Library によってサポートされている場合でも、LOB ロケータを送信しないようにサーバに伝えます。

サーバからの LOB ロケータの要求

デフォルトでは、LOB のカラムや値を選択すると、Adaptive Server ではネゴシエートされた LOB ロケータがサポートされているかどうかにかかわりなく、LOB ロケータの代わりに LOB データを送信します。明示的に LOB ロケータを要求するか、プリフェッチされたデータを要求するには、ct_options() を使用して次のクエリ処理オプションを設定します。

- CS_OPT_LOBLOCATOR − CS_TRUE に設定されている場合に、LOB 値ではなくロケータを返すようにサーバに要求するブール値。このオプションは、クエリをサーバに送信する前に設定します。デフォルトは CS_FALSE です。
- CS_OPT_LOBPREFETCHSIZE − サーバが送信する必要のある、プリフェッチされたデータのサイズを指定する整数。image ロケータの場合、このサイズはプリフェッチされたデータのバイト数を示し、text および unitext ロケータの場合は文字数を示します。

CS_OPT_LOBPREFETCHSIZE のデフォルト値は 0 で、これはプリフェッチされたデータを送信しないようにサーバに指示します。-1 の値は、要求された LOB の LOB データ全体をそのロケータと共に取得します。

ロケータの値とオプションのプリフェッチされたデータは CS_LOCATOR データ型に格納されます。クライアントは CS_LOCATOR 変数のメモリを割り付けてから、ロケータ・データを要求する必要があります。

例 トランケートする必要のあるテキスト値の LOB ロケータを取得します。さらに別のコード例については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

```
CS_LOCATOR *lobloc;
CS_INT     prefetchsize;
CS_BOOL    boolval;
CS_INT     start, length;
CS_INT     outlen;
CS_CHAR    charbuf[1024];
CS_BIGINT  totallen;
...

/*
** Turn on option CS_LOBLOCATOR first and set the prefetchsize to 100.
*/

boolval = CS_TRUE;
ct_options(conn, CS_SET, CS_OPT_LOBLOCATOR, &boolval, CS_UNUSED, NULL);
prefetchsize = 100;
ct_options(conn, CS_SET, CS_OPT_LOBPREFETCHSIZE, &prefetchsize, CS_UNUSED,
   NULL);

/*
** Allocate memory for the CS_LOCATOR.
*/
cs_locator_alloc(ctx, &lobloc);

/*
** Open a transaction and get the locator.The locator is only valid within a
** transaction.
*/
sprintf(cmdbuf, "begin transaction \
   select au_id, copy from pubs2..blurbs where au_id \
   like '486-29-%'");
ct_command(cmd, CS_LANG_CMD, cmdbuf , CS_NULLTERM, CS_UNUSED);
ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
{
   ...
```

```
}
   /*
   ** Bind the locator and fetch it.
   */
   strcpy(prmfmt.name, "@locatorparam");
   prmfmt.namelen = CS_NULLTERM;
   prmfmt.datatype = CS_TEXTLOCATOR_TYPE;
   prmfmt.maxlength = CS_UNUSED;
   ...

   ct_bind(cmd, 1, &fmt, lobloc, NULL, &indicator);
   ct_fetch(cmd, CS_UNUSED, CS_UNUSED, CS_UNUSED, &count);
}

/*
** Use the cs_locator() routine to retrieve data from the fetched locator.
** Get the prefetch length and the prefetch data.
*/
cs_locator(ctx, CS_GET, lobloc, CS_LCTR_PREFETCHLEN, (CS_VOID *)&prefetchsize,
   sizeof(CS_INT), &outlen);

cs_locator(ctx, CS_GET, lobloc, CS_LCTR_PREFETCHDATA, (CS_VOID *)charbuf,
   sizeof(charbuf), &outlen);

/*
** Retrieve the total length of the LOB data in the server for this
** locator.
*/
cs_locator(ctx, CS_GET, lobloc, CS_LCTR_LOBLEN,(CS_VOID *)&totallen,
   sizeof(totallen), &outlen);

/*
** Use the retrieved locator to perform an action to the LOB, pointed to by
** this locator in the server.
**
** Get a substring from the text in the server, using a parameterized language
** command.
*/
start = 10;
length = 20;
sprintf(cmdbuf, "select return_lob(text, substring(@locatorparam, \
   start, length))");
ct_command(cmd, CS_LANG_CMD, cmdbuf, CS_NULLTERM, CS_UNUSED);

/*
** Set the format structure and call ct_param()
*/
strcpy(prmfmt.name, "@locatorparam");
prmfmt.namelen = CS_NULLTERM;
prmfmt.datatype = CS_TEXTLOCATOR_TYPE;
prmfmt.format = CS_FMT_UNUSED;
```

```
prmfmt.maxlength = CS_UNUSED;
prmfmt.status = CS_INPUTVALUE;

indicator = 0;
ct_param(cmd, &prmfmt, (CS_VOID *)lobloc, CS_UNUSED, indicator);

/*
** Send the locator commands to the server.
*/
ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
{
   ...
}

/*
** Truncate the text to 20 bytes and commit the transaction.
*/
sprintf(cmdbuf, "truncate lob @locatorparam (length) \
   commit transaction");
ct_command(cmd, CS_LANG_CMD, cmdbuf, CS_NULLTERM, CS_UNUSED);
ct_param(cmd, &prmfmt, (CS_VOID *)lobloc, CS_UNUSED, indicator);

ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
{
   ...
}



/*
** The transaction is closed, deallocate the locator.
*/
cs_locator_drop(ctx, lobloc);
```

ロー内とロー外の LOB のサポート

Bulk-Library バージョン 15.7 では、Adaptive Server の text、image、unitext のラージ・オブジェクト (LOB) カラムをロー内に格納することをサポートしています。

Adaptive Server 15.7 では、ロー内に格納するようにマークされている LOB カラムは、ローに十分な領域が残っている場合にロー内に格納されます。ロー内に書き込むことができるのは、バインドされている LOB データのみです。bcp ユーティリティは LOB データをバインドするので、該当するロー内の LOB データを送信します。『ASE Transact-SQL ユーザーズ・ガイド』の「第 21 章 ロー内／ロー外の LOB」を参照してください。

Bulk-Library の *select into* ロギング

ローをプロキシ・テーブルに挿入する select into existing table 文を処理するために、ASE は Bulk-Library を使用してバルク・コピー・オペレーションを生成します。ただし、完全なロギングを通常のバルク・コピー・オペレーションに使用することはできません。BLK_CUSTOM_CLAUSE プロパティは、ASE が通常のバルク・コピー・オペレーションと、insert into 文から発生し、プロキシ・テーブルに影響するバルク・コピー・オペレーションとを区別できるようにします。このような insert into 文から発生するバルク・コピー・オペレーションは、BLK_CUSTOM_CLAUSE プロパティによって指定されるカスタム句に追加できます。ASE はこの句を検出し、完全なロギングを実行できます。

BLK_CUSTOM_CLAUSE

アプリケーションは、blk_props Bulk-Library ルーチンを使用して、BLK_CUSTOM_CLAUSE を設定または取得できます。

表 6: Client/Server BLK_CUSTOM_CLAUSE プロパティ

プロパティ名	説明	*buffer の値	適用対象	注意
BLK_CUSTOM_CLAUSE	insert bulk コマンドの既存の with 句の後に追加するアプリケーション固有のカスタム SQL 句。	カスタム句を含む文字列。	コピーインのみ	カスタム SQL 句をサポートするサーバ・バージョンのみによってサポートされます。現時点では内部の製品のみによって使用されています。

- select into オペレーションは、ASE の select into/bulkcopy/pllsort データベース・オプションがオンに設定されている場合にのみ許可されます。
- select into オペレーションを完全にロギングするには、ASE の full logging for select into データベース・オプションをオンに設定する必要があります。

例

BLK_CUSTOM_CLAUSE は blk_props によって設定されます。

```
blk_props(blkdesc, CS_SET, BLK_CUSTOM_CLAUSE,
(CS_VOID *)"from select_into", CS_NULLTERM, NULL);
```

ASE は、カスタム句の指定されたバルク・コピー・オペレーションを生成します。

```
insert bulk mydb.mytable with nodescribe from select_into
```

ここで、mydb と mytable は、影響を受けるデータベースとテーブルです。

Bulk-Library と *bcp* による非実体化カラムの処理

Bulk-Library は、Adaptive Server 15.7 で非実体化カラムを処理できるように強化されています。この機能強化により、非実体化カラムを含む、変更済みの Adaptive Server テーブルに対してデータのバルク・コピー・インを実行する場合は、Bulk-Library および bcp のバージョン 15.7 以降のみを使用できます。それより前のバージョンの bcp を使用して非実体化カラムへのデータのバルク・コピー・インを実行した場合は、Adaptive Server によってエラーが発生します。

オーバフロー・エラー

整数オーバーフローを発生させる DB-Library ルーチンを使用すると、次のエラーが発生します。

```
302 = SYBEINTOVFL, "DB-LIBRARY internal error: The arithmetic
operation results in integer overflow."
```

オーバーフローを発生させる dbcursoropen DB-Library ルーチンの scrollopt および nrows パラメータを掛け合わせると、次のエラーが発生します。

```
301 = SYBCOPNOV, "dbcursoropen(): The multiplication of
scrollopt and nrows results in overflow."
```

アプリケーション名の検出

Open Client および Open Server がどのように名前のないアプリケーションを処理するかを判断するには、CS_USE_DISCOVERED_APPNAME を *ocs.cfg* ランタイム設定ファイルに設定します。

表 7: CS_USE_DISCOVERED_APPNAME 設定ファイル・キーワード

キーワード	読み込まれるルーチン	有効値
CS_USE_DISCOVERED_APPNAME	ct_config、 ct_con_props	CS_TRUE または CS_FALSE。

CS_USE_DISCOVERED_APPNAME が CS_TRUE に設定されている場合、CS_APPNAME 値を持たない Open Client アプリケーションの実行可能ファイルの名前はオペレーティング・システムから取得 (検出) できます。

実行可能ファイル名は、30 文字に制限されています。30 文字を超える名前が検出された場合は、30 文字にトランケートされます。

TCP ソケット・バッファ・サイズの設定

TCP 入出力バッファのサイズは、Open Client および Open Server のコンテキスト・プロパティまたは接続プロパティとサーバ・プロパティを使用して設定できます。Open Client および Open Server アプリケーションは、これらのプロパティ設定を使用して、オペレーティング・システムの setsockopt コマンドでバッファ・サイズを設定します。setsockopt は TCP の connect および accept コマンドの前に呼び出す必要があるので、これらの Open Client および Open Server プロパティを設定してから接続を確立する必要があります。

プロパティ

TCP 入力および出力バッファ・サイズを設定するためのコンテキスト・プロパティおよび接続プロパティは、CS_TCP_RCVBUF と CS_TCP_SND です。

表 8: バッファ・サイズ設定用の Client-Library プロパティ

プロパティ	意味	*buffer の値	レベル
CS_TCP_RCVBUF	クライアント・アプリケーションの入力バッファのサイズ。	正の整数値。	コンテキスト、接続。
CS_TCP_SND	クライアント・アプリケーションの出力バッファのサイズ。	正の整数値。	コンテキスト、接続。

コンテキスト例

```
ct_config(*context, CS_SET, CS_TCP_RCVBUF, &bufsize,
CS_UNUSED, NULL);
```

接続例

```
ct_con_props(*connection, CS_SET, CS_TCP_RCVBUF, &bufsize,
CS_UNUSED, NULL);
```

TCP 入力および出力バッファ・サイズを設定するためのサーバ・プロパティは、SRV_S_TCP_RCVBUF と SRV_S_TCP_SNDBUF です。

表 9: バッファ・サイズ設定用のサーバ・プロパティ

プロパティ	設定／クリア	取得	cmd が CS_SET のときの bufp	cmd が CS_GET のときの bufp
SRV_S_TCP_RCVBUF	許可される。	許可される。	CS_INT	CS_INT
SRV_S_TCP_SNDBUF	許可される。	許可される。	CS_INT	CS_INT

サーバ例

```
srv_props(cp, CS_SET, SRV_S_TCP_SNDBUF, bufp,
CS_SIZEOF(CS_INT), (CS_INT *)NULL);
```

- これらのパラメータをアプリケーションに応じて設定します。たとえば、クライアントが大量のデータをサーバに送信することが想定される場合は、CS_TCP_SND と SRV_S_TCP_RCVBUF を大きな値に設定し、対応するバッファ・サイズを増加します。
- これらのプロパティのいずれかが CS_TCP_MAXBUF に設定されている場合、ソケットのバッファ・サイズはオペレーティング・システムの最大許容サイズに設定します。

すべての 64 ビット製品用 *isql64* および *bcp64*

64 ビット・バージョンの isql および bcp (isql64 および bcp64) は、Open Client と Open Server によってサポートされているすべての UNIX および Windows プラットフォームで使用できます。

Open Server および SDK 15.5 ESD #9 より前のバージョンでは、64 ビットの Windows で 64 ビットの isql.exe と bcp.exe のみを使用できます。isql.exe または bcp.exe を参照するスクリプトがあり、64 ビット・バージョンを使用する場合は、スクリプト内の参照を isql64.exe または bcp64.exe に変更する必要があります。

拡張された可変長ローのサポート

Adaptive Server 15.7 では、データオンリーロック (DOL) ローの可変長カラムの最大オフセットは 32767 バイトに拡張されており、8K を超える論理ページ・サイズが設定されている Adaptive Server が長い可変長の DOL ローをサポートできるようになっています。

Adaptive Server の論理ページに移植するために使用される Open Client と Open Server の Bulk-Library 15.7 ルーチンは、拡張された DOL ローをサポートしています。この機能は Bulk-Library 15.7 以降では自動的にアクティブ化されますが、Adaptive Server では有効にする必要があります。

長い DOL ローを使用するように設定されているデータベースは、Bulk-Library 15.5 以前を使用しているアプリケーションから送信された DOL ローを受け入れることができます。ただし、Bulk-Library 15.7 を使用するアプリケーションが長い DOL ローを Adaptive Server 15.5 以前、または古い形式の DOL ローを予期するデータベースに送信することはできません。送信した場合は、次のいずれかのエラーが発生します。

- `BCP failed to create rows in target table.Column %1! would start at an offset over 8191 bytes; this starting location cannot be represented accurately in the table's (row) format.`

- `BCP failed to create rows in target table.Column %1! starts at an offset greater than %2! bytes; this starting location is not permitted by the current database configuration.`

エラーを修正するには、次のどちらかの手順に従います。

- テーブルのロック・スキームをデータオンリーロックから全ページロックに変更します。
- Adaptive Server 15.7 以降に接続している場合は、allow wide dol rows オプションをターゲット・データベースで有効にします。『ASE パフォーマンス&チューニング・シリーズ：物理データベースのチューニング』の「第 2 章 データの格納」を参照してください。

ロー・フォーマットのキャッシュ

Open Client 15.7 は、ロー・フォーマット情報のキャッシュをサポートしています。これにより、クライアント・アプリケーションはデータ・サーバに対し、動的 SQL 文が呼び出されるたびにロー・フォーマット情報を送信しないように要求できます。ロー・フォーマットをキャッシュすると、データ・サーバとクライアント・アプリケーション間のネットワーク・トラフィックが削減されるので、システムのパフォーマンスが向上します。

デフォルトでは、ロー・フォーマットのキャッシュは Open Client 15.7 で有効になります。無効にするには、CS_CMD_SUPPRESS_FMT 応答機能を CS_FALSE に設定します。CS_CMD_SUPPRESS_FMT の値をチェックし、設定するには ct_cmd_props() を使用します。

サーバがロー・フォーマットの省略をサポートしているかどうかを判断するには、ct_capability() を使用して CS_RES_SUPPRESS_FMT の値を確認します。

注意 この機能は、ロー・フォーマットのキャッシュをサポートしているサーバにクライアント・アプリケーションが接続している場合にのみ使用できます。

カーソル・クローズ時のロックの解放のサポート

Open Client 15.7、Open Server 15.7、および Embedded SQL C プロセッサと COBOL 15.7 プロセッサは、Adaptive Server 15.7 に導入された release_locks_on_close カーソル・オプションをサポートしています。この機能を使用すると、カーソルのクローズ時に読み込みロックを解放できます。『ASE リファレンス・マニュアル：コマンド』を参照してください。

Client-Library の使用法

ct_cursor() 構文の *option* パラメータは、CS_CUR_RELLOCKS_ONCLOSE を含めるように拡張されています。このオプションを使用すると、カーソルのクローズ時に共有ロックを解放するよう Adaptive Server に指示できます。読み取り専用カーソルまたはスクロール可能カーソルに使用するには、OR ビット処理演算子、"|" (パイプ) を使用します。

- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_CURSOR
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_SEMISENSITIVE
- CS_CUR_RELLOCKS_ONCLOSE | CS_NOSCROLL_INSENSITIVE

注意 cpre と cobpre は次のオプションを生成できません。

- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE

例

例 1 クローズ時に共有ロックを解放するカーソルを宣言します。

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE);
```

例 2 クローズ時に共有ロックを解放する insensitive スクロール可能カーソルを宣言します。

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE);
```

この機能を示す Open Client サンプル・プログラムについては、*csr_disp_scrollcurs3.c* を参照してください。

Open Server の使用法

クライアント・アプリケーションが CS_CUR_RELLOCKS_ONCLOSE オプションを指定してカーソルを宣言した場合、Open Server では SRV_CURDESC 構造体の curstatus (カーソル・ステータス) フィールドを SRV_CUR_RELLOCKS_ONCLOSE に設定します。

ctos コード例の *cursor.c* の図を参照してください。

ESQL/C と ESQL/COBOL の使用法

ESQL/C と ESQL/COBOL の SQL DECLARE 構文は、RELEASE_LOCKS_ON_CLOSE キーワードを含むように拡張されています。

```
EXEC SQL DECLARE cursor_name
    [SEMI_SENSITIVE | INSENSITIVE]
    [SCROLL | NOSCROLL]
    [RELEASE_LOCKS_ON_CLOSE]
    CURSOR FOR “select stmt”
    [for {read only | update [ of column_name_list]}]
```

次のフォーム以外で RELEASE_LOCKS_ON_CLOSE に UPDATE 句を使用することはできません。

```
EXEC SQL declare cursor c1 release_locks_on_close
   cursor for select * from T for update of col_a
```

この場合、RELEASE_LOCKS_ON_CLOSE は無視されます。

cpre と cobpre は次の ct_cursor() オプションを生成できません。

- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE

ESQL/C コード例は *example8.cp* に含まれており、ESQL/COBOL コード例は *example7.pco* に含まれています。

ストアド・プロシージャ・パラメータとしてのラージ・オブジェクト

Open Client と Open Server 15.7 は、ストアド・プロシージャの入力パラメータおよび動的 SQL 文のパラメータとして text、unitext、image を使用することをサポートしています。

この機能の使用に関するログイン・ネゴシエーションを助長するために、2 つの接続機能が追加されています。

- CS_RPCPARAM_LOB − クライアント・アプリケーションはこの要求機能をサーバに送信し、ラージ・オブジェクト (LOB) データ型をストアド・プロシージャの入力パラメータとして使用できるかどうかを判断します。サーバはこの機能をサポートできないときに、初期ログイン・ネゴシエーションのこの機能ビットをクリアします。ユーザがこのようなサーバに LOB パラメータを送信しようとすると、エラーが発生します。
- CS_RPCPARAM_NOLOB − クライアント・アプリケーションはこの応答機能を送信し、パラメータとして LOB データを送信しないようにサーバに要求します。この機能はデフォルトにより有効になっています。

パラメータとしての少量の LOB データの送信

少量の LOB データをストアド・プロシージャの入力パラメータまたは準備された SQL 文のパラメータとして送信するプロセスは、LOB 以外のパラメータを送信するプロセスと同じです。

少量の LOB データを送信するには、コマンドとデータ用のメモリを割り付け、ct_param() または ct_setparam() を使用してこれらをサーバに直接送信します。

text、unitext、または image パラメータを使用する場合は、CS_DATAFMT 構造体の *maxlength* フィールドを設定する必要があります。*maxlength* 値は、すべての LOB データがサーバに一度に送信されるか、ストリーミングされるかを示します。*maxlength* がゼロより大きい場合、LOB データは 1 つのまとまりで送信されます。*maxlength* が CS_UNUSED に設定されている場合、LOB データはデータをまとまりで送信するための ct_send_data() 呼び出しのループを使用して、ストリームで送信されます。ゼロのまとまりの長さは、データ・ストリームの終わりを示します。

例 1 少量の LOB データをストアド・プロシージャの入力パラメータとして送信します。

```
CS_TEXT textvar[50];
CS_DATAFMT paramfmt;
CS_INT datalen;
CS_SMALLINT ind;

...
ct_command(cmd, CS_RPC_CMD, ...)

/*
** Clear and setup the CS_DATAFMT structure, then pass
** each of the parameters for the RPC.
*/
memset(&paramfmt, 0, sizeof(paramfmt));

/*
** First parameter, an integer.
*/
strcpy(paramfmt.name, "@intparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_INT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
ct_param(cmd, &paramfmt, (CS_VOID *)&intvar,
   sizeof(CS_INT), ind))

/*
** Second parameter, a (small) text parameter.
*/
```

```
strcpy((CS_CHAR *)textvar, "The Open Client and Open
   Server products both include Bulk-Library and
   CS-Library. ");
datalen = sizeof(textvar);
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_TEXT_TYPE;
paramfmt.maxlength = EX_MYMAXTEXTLEN;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
ct_setparam(cmd, &paramfmt, (CS_VOID *)&textvar,
   &datalen, &ind);

ct_send(cmd);
ct_results(cmd, &res_type);

...
```

例 2 少量の LOB データを準備文を使用して送信します。

```
/*
** Prepare the sql statement.
*/
sprintf(statement, "select title_id from mybooks where
   title like (?) ");

/*
** Send the prepared statement to the server
*/
ct_dynamic(cmd, CS_PREPARE, "my_dyn_stmt", CS_NULLTERM,
   statement, CS_NULLTERM);

ct_send(cmd);
handle_results(cmd);

/*
** Prompt user to provide a value for title
*/
printf("Enter title id value - enter an X if you wish
   to stop: \n");

while (toupper(title[0]) != 'X')
{
   printf("Retrieve detail record for title: ?");
   fgets(mytexttitle, 50,  stdin);

   /*
   ** Execute the dynamic statement.
   */

   ct_dynamic(cmd, CS_EXECUTE, "my_dyn_stmt",
   CS_NULLTERM, NULL, CS_UNUSED);
```

```
    /*
   ** Define the input parameter
   */

   memset(&data_format, 0, sizeof(data_format));
   data_format.status = CS_INPUTVALUE;
   data_format.namelen = CS_NULLTERM ;
   data_format.datatype = CS_TEXT_TYPE;
   data_format.format = CS_FMT_NULLTERM;
   data_format.maxlength = EX_MYMAXTEXTLEN;
   ct_setparam(cmd, &data_format,
      (CS_VOID *)mytexttitle, &datalen, &ind);


ct_send(cmd);
handle_results(cmd);
...
}
```

パラメータとしての大量の LOB データの送信

大量の LOB データは、リソースを効率的に管理するために、ストリームでサーバに送信されます。ct_send_data() をループ内で使用して、まとまったデータをサーバに送信します。

LOB データ・パラメータをまとまりで送信するには、次の設定を使用してパラメータを定義します。

- CS_DATAFMT 構造体の *datatype* フィールドを CS_TEXT_TYPE、CS_UNITEXT_TYPE、または CS_IMAGE_TYPE に設定します。
- CS_DATAFMT 構造体の *maxlength* フィールドを CS_UNUSED に設定します。
- ct_param() 関数の **data* ポインタ引数を NULL に設定します。
- ct_param() 関数の *datalen* 引数を 0 に設定します。

例 1 大量の LOB データ・パラメータをまとまりで送信します。

```
#define BUFSIZE 2048

int fp;
char sendbuf[BUFSIZE]

/*
** Clear and setup the CS_DATAFMT structure, then pass
** each of the parameters for the RPC.
*/
memset(&paramfmt, 0, sizeof(paramfmt));
strcpy(paramfmt.name, "@intparam");
```

```
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_INT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

ct_param(cmd, &paramfmt, (CS_VOID *)&intvar,
   sizeof(CS_INT), 0))

/*
** Text parameter, sent as a BLOB.
*/
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_TEXT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

/*
** Although the actual data will not be sent here, we
** must invoke ct_setparam() for this parameter to send
** the parameter format (paramfmt) information to the
** server, prior to sending all parameter data.
** Set *data to NULL and datalen = 0, to indicate that
** the length of text data is unknown and we want to
** send it in chunks to the server with ct_send_data().
*/
ct_setparam(cmd, &paramfmt, NULL, 0, 0);

/*
** Another LOB parameter (image), sent in chunks with
** ct_send_data()
*/
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_IMAGE_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

/*
** Just like the previous parameter, invoke
** ct_setparam() for this parameter to send the
** parameter format.
*/
ct_setparam(cmd, &paramfmt, NULL, 0, 0);

/*
** Repeat this sequence of filling paramfmt and calling
** ct_param() for any subsequent parameter that needs
```

```
** to be sent before finally sending the data chunks for
** the LOB type parameters.
*/
strcpy(paramfmt.name, "@any_otherparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_MONEY_TYPE;
...



/*
** Send the first LOB (text) parameter in chunks of
** 'BUFSIZE' to the server.We must end with a 0 bytes
** write to indicate the end of the current parameter.
*/
fp = open("huge_text_file", O_RDWR, 0666);

do
{
   num_read = read(fp, sendbuf, BUFSIZE);
   ct_send_data(cmd, (CS_VOID *)sendbuf, num_read);
}  while (num_read != 0);

/*
** Repeat the ct_send_data() loop for the next LOB
** parameter.
** Send the image parameter in chunks of 'BUFSIZE'
** to the server as well and end with a 0 bytes write
** to indicate the end of the current parameter.
*/
fp = open("large_image_file", O_RDWR, 0666);
do
{
   num_read = read(fp, sendbuf, BUFSIZE);
   ct_send_data(cmd, (CS_VOID *)sendbuf, num_read);
} while (num_read != 0);

/*
** Ensure that all the data is flushed to the server
*/
ct_send(cmd);
```

例 2 準備された SQL 文を使用して、ストリームとして LOB データを送信します。

```
/*
** Prepare the sql statement.
*/
sprintf(statement, "select title_id from mybooks
   where title like (?) ");

/*
```

```
** Send the prepared statement to the server
*/
ct_dynamic(cmd, CS_PREPARE, "mydyn_stmt", CS_NULLTERM,
   statement, CS_NULLTERM);

ct_send(cmd);
handle_results();

/*
** Promt user to provide a value for title
*/
printf("Enter title id value - enter an X if you wish
   to stop: \n");

while (toupper(myblobtitle[0]) != 'X')
{
   printf("Retrieve detail record for title: ?");
   fgets(myblobtitle, 50,  stdin);

   /*
   ** Execute the dynamic statement.
   */
   ct_dynamic(cmd, CS_PREPARE, "my_dyn_stmt",
   CS_NULLTERM, statement, CS_NULLTERM);

   /*
   ** Define the input parameter, a TEXT type that we
   want to send in chunks to the server.
   */
   memset(&data_format, 0, sizeof(data_format)) ;
   data_format.namelen = CS_NULLTERM ;
   data_format.datatype = CS_TEXT_TYPE;
   data_format.maxlength = CS_UNUSED;
   data_format.status = CS_INPUTVALUE;
   ct_setparam(cmd, &data_format, NULL, 0, 0);

   /*
   ** Send the 'myblobtitle' data in chunks of
   ** 'CHUNKSIZE' to the server with ct_send_data() and
   ** end with 0 bytes to indicate the end of data for
   ** this parameter.This is just an example to show
   ** how chunks can be sent.(myblobtitle[] is used as
   ** a simple example.This could also be replaced by
   ** large file which would be read in chunks from disk
   ** for example).
   */
   bytesleft = strlen(myblobtitle);
   bufp = myblobtitle;

   do
   {
```

```
            sendbytes = min(bytesleft, CHUNKSIZE);


            ct_send_data(cmd, (CS_VOID *)bufp, sendbytes);
            bufp += bufp + sendbytes;
            bytesleft -= sendbytes;
        }  while (bytesleft > 0)

        /*
        ** End with 0 bytes to indicate the end of current
        data.
        */
        ct_send_data(cmd, (CS_VOID *)bufp, 0);

        /*
        ** Insure that all the data is sent to the server.
        */
        ct_send(cmd);
        handle_results(cmd)
        ...
    }

    /*
    ** Deallocate the prepared statement and finish up.
    */

    ct_dynamic(cmd, CS_DEALLOC, "my_dyn_stmt", CS_NULLTERM,
        NULL, CS_UNUSED);

    ct_send(cmd);
    handle_results(cmd);
```

Open Server での LOB パラメータの取得

Open Server は、完全な LOB パラメータ・データを srv_xferdata() を使用して一度に取得するか、新しい srv_get_data() ルーチンを使用してまとまりで取得できます。Open Server は、パラメータの長さが CS_UNUSED に設定されている場合に、LOB パラメータをまとまりで取得します。srv_get_data() の詳細については、「srv_get_data」(32 ページ) を参照してください。

例 LOB パラメータの説明を取得します。

```
/*
** Retrieve the description of the parameters coming
** from client
*/

for (paramnum = 1; paramnum <= numparams; paramnum++)
{
    /*
    ** Get a description of the parameter.
```

```
*/
ret = srv_descfmt(spp, CS_GET, SRV_RPCDATA,
   paramnum, &(paramfmtp[paramnum - 1]));

/*
** Allocate space for the parameters and bind the
** data.
*/
if (paramfmtp[paramnum-1].maxlength >= 0)
{
   if (paramfmtp[paramnum-1].maxlength > 0)
   {
      data[paramnum-1] = calloc(1,
         paramfmtp[paramnum-1].maxlength);
   }
   else
   {
      ind[paramnum-1] = CS_NULLDATA;
   }
}
else
{
   /*
   ** Allocate a large size buffer for BLOB data
   ** (which length is unknown yet)
   */
   blobbuf[blobnum] = malloc(BUFSIZE);
   blobnum++;
}

srv_bind(spp, CS_GET, SRV_RPCDATA, paramnum,
   &(paramfmtp[paramnum-1]), data[paramnum-1],
   &(len[paramnum-1]), &(ind[paramnum-1]))

/*
** For every LOB parameter, call srv_get_data() in
** a loop as long as it succeeds
/*
for (i = 0; i < blobnum ; i++)
{
   bufp = blobbuf[i];
   bloblen[i] = 0;
   do
   {

      ret = srv_get_data(spp, bufp, BUFSIZE,
         &outlen);
      bufp += outlen;
      bloblen[i] += outlen;
   } while (ret == CS_SUCCEED);
```

```
            /*
            ** Check for the correct return code
            */
            if (ret != CS_END_DATA)
            {
               return CS_FAIL;
            }

         }

         /*
         ** And receive remaining client data srv_xferdata()
         */
         ret = srv_xferdata(spp, CS_GET, SRV_RPCDATA);
      }
```

srv_get_data

text、unitext、または image パラメータ・ストリームをまとまった単位でクライアントから読み込みます。

構文

```
CS_RETCODE srv_get_data(spp, bp, buflen, outlenp)

SRV_PROC *spp;
CS_BYTE *bp;
CS_INT buflen;
CS_INT *outlenp;
```

パラメータ

- *spp* − 内部スレッド制御構造体を指すポインタ。
- *bp* − クライアントからのデータが格納されているバッファを指すポインタ。
- *buflen* − **bp* ポインタのサイズ。このサイズは、連続して転送される 1 つのデータのまとまりをバイト数で示したものです。
- *outlenp* − **bp* バッファに読み込まれるバイト数を含む出力パラメータ。

戻り値

- CS_SUCCEED − srv_get_data() が正常に実行されました。保留中のデータがさらにあります。
- CS_FAIL − ルーチンが失敗しました。
- CS_END_DATA − srv_get_data() が text、unitext、または image パラメータ全体の読み取りを完了しました。

jConnect および Adaptive Server のドライバおよびプロバイダに対応する SDK 15.7 機能

この項では、jConnect、Adaptive Server ODBC ドライバ、Adaptive Server OLE DB プロバイダ、Adaptive Server ADO.NET Data Provider に導入された新機能について説明します。

ODBC ドライバのバージョン情報ユーティリティ

odbcversion ユーティリティは、ODBC ドライバに関する情報を表示します。

構文

odbcversion
-version |
-fullversion |
-connect *dsn userid password*

パラメータ

-version

ODBC ドライバの単純な数値のバージョン文字列を表示します。

-fullversion

ODBC ドライバの冗長バージョン文字列を表示します。

-connect *dsn userid password*

Adaptive Server のバージョンと、その Adaptive Server にインストールされている ODBC および OLEDB MDA スクリプトのバージョンを表示します。このパラメータには 3 つの変数が必要です。これらは、Adaptive Server のデータ・ソース名である *dsn* と、Adaptive Server への接続に使用されるユーザ ID およびパスワードです。

例

Adaptive Server への接続に使用される ODBC ドライバの単純な数値のバージョン文字列を取得します。

```
odbcversion -version
```

数値バージョン文字列が返されます。

```
15.05.00.1015
```

使用法

パラメータが指定されていない場合、odbcversion ユーティリティは有効なパラメータのリストを表示します。

SupressRowFormat2 接続文字列プロパティ

ADO.NET、OLE DB、または ODBC ドライバでは、SupressRowFormat2 接続文字列プロパティを使用して、Adaptive Server が TDS_ROWFMT2 バイト・シーケンスの代わりに TDS_ROWFMT バイト・シーケンスをできる限り使用してデータを送信することを強制できます。TDS_ROWFMT は、カタログ、スキーマ、テーブル、カラム情報を含む TDS_ROWFMT2 よりも少ないデータを含んでおり、多くの小さな select オペレーションでより優れたパフォーマンスを実現します。SupressRowFormat2 が 1 に設定されている場合、サーバは縮小された結果セット・メタデータを送信するので、一部の情報がクライアント・プログラムで使用できなくなります。欠如しているメタデータにアプリケーションが依存している場合、このプロパティは有効にしないでください。

値：

- 0 − デフォルト値。TDS_ROWFMT2 は抑制されません。
- 1 − サーバがデータをできる限り TDS_ROWFMT で送信するように強制します。

例

この接続文字列は、ADO.NET Data Provider との接続において、サーバがデータをできる限り TDS_ROWFMT で送信するように強制します。

```
Data Source='myASE';Port=5000;Database=myDB;
Uid=myUID;Pwd=myPWD;SupressRowFormat2=1
```

UseCusror プロパティの機能強化

Adaptive Server ODBC ドライバの UseCursor 接続文字列プロパティを使用すると、サーバ側のカーソルが、結果セットを生成する SQL 文に対してどのように使用されるかを判断できます。このプロパティは、サーバ側のカーソル (値 2) を作成する文をクライアント・アプリケーションが制御できるように更新されています。

値：

- 0 − デフォルト値。サーバ側のカーソルは使用されません。
- 1 − サーバ側のカーソルが、結果セットを生成するすべての文に使用されます。
- 2 − SQLSetCursorName ODBC 関数が呼び出されたときにのみ、結果セットを生成する文にサーバ側のカーソルが使用されます。カーソルはより多くのリソースを使用するので、この設定を使用すると、サーバ側カーソルから恩恵を得られる文のみにその使用を制限できます。

ODBC ドライバ・マネージャのトレースなしのロギング

Adaptive Server ODBC ドライバ 15.7 では、ODBC ドライバ・マネージャのトレースを使用せずに ODBC API の呼び出しをロギングできます。これは、ドライバ・マネージャが使用されないか、トレースをサポートしていないプラットフォームでドライバ・マネージャを実行している場合に便利です。

この機能を Microsoft Windows で有効にするには、LOGCONFIGFILE 環境変数または Microsoft Windows レジストリを使用します。Linux で有効にするには、LOGCONFIGFILE を使用します。

LOGCONFIGFILE を使用するときは、環境変数を ODBC ログの設定ファイルのフル・パスに設定します。LOGCONFIGFILE は既存のレジストレ・エントリをすべて上書きします。

Microsoft Windows レジストリを使用する場合は、LogConfigFile というエントリを *HKEY_CURRENT_USER*¥*Software*¥*Sybase*¥*ODBC* または *HKEY_LOCAL_MACHINE*¥*Software*¥*Sybase*¥*ODBC* に作成し、その値を ODBC ログの設定ファイルのフル・パスに設定します。次に例を示します。

```
Windows レジストリ・エディタ・バージョン 5.00

[HKEY_CURRENT_USER¥Software¥Sybase¥ODBC]
"LogConfigFile"="c:¥¥temp¥¥odbclog.properties"
```

ロギングを無効にするには、*LogConfigFile* の値を削除するか、名前を変更します。

注意 *HKEY_CURRENT_USER* に指定されている値は、*HKEY_LOCAL_MACHINE* に設定されている値を上書きします。

ログ設定ファイル

設定ファイルは ODBC ログ・ファイルのフォーマットと場所を制御します。この例では、太字の行がログ・ファイルの保存先を指定します。

```
log4cplus.rootLogger=OFF, NULL

log4cplus.logger.com.sybase.dataaccess.odbc.api=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.parameter=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.parameter=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.returncode=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.returncode=false

log4cplus.appender.NULL=log4cplus::NullAppender

log4cplus.appender.ODBCTRACE=log4cplus::FileAppender
```

```
log4cplus.appender.ODBCTRACE.File=c:¥temp¥odbc.log
log4cplus.appender.ODBCTRACE.layout=log4cplus::PatternLayout
log4cplus.appender.ODBCTRACE.ImmediateFlush=true
log4cplus.appender.ODBCTRACE.layout.ConversionPattern=%d{%H:%M:%S.%q} %t %p
   %-25.25c{2} %m%n
```

jConnect setMaxRows の機能強化

JDBC プログラムでは Statement.setMaxRows(int max) を使用して、結果セットが返すロー数を制限します。jConnect 7.0 以前では、select、insert、update、delete 文の結果が制限に対してカウントされます。

JDBC の仕様との一貫性を保つために、jConnect 7.07 には SETMAXROWS_AFFECTS_SELECT_ONLY 接続プロパティが導入されています。このプロパティは、true (デフォルト) に設定されている場合に SELECT 文が返すローのみを制限します。

Adaptive Server 15.5 以前に接続している場合、SETMAXROWS_AFFECTS_SELECT_ONLY は無視されます。

TDS ProtocolCapture

Adaptive Server ODBC ドライバ 15.7 では、ODBC アプリケーションと Adaptive Server 間で交換される Tabular Data Stream™ (TDS) パケットを受信するためのファイルを指定する ProtocolCapture 接続文字列プロパティを導入しています。ProtocolCapture の設定はすぐに反映されるので、接続の確立中に交換された TDS パケットはファイル・プレフィクスを使用して生成された一意のファイル名に書き込まれます。TDS パケットは、接続している期間中、ファイルに書き込まれます。TDS 取得ファイルを解釈するには、Ribo および他のプロトコル変換ツールを使用できます。

たとえば、*tds_capture* を TDS トレース・ログ・ファイル・プレフィクスとして指定するには、次のように入力します。

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;ProtocolCapture=tds_capture;
```

最初の接続は *tds_capture0.tds* を生成し、2 番目の接続は *tds_capture1.tds* を生成します (以下同様)。

バインド・パラメータ配列を使用しない ODBC データのバッチ処理

同じ SQL 文が異なるパラメータ値に対して実行される場合、クライアント・アプリケーションは通常パラメータ配列をバインドし、SQLExecute、SQLExecuteDirect、SQLBulkOperations を使用してパラメータの各セットを実行します。配列を SQL パラメータにバインドする際は、配列のメモリが割り付けられ、データがすべて配列にコピーされてから、SQL 文が実行されます。これにより、大量のトランザクションを処理する場合にメモリとリソースの使用効率が低下することがあります。

Adaptive Server ODBC ドライバ 15.7 ではクライアント・アプリケーションは SQLExecute を使用して、パラメータを配列としてバインドせずに、パラメータを Adaptive Server にバッチで送信します。SQLExecute は、最後のパラメータのバッチが送信および処理されるまで、SQL_BATCH_EXECUTING を返します。最後のパラメータのバッチが処理されると、実行ステータスが返されます。

SQLRowCount の呼び出しは、最後の SQLExecute 文が完了した後でのみ有効です。

データ・バッチの管理

Adaptive Server に送信されるパラメータのバッチを管理するために、Sybase 固有の接続属性である SQL_ATTR_BATCH_PARAMS が導入されています。SQL_ATTR_BATCH_PARAMS は SQLSetConnectAttr を使用して設定します。

値：

- SQL_BATCH_ENABLED － パラメータをバッチ処理するように Adaptive Server ODBC ドライバに伝えます。この状態では、現在処理中の文、つまり SQL_ATTR_BATCH_PARAMS を SQL_BATCH_ENABLED に設定した後に SQLExecute によって実行される最初の文以外の文が接続で実行されると、エラーが送信されます。
- SQL_BATCH_LAST_DATA － 次のパラメータのバッチが最後のバッチであり、パラメータにデータが含まれていることを指定します。
- SQL_BATCH_LAST_NO_DATA － 次のパラメータのバッチが最後のバッチであり、これらのパラメータを無視するように指定します。
- SQL_BATCH_CANCEL － バッチをキャンセルし、トランザクションをロールバックするように Adaptive Server ODBC ドライバに伝えます。

  ロールバックできるのは、コミットされていないトランザクションのみです。
- SQL_BATCH_DISABLED － (デフォルト値) Adaptive Server ODBC ドライバは、最後のパラメータのバッチを処理した後に、この状態に戻ります。SQL_ATTR_BATCH_PARAMS をこの値に手動で設定することはできません。

例

例 1 パラメータ配列をバインドせずにパラメータのバッチをサーバに送信します。

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to start
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
   (SQLPOINTER)SQL_BATCH_ENABLED, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Bind the parameters.This can be done once for the entire batch
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
   SQL_C_LONG, SQL_INTEGER, l1, 0, &c1, l1, &l1);
sr = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_LONGVARCHAR, l2, 0, buffer, l2, &l2);
}

// Run a batch of 10 for (int i = 0; i < 10; i++)
{
   c1 = i;
   memset(buffer, 'a'+i, l2);
   sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
   printError(sr, SQL_HANDLE_STMT, stmt);
}
```

例 2 バッチを終了して閉じます。

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to end
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
  (SQLPOINTER)SQL_BATCH_LAST_NO_DATA, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Call SQLExecDirect one more time to close the batch
// - Due to SQL_BATCH_LAST_NO_DATA, this will not
// process the parameters
sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
```

注意すべき点

- この機能は、結果セットを返さないか、出力パラメータを持たない文およびストアド・プロシージャのみをサポートしています。
- 非同期モードはサポートされていません。バッチ・モード中に、現在バッチ処理している文以外の文をアプリケーションが同じ接続で実行することはできません。
- SQL_DATA_AT_EXEC はサポートされていません。LOB パラメータは通常のパラメータとしてバインドします。

- パラメータ配列をバインドせずにデータをバッチ処理する場合に、SQL_ATTR_PARAM_STATUS_PTR が設定されているとき、Adaptive Server ODBC ドライバでは SQL_ATTR_PARAMSET_SIZE ではなく、SQLSetStmtAttr の StringLength パラメータから複数の配列要素を取得します。

jConnect での最適化されたバッチ処理

jConnect for JDBC 7.07 では PreparedStatement オブジェクトのバッチ・オペレーションを高速化するための内部アルゴリズムを実装しています。このアルゴリズムは HOMOGENEOUS_BATCH 接続プロパティが true に設定されている場合に呼び出されます。

注意 同種バッチ処理は、この機能をサポートしているサーバにクライアント・アプリケーションが接続している場合にのみ使用できます。Adaptive Server Enterprise 15.7 には同種バッチ処理のサポートが導入されています。

次の例は、addBatch および executeBatch メソッドを使用した PreparedStatement バッチ・オペレーションを示します。

```
String sql = "update members set lastname = ?
where member_id = ?";
prep_stmt = connection.prepareStatement(sql);

prep_stmt.setString(1, "Forrester");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();

prep_stmt.setString(1, "Robinson");
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();

prep_stmt.setString(1, "Servo");
prep_stmt.setLong(2, 45131);
prep_stmt.addBatch();

prep_stmt.executeBatch();
```

ここで、connection は接続インスタンスで、prep_stmt は準備文インスタンスで、? は準備文のパラメータ用プレースホルダを表します。

LOB カラムの同種バッチ処理

HOMOGENEOUS_BATCH および ENABLE_LOB_LOCATORS プロパティが true に設定されている場合、クライアント・アプリケーションが同じバッチ内で LOB 準備文と LOB 以外の準備文の setter メソッドを混合することはできません。たとえば、以下は無効です。

```
String sql = "update members SET catchphrase = ?WHERE member_id
= ?";
prep_stmt = connection.prepareStatement(sql);

prep_stmt.setString(1, "Push the button, Frank!");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();

Clob myclob = con.createClob();
myclob.setString(1, "Hi-keeba!");
prep_stmt.setClob(1, myclob);
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();

pstmt.executeBatch();
```

ここでのキャッチフレーズは、カラムのデータ型 text です。setString メソッドと setClob メソッドが同じカラムに対して同じバッチで使用されているので、このコードは失敗します。

ローを累積しない jConnect パラメータのバッチ処理

jConnect for JDBC 7.07 には SEND_BATCH_IMMEDIATE 接続プロパティが追加されています。これが true に設定されている場合、jConnect は PreparedStatement.addBatch() を呼び出した直後に現在のローのパラメータを送信します。これにより、クライアントのメモリの使用が最小化され、サーバにはバッチ・パラメータを処理するための時間がより多く与えられます。

デフォルトの SEND_BATCH_IMMEDIATE 値は false で、これが設定されている場合はこれまでと同様に、PreparedStatement.executeBatch() を呼び出した後にのみバッチ・パラメータを送信するように jConnect に示します。

過去のエラーを実行するための jConnect バッチ更新の機能強化

jConnect 7.07 には EXECUTE_BATCH_PAST_ERRORS 接続プロパティが導入されています。これが true に設定されている場合は、個々の文の実行中に遭遇した致命的でないエラーをバッチ更新オペレーションが無視し、バッチ更新を完了できるようにします。デフォルトの false に設定されている場合は、以前のバージョンのように、エラーに遭遇するとバッチ更新は中止されます。

バッチ更新の使用法とその結果の解釈については、『jConnect for JDBC プログラマーズ・リファレンス』を参照してください。

カーソル・クローズ時のロックの解放のサポート

Adaptive Server 15.7 では、release_locks_on_close オプションを含めるように declare cursor 構文が拡張されています。このオプションは、カーソルのクローズ時に独立性レベル 2 および 3 でカーソルの共有ロックを解放します。Adaptive Server ODBC ドライバ 15.7 と jConnect for JDBC 7.07 は、release-lock-on-close セマンティックをサポートしています。

この機能を Adaptive Server ODBC Driver 接続で作成されたすべての読み取り専用カーソルに適用するには、ReleaseLocksOnCursorClose 接続プロパティを 1 に設定します。デフォルトの ReleaseLocksOnCursorClose 値は 0 です。

jConnect 接続で適用するには、RELEASE_LOCKS_ON_CURSOR_CLOSE 接続プロパティを true に設定します。デフォルトの RELEASE_LOCKS_ON_CURSOR_CLOSE 値は false です。

これらの接続プロパティによって適用された設定は静的で、接続の確立後に変更することはできません。この設定は、release_locks_on_close をサポートしているサーバに接続されている場合にのみ有効です。

release_locks_on_close の詳細については、『ASE リファレンス・マニュアル：コマンド』を参照してください。

select for update のサポート

Adaptive Server 15.7 は、同じトランザクション内の後続の更新用にローをロックできる select for update と、更新可能なカーソル用の排他ロックをサポートしています。『ASE Transact-SQL ユーザーズ・ガイド』の「第 2 章 クエリ：テーブルからのデータの選択」を参照してください。

この機能は、for update 句が select 文に追加されたときと、クライアント内で開いている更新可能なカーソルに追加されたときにクライアントで自動的に使用可能になります。更新可能なカーソルの作成の詳細については、『ASE ODBC ドライバ・ユーザーズ・ガイド』および『jConnect for JDBC プログラマーズ・リファレンス』を参照してください。

拡張された可変長ローのサポート

16K 論理ページ・サイズを使用するように設定されている Adaptive Server 15.7 より前のバージョンでは、可変長カラムが行の先頭から 8191 バイトを超えた位置から始まっている場合、可変長ローを含むデータオンリー・ロック (DOL) テーブルは作成できませんでした。この制限は、Adaptive Server 15.7 以降では取り除かれています。『ASE パフォーマンス&チューニング・シリーズ：物理データベースのチューニング』の「第 2 章 データの格納」を参照してください。

ODBC および JDBC クライアントがこの機能を使用するにあたって特別な設定を行う必要はありません。長い DOL ローを受信するように設定されている Adaptive Server バージョン 15.7 に接続すると、これらのクライアントは長いオフセットを使用して自動的にレコードを挿入します。クライアントが長い DOL ローを以前のバージョンの Adaptive Server に接続しようとするか、長い DOL ロー・オプションが無効になっている 15.7 Adaptive Server に接続しようとすると、エラー・メッセージが送信されます。

非実体化カラムのサポート

Adaptive Server ODBC ドライバのバルク挿入ルーチンは、Adaptive Server 15.7 の非実体化カラムを処理できるように強化されています。以前のバージョンの Adaptive Server ODBC ドライバでは、テーブル定義に非実体化カラムが含まれている場合、Adaptive Server へのデータのバルク挿入は実行できません。以前のバージョンの Adaptive Server ODBC ドライバで非実体化カラムにバルク挿入を実行しようとすると、エラーが発生します。

ロー内とロー外の LOB 記憶領域のサポート

Adaptive Server 15.7 では、ロー内に格納するようにマークされている LOB カラムは、ロー全体を格納するのに十分なメモリがある場合、ロー内に格納されます。ロー内のカラムが更新されたために、ローのサイズが定義されている制限を超えた場合、ロー内に格納されている LOB カラムは制限を満たすためにロー外に移動されます。『ASE Transact-SQL ユーザーズ・ガイド』の「第 21 章 ロー内／ロー外の LOB」を参照してください。

Adaptive Server ODBC Driver 15.7 および jConnect for JDBC 7.07 のバルク挿入ルーチンは、Adaptive Server の text、image、unitext の LOB カラムのロー内およびロー外の記憶領域をサポートしています。以前のクライアント・バージョンからのバルク挿入ルーチンでは、LOB カラムは常にロー外に格納されます。

ストアド・プロシージャ・パラメータとしてのラージ・オブジェクト

ストアド・プロシージャ入力パラメータとして LOB データを渡す機能も Adaptive Server 15.7 に導入されています。

jConnect for JDBC 7.07、Adaptive Server ODBC ドライバ 15.7、Adaptive Server Enterprise OLE DB プロバイダ 15.7、Adaptive Server Enterprise ADO.NET Data Provider 15.7 は、ストアド・プロシージャ内での入力パラメータ、およびパラメータ・マーカ・データ型としての text、unitext、image の使用をサポートしています。

ラージ・オブジェクトのロケータのサポート

jConnect for JDBC 7.07 および Adaptive Server ODBC ドライバ 15.7 は、ラージ・オブジェクト (LOB) ロケータをサポートしています。LOB ロケータには、データ自体ではなく、LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。LOB ロケータに対するサーバのサポートは、Adaptive Server 15.7 以降に導入されています。

jConnect for JDBC 7.07 では、LOB ロケータをサポートする Adaptive Server に接続しており、autocommit がオフになっている場合に、サーバ側のロケータを使用して LOB データにアクセスします。それ以外の場合、jConnect はクライアント側で LOB データを実体化します。クライアント側で実体化された LOB データを完全な LOB API で使用することはできますが、データ量が大きくなるため、LOB ロケータを使用した場合よりも API のパフォーマンスが低下することがあります。

Adaptive Server ODBC ドライバ 15.7 クライアントは、LOB ロケータをサポートしている Adaptive Server に接続していない限り、LOB ロケータを使用できません。

注意 LOB ロケータを使用している場合、各ローに LOB データを含む大きな結果セットを取得すると、アプリケーションのパフォーマンスに影響が及ぶ場合があります。Adaptive Server では LOB ロケータを結果セットの一部として返します。LOB データを取得するには、jConnect と Adaptive Server ODBC ドライバが残りの結果セットをキャッシュに格納する必要があります。結果セットは小さいサイズを保持するか、カーソルのサポートを有効にしてキャッシュに格納するデータのサイズを制限することをおすすめします。

---

jConnect のサポート

LOB ロケータのサポートを有効にするには、ENABLE_LOB_LOCATORS 接続プロパティを true に設定して Adaptive Server への接続を確立します。有効にすると、クライアント・アプリケーションは java.sql パッケージの Blob、Clob、NClob クラスを使用して、ロケータにアクセスできるようになります。

注意 LOB ロケータと autocommit の両方が有効になっている場合、jConnect では Adaptive Server でのサポートが可能であっても、クライアント側で実体化されている LOB ロケータに自動的に切り替えます。これにより、クライアントが使用するメモリが増加し、パフォーマンスが低下する場合があります。したがって、autocommit off の状態で LOB ロケータを使用することをおすすめします。

Blob、Clob、NClob クラスの詳細については、Java のマニュアルを参照してください。

Adaptive Server ODBC ドライバのサポート

LOB ロケータのサポートを Adaptive Server ODBC ドライバで有効にするには、EnableLOBLocator 接続プロパティを 1 に設定して Adaptive Server への接続を確立します。When EnableLOBLocator がデフォルト値である 0 に設定されている場合、Adaptive Server ODBC ドライバは LOB カラムのロケータを取得できません。LOB ロケータを有効にする場合は、接続を autocommit off に設定してください。

sybasesqltypes.h ファイルをプログラムに含める必要もあります。*sybasesqltypes.h* ファイルは ODBC インストール・ディレクトリの下の *include* ディレクトリに含まれています。

ロケータをサポートするための ODBC データ型のマッピング

Adaptive Server ロケータのデータ型に対する ODBC データ型のマッピングは次のとおりです。

ASE データ型	ODBC SQL 型	ODBC C 型
text_locator	SQL_TEXT_LOCATOR	SQL_C_TEXT_LOCATOR
image_locator	SQL_IMAGE_LOCATOR	SQL_C_ IMAGE_LOCATOR
unitext_locator	SQL_UNITEXT_LOCATOR	SQL_C_ UNITEXT_LOCATOR

サポートされている変換

Adaptive Server ロケータのデータ型に対してサポートされている変換は次のとおりです。

	SQL_C_TEXT_ LOCATOR	SQL_C_IMAGE_ LOCATOR	SQL_C_UNITEXT_ LOCATOR
SQL_TEXT_LOCATOR	X		
SQL_IMAGE_LOCATOR		X	
SQL_UNITEXT_LOCATOR			X
SQL_LONGVARCHAR			
SQL_WLONGVARCHAR			
SQL_LONGVARBINARY			

凡例：X = サポートされている変換

LOB ロケータをサポートしている ODBC API メソッド

- SQLBindCol － *TargetType* には ODBC C ロケータの任意のデータ型を指定できます。
- SQLBindParameter － *ValueType* には ODBC C ロケータの任意のデータ型を指定できます。*ParameterType* には ODBC SQL ロケータの任意のデータ型を指定できます。
- SQLGetData － *TargetType* には ODBC C ロケータの任意のデータ型を指定できます。

- SQLColAttribute − SQL_DESC_TYPE および SQL_DESC_CONCISE_TYPE 記述子は、ODBC SQL ロケータの任意のデータ型を返すことができます。
- SQLDescribeCol − データ型のポインタには、ODBC SQL ロケータの任意のデータ型を指定できます。

『Microsoft ODBC API Reference』を参照してください。

プリフェッチされた LOB データの暗黙的変換

Adaptive Server ODBC ドライバ 15.7 では、Adaptive Server が LOB ロケータを返した場合、SQLGetData および SQLBindCol を使用して、text ロケータ用の SQL_C_CHAR または SQL_C_WCHAR、あるいは image ロケータ用の SQL_C_BINARY にカラムをバインドすることで、基本となるプリフェッチされた LOB データを取得できます。

接続内のロケータを有効または無効にするには、SQL_ATTR_LOBLOCATOR 属性を設定します。EnableLOBLocator が接続文字列内で指定されている場合、SQL_ATTR_LOBLOCATOR は EnableLOBLocator の値で初期化されます。それ以外の場合はデフォルト値である SQL_LOBLOCATOR_OFF に設定されます。ロケータを有効にするには、属性を SQL_LOBLOCATOR_ON に設定します。属性の値を設定するには SQLSetConnectAttr を使用し、属性の値を取得するには SQLGetConnectAttr を使用します。

SQLSetStatementAttr を使用して SQL_ATTR_LOBLOCATOR_FETCHSIZE を設定し、取得する LOB データのサイズを指定します。バイナリ・データのサイズはバイト数で指定し、文字データのサイズは文字数で指定します。デフォルト値の 0 は、プリフェッチされたデータが要求されないことを示し、-1 の値は LOB データ全体が取得されることを示します。

注意 取得するカラムの基本となる LOB データのサイズが、設定されているプリフェッチされたデータのサイズを超えた場合は、ODBC クライアントがデータを直接取得しようとしたときにネイティブ・エラー 3202 が発生します。これが発生した場合、クライアントは SQLGetData を呼び出すことで完全なデータを取得し、基本となるロケータを取得して、ロケータで利用できるオペレーションをすべて実行できます。

例 1 プリフェッチされたデータが完全な LOB 値を表しているときに、SQLGetData を使用して image ロケータを取得します。

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR, (SQLPOINTER)SQL_LOCATOR_ON,
   0);
```

```
// Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt, SQL_ATTR_LOBLOCATOR_FETCHSIZE, (SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);

printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);

sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

//Retrieve the binary data  (Complete Data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);
printError(sr, SQL_HANDLE_STMT, stmt);


//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR, (SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

例 2 プリフェッチされたデータがトランケートされた LOB 値を表しているときに、SQLGetData を使用して image ロケータを取得します。

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
   (SQLPOINTER)SQL_LOCATOR_ON, 0);
```

```
// Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt,
   SQL_ATTR_LOBLOCATOR_FETCHSIZE, (SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);
printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

// Retrieve the binary data(Truncated data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);

if(sr == SQL_SUCCESS_WITH_INFO)
{
   SQLTCHAR errormsg[ERR_MSG_LEN];
   SQLTCHAR sqlstate[SQL_SQLSTATE_SIZE+1];
   SQLINTEGER nativeerror = 0;
   SQLSMALLINT errormsglen = 0;

   retcode = SQLGetDiagRec(handleType, handle, 1, sqlstate, &nativeerror,
      errormsg, ERR_MSG_LEN, &errormsglen);

   printf("SqlState: %s Error Message: %s\n", sqlstate, errormsg);

   //Handle truncation of LOB data; if data was truncated call SQLGetData to
   // retrieve the locator.

   /* Warning returns truncated LOB data */
   if (NativeError == 32028) //Error code may change
   {
      BYTE ImageLocator[SQL_LOCATOR_SIZE];
      sr = SQLGetData(stmt, 1, SQL_C_IMAGE_LOCATOR, &ImageLocator,
         sizeof(ImageLocator), &Len);
      printError(sr, SQL_HANDLE_STMT, stmt);
```

```
      /*
         Perform locator specific calls using image Locator on a separate


         statement handle if needed
      */
   }
}

//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR, (SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

ロケータを使用した LOB のアクセスと操作

ODBC API は LOB ロケータを直接サポートしていません。ODBC クライアント・アプリケーションでは Transact-SQL® 関数を使用して、ロケータに対するオペレーションを行い、LOB 値を操作する必要があります。Adaptive Server ODBC ドライバには、必要な Transact-SQL 関数の使用を助長するためのストアド・プロシージャがいくつか導入されています。

この項では、LOB ロケータに対してさまざまなオペレーションを実行する方法について説明します。パラメータの入出力値には、Adaptive Server がストアド・プロシージャ定義に暗黙的に変換できる任意の型を指定できます。

ここに示す Transact-SQL コマンドおよび関数の詳細については、『ASE リファレンス・マニュアル：ビルディング・ブロック』の「Transact-SQL 関数」を参照してください。

text ロケータの初期化

sp_drv_create_text_locator を使用して text_locator を作成し、オプションでその値を初期化します。sp_drv_create_text_locator は、Transact-SQL 関数 create_locator にアクセスします。

構文　　sp_drv_create_text_locator [*init_value*]

入力パラメータ　　*init_value* － 新しいロケータの初期化に使用される varchar または text の値。

出力パラメータ　　なし。

結果セット　　text_locator 型のカラム。ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

unitext ロケータの初期化

sp_drv_create_unitext_locator を使用して unitext_locator を作成し、オプションでその値を初期化します。sp_drv_create_unitext_locator は、Transact-SQL 関数 create_locator にアクセスします。

構文

sp_drv_create_unitext_locator [*init_value*]

入力パラメータ

init_value − 新しいロケータの初期化に使用される univarchar または unitext。

出力パラメータ

なし。

結果セット

unitext_locator 型のカラム。ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

image ロケータの初期化

sp_drv_create_image_locator を使用して image_locator を作成し、オプションでその値を初期化します。sp_drv_create_image_locator は、Transact-SQL 関数 create_locator にアクセスします。

構文

sp_drv_create_image_locator [*init_value*]

入力パラメータ

init_value − 新しいロケータの初期化に使用される varbinary または image。

出力パラメータ

なし。

結果セット

image_locator 型のカラム。ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

text ロケータからの完全な *text* 値の取得

Transact-SQL 関数 return_lob にアクセスする sp_drv_locator_to_text を使用します。

構文

sp_drv_locator_to_text *locator*

入力パラメータ

locator − 値の取得対象となる text_locator。

出力パラメータ

なし。

結果セット

locator によって参照される text 値を含むカラム。

unitext ロケータからの完全な *unitext* 値の取得

Transact-SQL 関数 return_lob にアクセスする sp_drv_locator_to_unitext を使用します。

構文

sp_drv_locator_to_unitext *locator*

入力パラメータ

locator − 値の取得対象となる unitext_locator。

出力パラメータ

なし。

結果セット

locator によって参照される unitext 値を含むカラム。

image ロケータからの完全な *image* 値の取得

Transact-SQL 関数 return_lob にアクセスする sp_drv_locator_to_image を使用します。

構文

sp_drv_locator_to_image *locator*

入力パラメータ

locator − 値の取得対象となる image_locator。

出力パラメータ

なし。

結果セット

locator によって参照される image 値を含むカラム。

text ロケータからの部分文字列の取得

Transact-SQL 関数 substring にアクセスする sp_drv_text_substring を使用します。

構文

sp_drv_text_substring *locator*, *start_position*, *length*

入力パラメータ

- *locator* − 操作するデータを参照する text_locator。
- *start_position* − 読み込んで取得する最初の文字の位置を指定する integer。
- *length* − 読み込む文字数を指定する integer。

出力パラメータ

なし。

結果セット

取得された部分文字列を含む text 型のカラム。

unitext ロケータからの部分文字列の取得

Transact-SQL 関数 substring にアクセスする sp_drv_unitext_substring を使用します。

構文

sp_drv_unitext_substring *locator*, *start_position*, *length*

入力パラメータ

- *locator* − 操作するデータを参照する unitext_locator。
- *start_position* − 読み込んで取得する最初の文字の位置を指定する integer。
- *length* − 読み込む文字数を指定する integer。

出力パラメータ

なし。

結果セット

取得された部分文字列を含む unitext 型のカラム。

image ロケータからの部分文字列の取得

Transact-SQL 関数 substring にアクセスする sp_drv_image_substring を使用します。

構文

sp_drv_image_substring *locator*, *start_position*, *length*

入力パラメータ

- *locator* − 操作するデータを参照する image_locator。
- *start_position* − 読み込んで取得する最初のバイトの位置を指定する integer。
- *length* − 読み込むバイト数を指定する integer。

出力パラメータ

なし。

結果セット

取得された部分文字列を含む image 型のカラム。

指定した位置への *text* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_text_setdata を使用します。

構文

sp_drv_text_setdata *locator*, *offset*, *new_data*, *data_length*

入力パラメータ

- *locator* − 挿入先の text カラムを参照する text_locator。
- *offset* − 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data* − 挿入する varchar または text データ。

出力パラメータ

data_length − 書き込まれた文字数を含む integer。

結果セット

なし。

指定した位置への *unitext* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_unitext_setdata を使用します。

構文

sp_drv_unitext_setdata *locator*, *offset*, *new_data*, *data_length*

入力パラメータ

- *locator* − 挿入先の unitext カラムを参照する unitext_locator。
- *offset* − 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data* − 挿入する univarchar または unitext データ。

出力パラメータ

data_length − 書き込まれた文字数を含む integer。

結果セット

なし。

指定した位置への *image* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_image_setdata を使用します。

構文

sp_drv_image_setdata *locator*, *offset*, *new_data*, *datalength*

入力パラメータ

- *locator* – 挿入先の image カラムを参照する image_locator。
- *offset* – 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data* – 挿入する varbinary または image データ。

出力パラメータ

data_length – 書き込まれたバイト数を含む integer。

結果セット

なし。

ロケータが参照する *text* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_text_locator_setdata を使用します。

構文

sp_drv_text_locator_setdata *locator*, *offset*, *new_data_locator*, *data_length*

入力パラメータ

- *locator* – 挿入先の text カラムを参照する text_locator。
- *offset* – 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data_locator* – 挿入先の text データを参照する text_locator。

出力パラメータ

data_length – 書き込まれた文字数を含む integer。

結果セット

なし。

ロケータが参照する *unitext* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_unitext_locator_setdata を使用します。

構文

sp_drv_unitext_locator_setdata *locator*, *offset*, *new_data_locator*, *data_length*

入力パラメータ

- *locator* – 挿入先の unitext カラムを参照する unitext_locator。
- *offset* – 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data_locator* – 挿入先の unitext データを参照する unitext_locator。

出力パラメータ

data_length – 書き込まれた文字数を含む integer。

結果セット

なし。

ロケータが参照する *image* の挿入

Transact-SQL 関数 setadata にアクセスする sp_drv_image_locator_setdata を使用します。

構文

sp_drv_image_locator_setdata *locator*, *offset*, *new_data_locator*, *datalength*

入力パラメータ

- *locator* − 挿入先の image カラムを参照する image_locator。
- *offset* − 新しいコンテンツの書き込み開始位置を指定する integer。
- *new_data_locator* − 挿入先の image データを参照する image_locator。

出力パラメータ

data_length − 書き込まれたバイト数を含む integer。

結果セット

なし。

基本となる LOB データのトランケート

truncate lob を使用して、LOB ロケータが参照している LOB データをトランケートします。『ASE リファレンス・マニュアル：コマンド』を参照してください。

基本となる *text* データの文字長の確認

sp_drv_text_locator_charlength を使用して、text ロケータが参照している LOB カラムの文字長を確認します。sp_drv_text_locator_charlength では Transact-SQL 関数 char_length にアクセスします。

構文

sp_drv_text_locator_charlength *locator*, *data_length*

入力パラメータ

locator − 操作する text カラムを参照する text_locator。

出力パラメータ

data_length − *locator* が参照する text カラムの文字長を指定する integer。

結果セット

なし。

基本となる *text* データのバイト長の確認

sp_drv_text_locator_bytelength を使用して、text ロケータが参照している LOB カラムのバイト長を確認します。sp_drv_text_locator_bytelength では Transact-SQL 関数 data_length にアクセスします。

構文

sp_drv_image_locator_bytelength *locator*, *data_length*

入力パラメータ

locator − 操作する text カラムを参照する text_locator。

出力パラメータ

data_length − *locator* が参照する text カラムのバイト長を指定する integer。

結果セット

なし。

基本となる *unitext* データの文字長の確認

sp_drv_unitext_locator_charlength を使用して、unitext ロケータが参照している LOB カラムの文字長を確認します。sp_drv_unitext_locator_charlength では Transact-SQL 関数 char_length にアクセスします。

構文

sp_drv_unitext_locator_charlength *locator*, *data_length*

入力パラメータ

locator − 操作する unitext カラムを参照する unitext_locator。

出力パラメータ

data_length − *locator* が参照する unitext カラムの文字長を指定する integer。

結果セット

なし。

基本となる *unitext* データのバイト長の確認

sp_drv_unitext_locator_bytelength を使用して、unitext ロケータが参照している LOB カラムのバイト長を確認します。sp_drv_unitext_locator_bytelength では Transact-SQL 関数 data_length にアクセスします。

構文

sp_drv_image_locator_bytelength *locator*, *data_length*

入力パラメータ

locator − 操作する unitext カラムを参照する unitext_locator。

出力パラメータ

data_length − *locator* が参照する unitext カラムのバイト長を指定する integer。

結果セット

なし。

基本となる *image* データのバイト長の確認

sp_drv_image_locator_bytelength を使用して、image ロケータが参照している LOB カラムのバイト長を確認します。sp_drv_image_locator_bytelength では Transact-SQL 関数 data_length にアクセスします。

構文

sp_drv_image_locator_bytelength *locator*, *data_length*

入力パラメータ

locator − 操作する image カラムを参照する image_locator。

出力パラメータ

data_length − *locator* が参照する image カラムのバイト長を指定する integer。

結果セット

なし。

ロケータが参照する *text* カラム内の検索文字列位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_varchar_charindex を使用します。

構文

sp_drv_varchar_charindex *search_string*, *locator*, *start*, *position*

入力パラメータ

- *search_string* − 検索対象となる varchar 型のリテラル。
- *locator* − 検索元の text カラムを参照する text_locator。
- *start* − 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* － *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する integer。

結果セット　なし。

別のロケータが参照している *text* カラム内の *text* ロケータが参照している文字列の位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_textlocator_charindex を使用します。

構文

```
sp_drv_textlocator_charindex search_locator, locator, start, position
```

入力パラメータ

- *search_locator* － 検索するリテラルを指す text_locator。
- *locator* － 検索元の text カラムを参照する text_locator。
- *start* － 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* － *locator* が参照する LOB カラム内のリテラルの開始位置を指定する integer。

結果セット　なし。

ロケータが参照する *unitext* カラム内の検索文字列位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_univarchar_charindex を使用します。

構文

```
sp_drv_univarchar_charindex search_string, locator, start, position
```

入力パラメータ

- *search_string* － 検索対象となる univarchar 型のリテラル。
- *locator* － 検索元の unitext カラムを参照する unitext_locator。
- *start* － 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* － *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する integer。

結果セット　なし。

別のロケータが参照している *unitext* カラム内の *unitext* ロケータが参照している文字列の位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_unitext_locator_charindex を使用します。

構文

```
sp_drv_charindex_unitextloc_in_locator search_locator, locator, start,
position
```

入力パラメータ

- *search_locator* － 検索するリテラルを指す unitext_locator。
- *locator* － 検索元の text カラムを参照する unitext_locator。
- *start* － 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* − *locator* が参照する LOB カラム内のリテラルの開始位置を指定する integer。

結果セット　なし。

image ロケータが参照するカラム内のバイト・シーケンス位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_varbinary_charindex を使用します。

構文

```
sp_drv_varbinary_charindex byte_sequence, locator, start, position
```

入力パラメータ

- *byte_sequence* − 検索対象の varbinary 型のバイト・シーケンス。
- *locator* − 検索元の image カラムを参照する image_locator。
- *start* − 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*position* − *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する integer。

結果セット　なし。

別のロケータが参照している *image* カラム内の *image* ロケータが参照しているバイト・シーケンス位置の確認

Transact-SQL 関数 charindex にアクセスする sp_drv_image_locator_charindex を使用します。

構文

```
sp_drv_image_locator_charindex sequence_locator, locator, start, start_position
```

入力パラメータ

- *sequence_locator* − 検索対象のバイト・シーケンスを指す image_locator。
- *locator* − 検索元の image カラムを参照する image_locator。
- *start* − 検索の開始位置を指定する整数。最初の位置は 1 になります。

出力パラメータ　*start_position* − *locator* が参照する LOB カラム内のバイト・シーケンスの開始位置を指定する integer。

結果セット　なし。

text_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする sp_drv_text_locator_valid を使用します。

構文

```
sp_drv_text_locator_valid locator
```

入力パラメータ　*locator* − 検証する text_locator。

出力パラメータ　次の値のいずれかを表す bit。

- 0 − false。*locator* は無効です。
- 1 − true。*locator* は有効です。

結果セット　なし。

unitext_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする sp_drv_unitext_locator_valid を使用します。

構文

sp_drv_unitext_locator_valid *locator*

パラメータ

locator － 検証する unitext_locator。

出力パラメータ

次の値のいずれかを表す bit。

- 0 － false。*locator* は無効です。
- 1 － true。*locator* は有効です。

結果セット

なし。

image_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする sp_drv_image_locator_valid を使用します。

構文

sp_drv_image_locator_valid *locator*

パラメータ

locator － 検証する image_locator。

出力パラメータ

次の値のいずれかを表す bit。

- 0 － false。*locator* は無効です。
- 1 － true。*locator* は有効です。

結果セット

なし。

LOB ロケータの解放または割り付け解除

deallocate locator を使用します。『ASE リファレンス・マニュアル：コマンド』を参照してください。

例

例 1 ハンドルを割り付け、接続を確立します。

```
// Assumes that DSN has been named "sampledsn" and
// UseLobLocator has been set to 1.

SQLHENV environmentHandle = SQL_NULL_HANDLE;
SQLHDBC connectionHande = SQL_NULL_HANDLE;
SQLHSTMT statementHandle = SQL_NULL_HANDLE;
SQLRETURN ret;

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &environmentHandle);
SQLSetEnvAttr(environmentHandle, SQL_ATTR_ODBC_VERSION, SQL_ATTR_OV_ODBC3);
SQLAllocHandle(SQL_HANDLE_DBC, environmentHandle, &connectionHandle);
Ret = SQLConnect(connectionHandle, "sampledsn",
   SQL_NTS, "sa", SQL_NTS, "Sybase",SQL_NTS);
```

例 2 カラムを選択し、ロケータを取得します。

```
// Selects and retrieves a locator for bk_desc, where
// bk_desc is a column of type text defined in a table
// named books. bk_desc contains the text "A book".

SQLPrepare(statementHandle, "SELECT bk_desc FROM books
   WHERE bk_id =1", SQL_NTS);

SQLExecute(statementHandle);
BYTE TextLocator[SQL_LOCATOR_SIZE];
SQLLEN Len = 0;
ret = SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);

If(Len == sizeof(TextLocator))
{
   Cout << Locator was created with expected size <<
   Len;
}
```

例 3 データ長を判別します。

```
SQLLEN LocatorLen = sizeof(TextLocator);
ret = SQLBindParameter(statementHandle, 1,
   SQL_PARAM_INPUT, SQL_C_TEXT_LOCATOR,
   SQL_TEXT_LOCATOR, SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), &LocatorLen);

SQLLEN CharLenSize = 0;
SQLINTEGER CharLen = 0;
ret = SQLBindParameter(statementHandle, 2,
SQL_PARAM_OUTPUT, SQL_C_LONG,SQL_INTEGER,0 , 0,
&CharLen, sizeof(CharLen), &CharLenSize);
SQLExecDirect(statementHandle,
   "{CALL sp_drv_text_locator_charlength( ?,?) }" , SQL_NTS);

cout<< "Character Length of Data " << charLen;
```

例 4 テキストを LOB カラムに追加します。

```
SQLINTEGER retVal = 0;
SQLLEN Col1Len = sizeof(retVal);
SQLCHAR appendText[10]=”abcdefghi on C++”;

SQLBindParameter(statementHandle, 14,
   SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &retVal, 0, Col1Len);

SQLBindParameter(statementHandle, 21, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, &TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);
```

```
SQLBindParameter(statementHandle, 32, SQL_PARAM_INPUT,
   SQL_C_SLONG, SQL_INTEGER, 0, 0, &charLen, 0, SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 43, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 10, 0, append_text,
   sizeof(append_text), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle,
   "{? = CALL sp_drv_setdata_text (?, ?, ?,?) }" , SQL_NTS);

SQLFreeStmt(statementHandle, SQL_CLOSE);
```

例 5 LOB データを LOB ロケータから取得します。

```
SQLCHAR description[512];
SQLLEN descriptionLength = 512;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle, "{CALL sp_drv_locator_to_text(?)}", SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, 1,SQL_C_CHAR, description,
   descriptionLength, &descriptionLength)

Cout << "LOB data referenced by locator:" << description
   << endl;

Cout << "Expected LOB data:A book on C++" << endl;
```

例 6 クライアント・アプリケーションのデータを LOB ロケータに転送します。

```
description = "A lot of data that will be used for a lot
   of inserts, updates and deletes"; descriptionLength = SQL_NTS;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 512, 0, description,
   sizeof(description), &descriptionLength);

SQLExecDirect(statementHandle,
   "{CALL sp_drv_create_text_locator(?)}", SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);
```

Adaptive Server Enterprise (拡張モジュール Python 版)

Python 用の Adaptive Server Enterprise 拡張モジュールは、Adaptive Server データベースに対してクエリを実行するための Sybase 固有の Python インタフェースです。このモジュールは、拡張機能の付いた Python データベース API 仕様バージョン 2.0 を実装し、Python バージョン 2.6 で使用します。Python データベース API の仕様については、http://www.python.org/dev/peps/pep-0249 を参照してください。

Python 用の Adaptive Server Enterprise 拡張モジュールは、SDK インストーラからインストールできます。インストール手順については、『Software Developer's Kit/Open Server インストール・ガイド』と、『Software Developer's Kit/Open Server リリース・ノート』を参照してください。Python 用 Adaptive Server Enterprise 拡張モジュールの使用については、『Python 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』を参照してください。

Adaptive Server Enterprise (拡張モジュール PHP 版)

PHP 用の Adaptive Server Enterprise 拡張モジュールは、Adaptive Server データベースに対するクエリを実行し、クエリ結果を処理するためのインタフェースで、データベースへのアクセスに必要な PHP API を備えています。このモジュールは PHP バージョン 5.3.6 で使用します。PHP 用 Adaptive Server Enterprise 拡張モジュールの使用については、『PHP 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』を参照してください。

Perl 用 Adaptive Server Enterprise データベース・ドライバ

Perl 用の Adaptive Server Enterprise データベース・ドライバは、汎用 Perl DBI インタフェースで呼び出され、CT-Lib を使用した Open Client SDK 経由で、Perl DBI API 呼び出しを Adaptive Server が理解できる形式に変換します。これにより、Perl スクリプトは Adaptive Server Enterprise データベース・サーバに直接アクセスできるようになります。このドライバは、Perl バージョン 5.14 および DBI バージョン 1.616 で使用します。

Perl DBI の仕様については、http://search.cpan.org/~timb/DBI-1.616/DBI.pm を参照してください。Perl 用 Adaptive Server Enterprise データベース・ドライバの使用については、『Perl 用 Adaptive Server Enterprise データベース・ドライバ・プログラマーズ・ガイド』を参照してください。

非推奨機能

次の機能はこのリリースの Open Server と SDK ではサポートされていません。

DCE サービス・ライブラリ

分散コンピューティング環境 (DCE) ディレクトリ・サービス・ライブラリ *libsybddce.dll* と DCE セキュリティ・サービス・ライブラリ *libsybsdce.dll* は、Windows 32 ビット・プラットフォーム用の Open Client および Open Server から削除されています。15.7 より前のバージョンの Open Client と Open Server では、これらのライブラリは *%SYBASE%¥OCS-15_0¥dll* ディレクトリに含まれていました。

dsedit_dce ユーティリティ・ファイル

dsedit_dce X-Windows のデフォルト・ファイルである *OCS-15_0/xappdefaults/Dsedit_dce* と dsedit_dce ヘルプ・ファイルである *OCS-15_0/sybhelp/dsedit_dceHelpTextMsgs* は削除されています。

サポートされていないプラットフォーム

Open Server と SDK では HP-UX PA-RISC と Mac OS をサポートしていません。

アクセシビリティ機能

第 508 条では、米国連邦機関の電子および情報技術は身体障害者が利用できるものと規定されています。Sybase は、第 508 条を強力にサポートし、Open Client および Open Server バージョン 15.7 を含む広範な製品を、第 508 条に準拠させています。

リリース 15.7 のマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。Open Client および Open Server のマニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。